UNIX Programmer's Manual

Part 1
Commands and Application Programs

N. Plat

editor

Delft University of Technology
Faculty of Mathematics and Informatics
March 1987

Western Electric

Patent Licensing

Graham Center
P.O. Box [illegible]
Greensboro, NC 27420
919 697 [illegible]

OCT 03 1980

TECHNISCHE HOGESCHOOL DELFT
Department of Mathematics
Juianalaan 132
2628 BL Delft
The Netherlands

Attn: Mr. P. J. van der Hoff

Gentlemen:

Re: May 1, 1979 Software Agreement Between Us
Relating to PWB/UNIX* Time Sharing Operating
System

In response to the August 22, 1980 request from Mr. van der Hoff, your institution may use the licensed software pursuant to the referenced agreement on the following additional CPU's:

LSI-11, Serial No. WM 1720
PDP 11/60, Serial No. AG 00073
Technische Hogeschool Delft
Department of Mathematics - Comp. Rm. 0.101
Julianalaan 132
2628 BL Delft
The Netherlands

Yours truly,

O. L. WILSON
Patent Licensing Manager

*UNIX is a trademark of Bell Laboratories.

PREFACE

These manuals are intended to describe the UNIX® systems as used on the PDP-11/73 and the Ada Geminix at the Delft University of Technology, Faculty of Mathematics and Informatics.

I welcome any suggestions that might render future editions of this manual a more accurate reflection of these systems.

Delft, March 1987
the editor

Knowledge is of two kinds:
We know a subject ourselves, or we know where we can find information upon it

Samuel Johnson

® UNIX is a registered trademark of AT&T in the USA and other countries

INTRODUCTION

This manual describes the features of ULTRIX-11 V2.0 (used on the PDP-11/73) and of the UNIX version used on the Ada Geminix, which is derived from UNIX V7. It provides neither a general overview of UNIX (for that, see The UNIX Time-Sharing System, *Comm. ACM* 17(7):365-75, July 1974, by D.M. Ritchie and K. Thompson), nor details of the implementation of the system.

This manual is divided into eight sections:

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

Section I (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory */bin* (for *bin*ary programs). Some programs also reside in */usr/bin*, to save space in */bin*. The so-called TU-added commands (commands added to the original systems at the Delft University of Technology) reside in */usr/thd* on the PDP-11/73 and in */usr/bin/tud* on the Geminix. These directories are searched automatically by the command interpreter called the Cshell.

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

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

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

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

Section VI (*Games*) describes some games available on the systems. All the games reside in */usr/bin/games*.

Section VII (*Miscellaneous*) describes miscellaneous information of the UNIX System.

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

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

The center part of each page header contains one of the following:

PDP/GMX	This entry is both available on the PDP-11/73 and on the Geminix.
GMX ONLY	This entry is only available on the Geminix.
PDP ONLY	This entry is only available on the PDP-11/73.
GMX	This entry differs from the version on the PDP-11/73.
PDP	This entry differs from the version on the Geminix.

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

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

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

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

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

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

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

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

The DESCRIPTION part discusses in detail the subject at hand.

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

The SEE ALSO part gives pointers to related information.

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

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

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

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

TABLE OF CONTENTS

PERMUTED INDEX

asctime, timezone(3): convert date and time to ASCII
ascii(7): map of ASCII character set
atof, atoi, atol(3): convert ASCII to numbers
ascii(7): map of ASCII character set
ctime, localtime, gmtime, asctime, timezone(3): convert date and time to
sin, cos, tan, asin, acos, atan, atan2(3): trigonometric
help(1): ask for help
as (1): assembler
a.out(5): assembler and link editor output
m11(1): Macro-11 assembler for UNIX
assert(3): program verification
setbuf(3): assign buffering to a stream
at(1): execute commands at a later time
sin, cos, tan, asin, acos, atan, atan2(3): trigonometric functions
sin, cos, tan, asin, acos, atan, atan2(3): trigonometric functions
atof, atoi, atol(3): convert ASCII to numbers
atof, atoi, atol(3): convert ASCII to numbers
atof, atoi, atol(3): convert ASCII to numbers
yacc_prep(1): pre-process some attributes for YACC
wait(1): await completion of process
awk(1): pattern scanning and processing
banner(6): print large banner on printer
banner(6): make long posters
banner(6): print large banner on printer
bas(1): basic
bas(1): basic
bc(1): arbitrary-precision arithmetic language
bdiff(1): big diff
cb(1): C program beautifier
j0, j1, jn, y0, y1, yn(3): bessel functions
the printable strings in a object, or other binary, file
fread, fwrite(3): buffered binary input/output
sync(1): update the super block
sum(1): sum and count blocks in a file
qhul (1): bold filter for the qume printer
ching, fortune(6): the book of changes and other cookies
brk, sbrk, break(2): change core allocation
brk, sbrk, break(2): change core allocation
fread, fwrite(3): buffered binary input/output
stdio(3): standard buffered input/output package
setbuf(3): assign buffering to a stream
cc, pcc(1): C compiler
cb(1): C program beautifier
lint(1): a C program verifier
xstr(1): extract strings from C programs to implement shared strings
create an error message file by massaging C source
hypot, cabs(3): euclidean distance
dc(1): desk calculator
calendar(1): reminder service
indir(2): indirect system call
cu_v7(1): call UNIX (original V7 version)
malloc, free, realloc, calloc(3): main memory allocator
intro, errno(2): introduction to system calls and error numbers
croff, diroff(1): formatters for the Canon Laserprinter
dint, dcan(1): print diroff-output on the Canon Laserprinter
termcap(5): terminal capability data base
ccat(1): compress and uncompress files, and cat them

comm(1): select or reject lines common to two sorted files
users(1): compact list of users who are on the system
compact, uncompact, ccat(1): compress and
diff(1): differential file comparator
cmp(1): compare two files
sccsdiff(1): compare two versions of an SCCS file
diff3(1): 3-way differential file comparison
regex, regcmp(3): regular expression compile/execute
cc, pcc(1): C compiler
das(1): das compiler
f77(1): Fortran 77 compiler
ack(1): Amsterdam Compiler Kit
yacc(1): yet another compiler-compiler
wait(1): await completion of process
compact, uncompact, ccat(1): compress and uncompress files, and cat them
learn(1): computer aided instruction about UNIX
link(1): link to a remote computer via a terminal line/modem
test(1): condition command
tip, cu(1): connect to a remote system
deroff(1): remove nroff, troff, tbl and eqn constructs
ls(1): list contents of directory
uustat(1): uucp status inquiry and job control
vc(1): version control
ioctl, stty, gtty(2): control device
intro(3): summary of job control facilities
zaptty(1): run a program without a controlling terminal.
zaptty(2): zap the controlling tty
ecvt, fcvt, gcvt(3): output conversion
printf, fprintf, sprintf(3): formatted output conversion
scanf, fscanf, sscanf(3): formatted input conversion
units(1): conversion program
dd(1): convert and copy a file
arcv(1): convert archives to new format
ranlib(1): convert archives to random libraries
atof, atoi, atol(3): convert ASCII to numbers
l3tol, ltol3(3): convert between 3-byte integers and long
ctime, localtime, gmtime, asctime, timezone(3): convert date and time to ASCII
getdate(3): convert time and date from ASCII
fortune(6): the book of changes and other cookies
cp(1): copy
uucp, uulog(1): unix to unix copy
public UNIX-to-UNIX file copy
dd(1): convert and copy a file
copy(1): copy a subtree of a directory
cptree(1): copy a subtree of a filesystem
cpio(1): copy file archives in and out
copy(1): copy a subtree of a directory
brk, sbrk, break(2): change core allocation
core(5): format of core image file
core(5): format of core image file
sin, cos, tan, asin, acos, atan, atan2(3):
sinh, cosh, tanh(3): hyperbolic functions
wc(1): word count
sum(1): sum and count blocks in a file
cp(1): copy
cpio(1): copy file archives in and out
cptree(1): copy a subtree of a filesystem

ed(1): text editor
ex, edit(1): text editor
iie(1): interactive ingres editor
sed(1): stream editor
vi(1): screen oriented (visual) display editor based on ex
em(1): editor for mortals
a.out(5): assembler and link editor output
whoami(1): print effective current user id
grep, egrep, fgrep(1): search a file for a pattern
em(1): editor for mortals
crypt(1): encode/decode
crypt, encrypt(3): a one way hashing encryption
crypt, encrypt(3): a one way hashing encryption algorithm
end, etext, edata(3): last locations in program
getgrent, getgrgid, getgrnam, setgrent, endgrent(3): get group file entry
getpwent, getpwuid, getpwnam, setpwent, endpwent
nlist(3): get entries from name list
chfn(1): change finger entry
getgrnam, setgrent, endgrent(3): get group file entry
unlink(2): remove directory entry
chghist(1): change the history entry of an SCCS delta
execle, execve, execlp, execvp, exec, exece, environ (2): execute a file
environ(5): user environment
environ(5): user environment
printenv(1): print out the environment
getenv(3): value for environment name
eqnchar(7): special character definitions for eqn
deroff(1): remove nroff, troff, tbl and eqn constructs
eqn (PDP only), neqn, checkeq (PDP only) (1): eqnchar(7): special character definitions for
intro, errno(2): introduction to system calls and
mkstr(1): create an error message file by massaging C source
perror, sys_errlist, sys_nerr(3): system error messages
errno(2): introduction to system calls and error numbers
fperr(2): get floating point error status
spell, spellin, spellout(1): find spelling errors
plot: openpl et al.(3): graphics interface
end, etext, edata(3): last locations in program
hypot, cabs(3): euclidean distance
expr(1): evaluate arguments as an expression
ex, edit(1): text editor
execl, execv, execle, execve, execlp, execvp, exec, exece, environ (2): execute a file
execv, execle, execve, execlp, execvp, exec, exece, environ (2): execute a file
execl, execv, execle, execve, execlp, execvp, exec, exece,
execl, execv, execle, execve, execlp, execvp, exec, exece,
execl, execv, execle, execve, execlp, execvp, exec, exece, environ (2): execute a file
at(1): execute commands at a later time
uux(1): unix to unix command execution
sleep(1): suspend execution for an interval
sleep(3): suspend execution for interval
monitor(3): prepare execution profile
profil(2): execution time profile
execl, execv, execle, execve, execlp, execvp, exec, exece, environ
execl, execv, execle, execve, execlp, execvp, exec, exece, environ
execl, execv, execle, execve, execlp, execvp, exec, exece, environ (2): execute a
exit(2): terminate process

exp, log, log10, pow, sqrt(3): exponential,
expand(1): expand tabs to spaces
expand(1): expand tabs to spaces
frexp, ldexp, modf(3): split into mantissa and exponent
exp, log, log10, pow, sqrt(3): exponential, logarithm, power, square root
expr(1): evaluate arguments as an expression
expr(1): evaluate arguments as an expression
regex, regcmp(3): regular expression compile/execute
xstr(1): extract strings from C programs to implement
kill(1): terminate a process with extreme prejudice
f77(1): Fortran 77 compiler
fabs, floor, ceil(3): absolute value, floor,
intro(3): summary of job control facilities
factor, primes(1): factor a number, generate large primes
factor, primes(1): factor a number, generate
true, false(1): provide truth values
abort(3): generate IOT fault
opend, fopend, closed, fclosd(3): generic device open/close routines
fclose, fflush(3): close or flush a stream
ecvt, fcvt, gcvt(3): output conversion
fopen, freopen, fdopen(3): open a stream
feof, ferror, clearerr, fileno(3): stream
feof, ferror, clearerr, fileno(3): stream status
dbminit, fetch, store, delete, firstkey, nextkey(3):
fclose, fflush(3): close or flush a stream
getc, getchar, fgetc, getw(3): get character or word from
gets, fgets(3): get a string from a stream
grep, egrep, fgrep(1): search a file for a pattern
cut(1): cut out selected fields of each line of a file
access(2): determine accessibility of file
chmod(2): change mode of file
chown(2): change owner and group of a file
close (2): close a file
core(5): format of core image file
creat (2): create a new file
ctags(1): create a tags file
cut out selected fields of each line of a file
dd(1): convert and copy a file
make a delta (change) to an SCCS file
execvp, exec, exece, environ (2): execute a file
link(2): link to a file
mknod(2): make a directory or a special file
passwd(5): password file
of several files or subsequent lines of one file
pr (1): print file
prs(1): print an SCCS file
read(2): read from file
rev(1): reverse lines of a file
rmdel(1): remove a delta from an SCCS file
compare two versions of an SCCS file
size(1): size of an object file
printable strings in a object, or other binary, file
sum(1): sum and count blocks in a file
tail(1): deliver the last part of a file
touch(1): update date last modified of a file
undo a previous get of an SCCS file
uniq(1): report repeated lines in a file

fold(1): fold long lines for finite width
fopen, freopen, fdopen(3): open a stream
opend, fopend, closed, fclosd(3): generic device
fork (2): spawn new process
ar(5): archive (library) file format
arcv(1): convert archives to new format
tar(5): tape archive format
core(5): format of core image file
tbl(1): format tables for nroff or troff
scanf, fscanf, sscanf(3): formatted input conversion
printf, fprintf, sprintf(3): formatted output conversion
croff, diroff(1): formatters for the Canon Laserprinter
nroff, troff(1): text formatting and typesetting
forth(1): forth interpreter
forth(1): forth interpreter
f77(1): Fortran 77 compiler
ratfor(1): rational Fortran dialect
struct(1): structure Fortran programs
ching, fortune(6): the book of changes and other
fperr(2): get floating point error status
printf, fprintf, sprintf(3): formatted output
fpsim(2): report or change the status of
putc, putchar, fputc, putw(3): put character or word on a
puts, fputs(3): put a string on a stream
fread, fwrite(3): buffered binary input/output
malloc, free, realloc, calloc(3): main memory allocator
fopen, freopen, fdopen(3): open a stream
frexp, ldexp, modf(3): split into mantissa and
from(1): who is my mail from?
scanf, fscanf, sscanf(3): formatted input conversion
fseek, ftell, rewind(3): reposition a stream
stat, fstat(2): get file status
fseek, ftell, rewind(3): reposition a stream
time, ftime(2): get date and time
floor, ceil(3): absolute value, floor, ceiling functions
intro(3): introduction to library functions
j0, j1, jn, y0, y1, yn(3): bessel functions
tan, asin, acos, atan, atan2(3): trigonometric functions
sinh, cosh, tanh(3): hyperbolic functions
curses(3): screen functions with "optimal" cursor motion
fread, fwrite(3): buffered binary input/output
ecvt, fcvt, gcvt(3): output conversion
tree(1): generate a nice listing of a UNIX file tree
abort(3): generate IOT fault
factor, primes(1): factor a number, generate large primes
rand, srand(3): random number generator
report(1): report generator for Ingres
lex(1): generator of lexical analysis programs
opend, fopend, closed, fclosd(3): generic device open/close routines
getc, getchar, fgetc, getw(3): get character or
getc, getchar, fgetc, getw(3): get character or word
getdate(3): convert time and date from
getuid, getgid, geteuid, getegid(2): get user and group identity
getenv(3): value for environment name
getuid, getgid, geteuid, getegid(2): get user and group
getuid, getgid, geteuid, getegid(2): get user and group
getgrent, getgrgid, getgrnam, setgrent,

lorder(1): find ordering relation for an object
nice, nohup(1): run a command at low priority
lpq(1): show lineprinter queue
lpr(1): line printer spooler
lprm(1): delete jobs from the lineprinter queue
ls(1): list contents of directory
lseek, tell(2): move read/write pointer
l3tol, ltol3(3): convert between 3-byte integers and
m11(1): Macro-11 assembler for UNIX
m4(1): macro processor
m4(1): macro processor
m11(1): Macro-11 assembler for UNIX
man(7): macros to typeset manual
mail, rmail(1): send mail to users or read mail
mail(1): send and receive mail
from(1): who is my mail from?
mail, rmail(1): send mail to users or read mail
mail, rmail(1): send mail to users or read mail
mail(1): send and receive mail
malloc, free, realloc, calloc(3): main memory allocator
make(1): maintain program groups
make(1): maintain program groups
malloc, free, realloc, calloc(3): main memory
man(1): find manual information by keywords;
man(1): print sections of this manual
man(7): macros to typeset manual
nostk(2): allow process to manage its own stack
sighold, sigignore, sigrelse, sigpause(3): manage signals
ingres(1): relational data base management system
tp(1): manipulate tape archive
frexp, ldexp, modf(3): split into mantissa and exponent
manual information by keywords; print out the manual
man(1): print sections of this manual
man(7): macros to typeset manual
man(1): find manual information by keywords; print out the
memstat(1): print memory usage map
ascii(7): map of ASCII character set
umask(2): set file creation mode mask
mkstr(1): create an error message file by massaging C source
only), neqn, checkeq (PDP only) (1): typeset mathematics
lock(2): lock a process in primary memory
malloc, free, realloc, calloc(3): main memory allocator
memstat(1): print memory usage map
memstat(1): print memory usage map
sort(1): sort or merge files
paste(1): merge same lines of several files or subsequent
mesg (1): permit or deny messages
mkstr(1): create an error message file by massaging C source
mesg (1): permit or deny messages
perror, sys_errlist, sys_nerr(3): system error messages
mkdir (1): make a directory
mknod(2): make a directory or a special file
mkstr(1): create an error message file by
mktemp(3): make a unique file name
chmod(1): change mode
umask(2): set file creation mode mask
chmod(2): change mode of file

tset, reset(1): set terminal modes
frexp, ldexp, modf(3): split into mantissa and exponent
touch(1): update date last modified of a file
monitor(3): prepare execution profile
more, page(1): file perusal filter for crt
em(1): editor for mortals
screen functions with "optimal" cursor motion
mount, umount(2): mount or remove file system
mount, umount(2): mount or remove file system
mv (1): move or rename files and directories
lseek, tell(2): move read/write pointer
mv (1): move or rename files and directories
id(1): print user and group IDs and names
eqn (PDP only), neqn, checkeq (PDP only) (1): typeset
newtty(4): summary of the 'new' tty driver
newgrp(1): log in to a new group
checknews(1): check to see if user has news
postnews(1): submit news articles
readnews(1): read news articles
newtty(4): summary of the 'new' tty driver
dbminit, fetch, store, delete, firstkey, nextkey(3): data base subroutines
tree(1): generate a nice listing of a UNIX file tree
nice, nohup(1): run a command at low priority
nice(2): set program priority
nlist(3): get entries from name list
nm (1): print name list
nice, nohup(1): run a command at low priority
setjmp, longjmp(3): non-local goto
nostk(2): allow process to manage its own stack
tbl(1): format tables for nroff or troff
colcrt(1): filter nroff output for CRT previewing
deroff(1): remove nroff, troff, tbl and eqn constructs
nroff, troff(1): text formatting and
null(4): data sink
factor, primes(1): factor a number, generate large primes
atof, atoi, atol(3): convert ASCII to numbers
introduction to system calls and error numbers
size(1): size of an object file
lorder(1): find ordering relation for an object library
strings(1): find the printable strings in a object, or other binary, file
od (1): octal dump
od (1): octal dump
eqn (PDP only), neqn, checkeq (PDP only) (1): typeset mathematics
eqn (PDP only), neqn, checkeq (PDP only) (1): typeset
open(2): open for reading or writing
fopend, closed, fclosd(3): generic device open/close routines
opend, fopend, closed, fclosd(3): generic
plot: openpl et al.(3): graphics interface
tgetstr, tgoto, tputs(3): terminal independent operation routines
strncpy, strlen, index, rindex(3): string operations
join(1): relational database operator
renice(1): oppress running processes
curses(3): screen functions with "optimal" cursor motion
getopt(1): parse command options
stty(1): set terminal options
lorder(1): find ordering relation for an object library
vi(1): screen oriented (visual) display editor based on ex

cu_v7(1): call UNIX (original V7 version)
a.out(5): assembler and link editor output
ecvt, fcvt, gcvt(3): output conversion
printf, fprintf, sprintf(3): formatted output conversion
fold(1): fold long lines for finite width output device
colcrt(1): filter nroff output for CRT previewing
chog(1): change owner and group
chown(2): change owner and group of a file
chown, chgrp(1): change owner or group
quot(1): summarize file system ownership
stdio(3): standard buffered input/output package
pkopen, pkclose, pkread, pkwrite, pkfail(3): packet driver simulator
more, page(1): file perusal filter for crt viewing
tk(1): paginator for the Tektronix 4014
getopt(1): parse command options
passwd(1): change login password
passwd(5): password file
getpass(3): read a password
passwd(1): change login password
passwd(5): password file
paste(1): merge same lines of several files or
grep, egrep, fgrep(1): search a file for a pattern
awk(1): pattern scanning and processing language
pause(2): stop until signal
cc, pcc(1): C compiler
popen, pclose(3): initiate I/O to/from a process
eqn (PDP only), neqn, checkeq (PDP only) (1): typeset mathematics
eqn (PDP only), neqn, checkeq (PDP only) (1):
mesg (1): permit or deny messages
ptx(1): permuted index
perror, sys_errlist, sys_nerr(3): system error
more, page(1): file perusal filter for crt viewing
tc(1): phototypesetter simulator
phys(2): allow a process to access physical
phys(2): allow a process to access physical addresses
split(1): split a file into pieces
tee(1): pipe fitting
pipe(2): create an interprocess channel
pkopen, pkclose, pkread, pkwrite, pkfail(3): packet
pkopen, pkclose, pkread, pkwrite, pkfail(3): packet driver simulator
pkopen, pkclose, pkread, pkwrite, pkfail(3):
pkopen, pkclose, pkread, pkwrite, pkfail(3): packet driver
pkopen, pkclose, pkread, pkwrite, pkfail(3): packet driver simulator
plot: openpl et al.(3): graphics interface
plot(1): graphics filters
lseek, tell(2): move read/write pointer
pop2(1): interactive pop2 language
pop2(1): interactive pop2 language
popen, pclose(3): initiate I/O to/from a
banner(6): make long posters
postnews(1): submit news articles
exp, log, log10, pow, sqrt(3): exponential, logarithm, power,
log10, pow, sqrt(3): exponential, logarithm, power, square root
pr (1): print file
kill(1): terminate a process with extreme prejudice
prep(1): prepare text for statistical
monitor(3): prepare execution profile

prep(1): prepare text for statistical processing
yacc_prep(1): pre-process some attributes for YACC
colcrt(1): filter nroff output for CRT previewing
unget(1): undo a previous get of an SCCS file
lock(2): lock a process in primary memory
primes(1): factor a number, generate large primes
factor, primes(1): factor a number, generate large
cat(1): catenate and print
prs(1): print an SCCS file
date(1): print and set the date
sact(1): print current SCCS file editing
dint, dcan(1): print diroff-output on the Canon Laserprinter
whoami(1): print effective current user id
pr (1): print file
qume(1): print file on daisywheel printer
banner(6): print large banner on printer
memstat(1): print memory usage map
nm (1): print name list
printenv(1): print out the environment
man(1): find manual information by keywords; print out the manual
man(1): print sections of this manual
id(1): print user and group IDs and names
diction(1): print wordy sentences
strings(1): find the printable strings in a object, or other binary,
printenv(1): print out the environment
banner(6): print large banner on printer
qhul (1): bold filter for the qume printer
qume(1): print file on daisywheel printer
lpr(1): line printer spooler
printf, fprintf, sprintf(3): formatted output
nice, nohup(1): run a command at low priority
nice(2): set program priority
renice(2): set program priority
exit(2): terminate process
fork (2): spawn new process
kill (2): send signal to a process
popen, pclose(3): initiate I/O to/from a process
wait(1): await completion of process
killpg(2): send signal to a process or a process group
setpgrp, getpgrp(2): set/get process group
getpid (2): get process identification
getpid, getppid (2): get process identification
lock(2): lock a process in primary memory
killpg(2): send signal to a process or a process group
ps(1): process status
times(2): get process times
phys(2): allow a process to access physical addresses
nostk(2): allow process to manage its own stack
wait(2): wait for process to terminate
wait2(2): wait for process to terminate
ptrace (2): process trace
kill(1): terminate a process with extreme prejudice
renice(1): oppress running processes
prep(1): prepare text for statistical processing
awk(1): pattern scanning and processing language
m4(1): macro processor
prof(1): display profile data

profil(2): execution time profile
monitor(3): prepare execution profile
profil(2): execution time profile
prof(1): display profile data
lex(1): generator of lexical analysis programs
struct(1): structure Fortran programs
xstr(1): extract strings from C programs to implement shared strings
prolog(1): interactive prolog language
prolog(1): interactive prolog language
true, false(1): provide truth values
prs(1): print an SCCS file
ps(1): process status
ptrace (2): process trace
ptx(1): permuted index
ungetc(3): push character back into input stream
putc, putchar, fputc, putw(3): put character or
putc, putchar, fputc, putw(3): put character or word
puts, fputs(3): put a string on a stream
putc, putchar, fputc, putw(3): put character or word on a stream
pwd(1): working directory name
qhul (1): bold filter for the qume printer
qsort(3): quicker sort
lpq(1): show lineprinter queue
lprm(1): delete jobs from the lineprinter queue
uuq(1): display queue for uucp and uux
qsort(3): quicker sort
qhul (1): bold filter for the qume printer
qume(1): print file on daisywheel printer
quot(1): summarize file system ownership
rand, srand(3): random number generator
ranlib(1): convert archives to random libraries
rand, srand(3): random number generator
ranlib(1): convert archives to random libraries
ratfor(1): rational Fortran dialect
ratfor(1): rational Fortran dialect
getpass(3): read a password
read(2): read from file
mail, rmail(1): send mail to users or read mail
readnews(1): read news articles
read(2): read from file
open(2): open for reading or writing
readnews(1): read news articles
lseek, tell(2): move read/write pointer
malloc, free, realloc, calloc(3): main memory allocator
mail(1): send and receive mail
refer, lookbib(1): find and insert literature
refer, lookbib(1): find and insert literature references in documents
regex, regcmp(3): regular expression compile/execute
regex, regcmp(3): regular expression
regex, regcmp(3): regular expression compile/execute
comm(1): select or reject lines common to two sorted files
lorder(1): find ordering relation for an object library
ingres(1): relational data base management system
join(1): relational database operator
strip (1): remove symbols and relocation bits
calendar(1): reminder service
link(1): link to a remote computer via a terminal line/modem

grep, egrep, fgrep(1): search a file for a pattern
man(1): print sections of this manual
sed(1): stream editor
comm(1): select or reject lines common to two sorted
cut(1): cut out selected fields of each line of a file
mail(1): send and receive mail
mail, rmail(1): send mail to users or read mail
kill (2): send signal to a process
killpg(2): send signal to a process or a process group
diction(1): print wordy sentences
script(1): make typescript of terminal session
setbuf(3): assign buffering to a stream
setpgrp, getpgrp(2): set/get process group
setuid, setgid(2): set user and group ID
getgrent, getgrgid, getgrnam, setgrent, endgrent(3): get group file entry
setjmp, longjmp(3): non-local goto
setpgrp, getpgrp(2): set/get process group
getpwent, getpwuid, getpwnam, setpwent, endpwent
setuid, setgid(2): set user and group ID
sh(1): command language
chsh(1): change default login shell
system(3): issue a shell command
csh(1): a shell (command interpreter) with C-like syntax
sigset, signal, sighold, sigignore, sigrelse, sigpause(3):
sigset, signal, sighold, sigignore, sigrelse, sigpause(3): manage
login (1): sign on
pause(2): stop until signal
alarm(2): schedule signal after specified time
sigset, signal, sighold, sigignore, sigrelse,
kill (2): send signal to a process
killpg(2): send signal to a process or a process group
signal(2): catch or ignore signals
signal(2): catch or ignore signals
sigignore, sigrelse, sigpause(3): manage signals
sigsys(2): catch or ignore signals
sigset, signal, sighold, sigignore, sigrelse, sigpause(3): manage signals
sigset, signal, sighold, sigignore, sigrelse, sigpause(3): manage signals
sigset, signal, sighold, sigignore, sigrelse, sigsys(2): catch or ignore signals
report or change the status of floating point simulation
pkread, pkwrite, pkfail(3): packet driver simulator
tc(1): phototypesetter simulator
sin, cos, tan, asin, acos, atan, atan2(3):
sinh, cosh, tanh(3): hyperbolic functions
null(4): data sink
size(1): size of an object file
size(1): size of an object file
tapeskip(1): skip file on /dev/rht0
sleep(1): suspend execution for an interval
sleep(3): suspend execution for interval
spline(1): interpolate smooth curve
qsort(3): quicker sort
tsort(1): topological sort
sort(1): sort or merge files
sort(1): sort or merge files
comm(1): select or reject lines common to two sorted files
look(1): find lines in a sorted list

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index,
strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex(3): string/
struct(1): structure Fortran programs
struct(1): structure Fortran programs
ioctl, stty, gtty(2): control device
stty(1): set terminal options
style (1): analyze surface characteristics of
su(1): substitute user id temporarily
postnews(1): submit news articles
store, delete, firstkey, nextkey(3): data base subroutines
paste(1): merge same lines of several files or subsequent lines of one file
su(1): substitute user id temporarily
copy(1): copy a subtree of a directory
cptree(1): copy a subtree of a filesystem
sum(1): sum and count blocks in a file
sum(1): sum and count blocks in a file
du (1): summarize disk usage
quot(1): summarize file system ownership
intro(3): summary of job control facilities
newtty(4): summary of the 'new' tty driver
sync(1): update the super block
sync(2): update super-block
style (1): analyze surface characteristics of a document
sleep(1): suspend execution for an interval
sleep(3): suspend execution for interval
swab(3): swap bytes
swab(3): swap bytes
strip (1): remove symbols and relocation bits
sync(1): update the super block
sync(2): update super-block
a shell (command interpreter) with C-like syntax
perror, sys_errlist, sys_nerr(3): system error messages
perror, sys_errlist, sys_nerr(3): system error messages
system(3): issue a shell command
tbl(1): format tables for nroff or troff
tabs(1): set terminal tabs
expand(1): expand tabs to spaces
tabs(1): set terminal tabs
ctags(1): create a tags file
tail(1): deliver the last part of a file
sin, cos, tan, asin, acos, atan, atan2(3): trigonometric
sinh, cosh, tanh(3): hyperbolic functions
tp(1): manipulate tape archive
tar(5): tape archive format
tar(1): tape archiver
tapeskip(1): skip file on /dev/rht0
tar(1): tape archiver
tar(5): tape archive format
deroff(1): remove nroff, troff, tbl and eqn constructs
tbl(1): format tables for nroff or troff
tc(1): phototypesetter simulator
tee(1): pipe fitting
tk(1): paginator for the Tektronix 4014
last(1): indicate last logins of users and teletypes
lseek, tell(2): move read/write pointer
su(1): substitute user id temporarily
termcap(5): terminal capability data base

zaptty(1): run a program without a controlling terminal.
termcap(5): terminal capability data base
/tgetnum, tgetflag, tgetstr, tgoto, tputs(3): terminal independent operation routines
tty(4): general terminal interface
link(1): link to a remote computer via a terminal line/modem
tset, reset(1): set terminal modes
tty(1): get terminal name
stty(1): set terminal options
kermit(1): file transfer, virtual terminal over tty link
script(1): make typescript of terminal session
tabs(1): set terminal tabs
wait(2): wait for process to terminate
wait2(2): wait for process to terminate
kill(1): terminate a process with extreme prejudice
exit(2): terminate process
test(1): condition command
ed(1): text editor
ex, edit(1): text editor
prep(1): prepare text for statistical processing
nroff, troff(1): text formatting and typesetting
tgetent, tgetnum, tgetflag, tgetstr, tgoto,
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs(3): terminal
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs(3):
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs(3): terminal independent
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs(3): terminal independent operation
time, ftime(2): get date and time
time(1): time a command
times(2): get process times
ctime, localtime, gmtime, asctime, timezone(3): convert date and time to ASCII
tip, cu(1): connect to a remote system
tk(1): paginator for the Tektronix 4014
popen, pclose(3): initiate I/O to/from a process
tsort(1): topological sort
touch(1): update date last modified of a file
tp(1): manipulate tape archive
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs(3): terminal independent operation/
tr(1): translate characters
trac(1): a TRAC-processor
ptrace (2): process trace
trac(1): a TRAC-processor
kermit(1): kermit file transfer
kermit(1): file transfer, virtual terminal over tty link
tr(1): translate characters
tree(1): generate a nice listing of a UNIX file tree
tree(1): generate a nice listing of a UNIX file
sin, cos, tan, asin, acos, atan, atan2(3): trigonometric functions
tbl(1): format tables for nroff or troff
deroff(1): remove nroff, troff, tbl and eqn constructs
nroff, troff(1): text formatting and typesetting
true, false(1): provide truth values
true, false(1): provide truth values
tset, reset(1): set terminal modes
tsort(1): topological sort
zaptty(2): zap the controlling tty
w''u"newtty(4):summaryofthe'new'ttydriver
kermit(1): file transfer, virtual terminal over tty link
tty(1): get terminal name

Chapter 1

Commands and application programs

NAME

intro – introduction to commands

DESCRIPTION

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

(1) Commands of general utility.

(1C) Commands for communication with other systems.

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

(1M) Commands used primarily for system maintenance.

(1TU) Command added to the original system at the Delft University of Technology.

SEE ALSO

The **Introduction** of this manual.

DIAGNOSTICS

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

NAME

a68 ALGOL68S compiler

SYNOPSIS

a68 [-wls#] file
ago

DESCRIPTION

A68 compiles an ALGOL68S program. There are two phases to running an ALGOL68S program: first, the compiler *a68* is invoked to produce an object program *a68.out*, and then the run-time support module *ago* is invoked to run the generated program.

Parameters.
a68 knows the following flags:

w	requests warnings to be suppressed.
l	requests a source listing on standard output.
s#	specifies the stropping with # one of:
r	reserved word: int i;
u	upper: INT i;
l	lower: int I;

If -s is not specified, stropping is point-style (.int i;).

Implementation characteristics.
The '!' and the '|' can be used interchangeably for the stick symbol. The '?', '&', '{', '}', and '`' are illegal except in strings and comments, '~' is ignored everywhere.

The environment enquiries (which are not in the standard prelude!) have the following values:

```
int lengths = 1,        int shorths =  0,
real lengths = 1,       real shorths = 0,
bits lengths = 0,       bits shorths = 0,
bytes lengths =         0,      bytes shorths =         0,
bits width = 15,        bytes width =  ?
```

The following non-bold operator-indications are allowed:

```
+       +*      +:=     +=:     -       -:=
*       **      *:=     /       /=      /:=
<       <=      >       >=      =       ^
%       %*      %:=     %*:=
```

Extensions.
The implemented language is ALGOL68S extended with parallel processing (simulated) and heap-generators and a garbage collector. There are no heap-variable-declarations, so

```
.heap [1:2] .real x := (1.2, 2.3);
```

must be written

```
.ref [] .real x = .heap [1:2] .real := (1.2, 2.3);
```

FILES

/usr/bin/a68	compiler load module
/usr/bin/ago	ALGOL68S runtime load module
/etc/alg68	compiler
/etc/a68rte	runtime system
/usr/src/thd/a68s	sources, examples, etc

AUTHOR

Howard J. Ferch, Dept. of Comp. Science, University of Manitoba, sept. 1977.

SEE ALSO

Hibbard, P.G., A Sublanguage of ALGOL68, SIGPLAN Notices 12, 5 (May 1977), which sums up the differences between ALGOL68S and full ALGOL68.

BUGS

Too long programs (200-400 lines) sometimes cause a compiler crash.
The operators .i, .re, and .im give strange error messages. Use *+, re.of, and im.of instead.
Reading complex numbers does not work.
Static nesting of more than 15-30 levels causes strange error messages.
The procedure 'bytespack' does not work. In fact, there seems to be no way at all to do something with .bytes .
Although the compiler attempts to read from standard input if the file parameter is missing, it sometimes fails.

NAME

ac – login accounting

SYNOPSIS

ac [**–w** wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. **–w** is used to specify an alternate *wtmp* file. **–p** prints individual totals; without this option, only totals are printed. **–d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), login(1), utmp(5).

NAME

ack – Amsterdam Compiler Kit

SYNOPSIS

ack arguments
acc arguments
apc arguments
abc arguments
machine arguments

DESCRIPTION

Ack transforms sources in several languages to load files for a variety of machines, internally using several phases. The transformation can be stopped at any phase. Combining sources from several languages is allowed. The run-time system of the first language mentioned, either in the program call name or in the arguments, is automatically included. The libraries of all other languages mentioned, containing most of the run-time systems, are also automatically included. Two types of load files can be distinguished, *a.out* files containing machine code and *e.out* files containing virtual EM machine code. The last type is designed for interpretation. Compilation time for interpretation is fast and gives many run-time checks, but execution is about seven times slower. Which combinations of languages and machines are allowed varies in time and depends on the installation.

The actions of *ack* are to repeatedly transform files with a particular suffix into files with another suffix, finally combining the results into a single file.

Different machines can use different suffices, but the following are recognized by most machines:

.p Pascal program.

.c C module.

.b Basic program.

.e EM assembly module in human readable form.

.k Compact EM assembly code.

.m Optimized compact EM assembly code.

.s Machine assembly language code.

.o Object file.

Ack accepts the following flags:

machine used

–m *machine*
This flag tells *ack* to generate a load file for *machine*. *Machine* can also be used as the program call name, instead of *ack*. e.g. *ack – mi86 file.p* is equivalent to *i86 file.p*.

output files

–o Use the next argument as the name of the resulting file. *Ack* produces *a.out* or *e.out* by default. This flag can always be used when *ack* produces a single output file, as in

ack -c.s main.c -o new.s.

The output is produced on *new.s* instead of *main.s*.

−c.*suffix*

−c *Ack* tries to transform each source into a file with the *suffix*. When no *suffix* is specified *ack* stops just before the phase where it combines all arguments into a load file, thereby transforming the sources into *.k*, *.s*, *.o* or *.m* files. One extra *suffix* is recognized here, *.i*, this tells *ack* to only preprocess all human readable sources, producing files with *suffix .i*.
Note: *ack* refuses to overwrite argument *.e* files.

−t Preserve all intermediate files. If two −t are used, *ack* also preserves core dumps and output of failed transformations.

messages

−w Suppress all warning messages.

−E Produce a complete listing of each Pascal source program. Normally for each error, one message, including the source line number, is given.

−e List only the erroneous lines of each Pascal source program.

−v Verbose. Print information while juggling with files.

preprocessing

−I*dir* `#include´ files whose names do not begin with `/´ are always sought first in the directory of the *file* argument, then in the directories named in −I options, then in directories on a standard list.

−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*, before preprocessing.

debugging

−p This flag tells both the Pascal and C front ends to include code enabling the user to do some monitoring/debugging. Each time a routine is entered the routine **procentry** is called and just before each return **procexit** is called. These routines are supplied with one parameter, a pointer to a string containing the name of the routine.

optimizing

−O Use as many optimizers as possible. *Ack* can almost always use the EM peephole optimizer. Sometimes the global optimizer or machine-dependent optimizers can be used.

−L Disable the generation of code by the front ends to record line number and source file name at run-time.

libraries

−l*name*
Tells *ack* to insert a library module at this point. For example: the library *mon* contains the routines for systems calls needed by both C and Pascal.

−.*suffix*
When linking multiple *.o* or *.m* files created by separate calls of *ack* together, *ack* cannot deduce the run-time system needed, unless called as *apc* or *acc*. This flag serves to tell *ack* which runtime system is needed in such a case. For example: "ack −c x.c ; ack −.c x.o".

−r.*suffix*
Most frontends and backends use one or more run-time libraries. These flags tell *ack* to include the libraries needed when a file with *suffix* would be included in the arguments.

−LIB This flag tells the peephole optimizer to add information about the visibility of the names used to each output module. This is needed by most assembler/linkers when these modules are to be inserted in libraries.

interpreter

−{xxx}
The string starting after `{' and terminated by a `}' is passed as an option string to the Pascal compiler and supersedes corresponding options given in the source file. See the ACK reference manual [4] for a list of options.

−+xxx, −−xxx
When you want to interpret your program, you may select some options during interpretation, like test, profile, flow, extra and count. A short description of these flags follows:

t(est)	test for undefined, overflow, array bound etc.
f(low)	keep track of executed source lines.
c(ount)	count the number of times a source line is executed.
p(rofile)	count the memory cycles executed per source line.

Test is on by default, the others are off. Normally, you give these flag options each time you run the interpreter. The EM assembler/linker gives you the opportunity to change the defaults per program. The changed options are recorded in the "e.out" header. These flags −− and −+ are passed to the assembler for this purpose. So, −−t and −+pfce invert the defaults.

general

−R*program*= *xxx*
Replace the *program* by the pathname *xxx*. The program names referred to later in this manual are allowed here.

−R*program*− *xxx*
The flag argument −*xxx* is given to *program*.

−R*program*:*n*

Set the priority of the indicated transformation to *n*. The default priority is 0, setting it to -1 makes it highly inlikely the the phase will be used, setting it to 1 makes it very likely that the phase will be used.

−k Do not stop when an error occurs, but try to transform all other arguments as far as possible.

−g Try to run the resulting load file. No arguments can be passed this way, so it is only useful in simple cases.

All arguments without a suffix or with an unrecognized suffix are passed to the loaders, as for flags.

PREPROCESSOR

All C source programs are run through the preprocessor before they are fed to the compiler proper. Other human readable sources (Pascal programs and machine assembly) are only preprocessed when they start with a `#´.

Ack adds a few macro definitions when it calls the preprocessor. These macro´s contain the word- and pointer-size and the sizes of some basic types used by the Pascal and/or C compiler. All sizes are in bytes.

EM_WSIZE	wordsize	EM_PSIZE	pointer size
EM_SSIZE	size of shorts (C)	EM_LSIZE	size of longs (C+Pascal)
EM_FSIZE	size of floats (C)	EM_DSIZE	size of doubles (C+Pascal)

The name of the *machine* or something like it when the machine name is numeric is also defined (as 1).

The default directories searched for include files differ for each machine. Some machines do not even use */usr/include.*

PROGRAMS

Ack uses one or more programs in each phase of the transformation. The table below gives the names *ack* uses for these programs. Internally *ack* maintains a mapping of these names to pathnames for load files. The table specifies which type of files are accepted by each program as input and the file type produced as output.

input	name	output	description
.c	cem	.k	C front end [4,5,6]
.p	pc	.k	Pascal front end [2,3,6]
.b	abc	.k	Basic front end [6,8]
.e	encode	.k	Compactify EM assembly language [1]
.k	opt	.m	EM peephole optimizer
.k .m	decode	.e	Produce human readable EM assembly
.k .m	emass	e.out	Linker producing EM machine code [1]
.m	be	.s	backend
.s	asld	a.out	Assembler/linker producing machine code
.s	as	.o	Assembler
.o	ld	a.out	Linker producing machine code

SEE ALSO

All manual pages can be found in /proj/ack/man on the PDP-11/73 and in /das/man on the Geminix.

Most of the documents listed below can be found in /proj/ack/doc on the PDP-11/73.

[1] A.S. Tanenbaum, Hans van Staveren, Ed Keizer and Johan Stevenson *Description of a machine architecture for use with block structured languages* Informatica report

IR-81.
[2] K. Jensen and N. Wirth *PASCAL, User manual and report* Springer Verlag.
[3] The ISO Pascal standard proposal ISO/TC97/SC5-N462.
[4] B.W. Kernighan and D.M. Ritchie, *The C Programming language*, Prentice-Hall, 1978
[5] D.M. Ritchie, *C Reference Manual*
[6] Amsterdam Compiler Kit, reference manuals and UNIX manual pages.
[7] E.G. Keizer, Ack description file reference manual.
[8] M.L. Kersten, *The ABC compiler.*

DIAGNOSTICS

The diagnostics are intended to be self−explanatory.

BUGS

The **.-g** flag is inoperative.
Not all warning messages are superseded by **−w**.
Argument assembly files are not preprocessed when fed into the universal assembler.
Ack on the Geminix cannot compile programs written in Pascal.

AUTHOR

Ed Keizer, Vrije Universiteit, Amsterdam

NAME

adb – debugger

SYNOPSIS

adb [– **w**] [objfil [corfil]]

DESCRIPTION

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

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

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

In general requests to *adb* are of the form

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

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

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

EXPRESSIONS

. The value of *dot.*

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

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

integer.fraction
A 32 bit floating point number.

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

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

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

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

routine.name

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

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

Monadic operators

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

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

– *exp* Integer negation.

~ *exp* Bitwise complement.

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

e1+ *e2* Integer addition.

e1– *e2* Integer subtraction.

*e1** *e2* Integer multiplication.

e1% *e2* Integer division.

e1& *e2* Bitwise conjunction.

e1| *e2* Bitwise disjunction.

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

COMMANDS

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

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

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

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

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

o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O 4 Print 4 bytes in octal.
q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.
x 2 Print 2 bytes in hexadecimal.
X 4 Print 4 bytes in hexadecimal.
u 2 Print as an unsigned decimal number.
U 4 Print long unsigned decimal.
f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.

c 1 Print the addressed character.

C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.

s *n* Print the addressed characters until a zero character is reached.

S *n* Print a string using the @ escape convention. *n* is the length of the string including its zero terminator.

Y 4 Print 4 bytes in date format (see *ctime*(3)).

i n Print as PDP11 instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.

a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

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

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.

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

r 0 Print a space.

n 0 Print a newline.

"..." 0 Print the enclosed string.

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

\+ *Dot* is incremented by 1. Nothing is printed.

\- *Dot* is decremented by 1. Nothing is printed.

newline

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

[?/]**l** *value mask*

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

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

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

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

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

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

&modifier

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

"..." " 0"
Print the enclosed string.

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

+ *Dot* is incremented by 1. Nothing is printed.

− *Dot* is decremented by 1. Nothing is printed.

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

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

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

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

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

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

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

$*modifier*
Miscellaneous commands. The available *modifiers* are:

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

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

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

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

b Print all breakpoints and their associated counts and commands.

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

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

k The current subprocess, if any, is terminated.

VARIABLES

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

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

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

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

ADDRESSES

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

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

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

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

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

RESTRICTIONS

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

NAME

admin − create and administer SCCS files

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

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

−f*flag* This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *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 Causes the "No id keywords (ge6)" message issued by *get*(1) or *delta*(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get*(1)) are found in the text retrieved or stored in the SCCS file.

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

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 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.

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

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

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

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

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

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

FILES

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

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

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

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

apl-apl interpreter

SYNPOSIS

apl

DESCRIPTION

This is the really improved APL interpreter. It puts the terminal into a funny mode, so watch out. If it blows up, push the alt. char set and/or char set lock key(s) to switch back. You might also need a "stty erase ?" call to switch the erase kill character back to normal.

All of the operators are exactly as in apl\360. Overstrikes are often required, and they work (use space, not cursor right). The keyboard is the LA36-keyboard with apl-font.

Function definition is not what you would expect. Functions are loaded from files. The first line of the file is the function header, as you would expect it but with no del. The rest of the file is the lines of the function. Lines are numbered, but there's none of the square bracket jazz. If you say)READ FILE it will load the function in that file. If you say)EDIT FILE it will put you in the unix editor to change that file. Upon exit, it will read the file in as though by)READ.

All of the usual operators are available, including domino. Also available are monadic encode and epsilon.

FILES

apl_ws temporary workspace file.

continue - continue workspace

SEE ALSO

apl.2 synposis of system commands

BUGS

I don't know of too many, but mail any trouble reports to jrl.

Interpreter occasionally blows up, giving core images.

Character comparisons don't work (but try concatenating a character vector to a numeric vector and vice versa.)

Many mixed functions take less general arguments than you might expect. Then again any integer origin is allowed.

NAME

ar – archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

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

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

d Delete the named files from the archive file.

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

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

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

p Print the named files in the archive.

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

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

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

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

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

FILES

/tmp/v*temporaries

SEE ALSO

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

RESTRICTIONS

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

NAME

arcv – convert archives to new format

SYNOPSIS

arcv file ...

DESCRIPTION

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

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

FILES

/tmp/v*, temporary copy

SEE ALSO

ar(1), ar(5)

NAME

as − assembler

SYNOPSIS

as [−] [−**V**] [−**o** objfile] file ...

DESCRIPTION

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

)	Parentheses error
]	Parentheses error
<	String not terminated properly
*	Indirection used illegally
a	Error in address
b	Branch instruction is odd or too remote
e	Error in expression
f	Error in local ('f' or 'b') type symbol
g	Garbage (unknown) character
i	End of file inside an if
m	Multiply defined symbol as label
o	Word quantity assembled at odd address
p	'.' different in pass 1 and 2
r	Relocation error
u	Undefined symbol
x	Syntax error

RESTRICTIONS

Syntax errors can cause incorrect line numbers in following diagnostics.

NAME

at – execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

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

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

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

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

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

Standard output or error output is lost unless redirected.

FILES

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

SEE ALSO

calendar(1), cron(8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

RESTRICTIONS

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

NAME

awk − pattern scanning and processing language

SYNOPSIS

awk [−**F***c*] [prog] [file] ...

DESCRIPTION

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

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

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

A pattern-action statement has the form

pattern { action }

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

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

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

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

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

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

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

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

A relational expression is one of the following:

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

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

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

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

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

or by using the **−F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

SEE ALSO

lex(1), sed(1)

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

RESTRICTIONS

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

NAME

bas – basic

SYNOPSIS

bas [file]

DESCRIPTION

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

statement
integer statement

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

Statements have the following syntax:

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

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

done
: Return to system level.

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

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

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

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

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

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

listed. If two arguments are given, all internal statements inclusively between the arguments are printed.

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

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

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

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

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

Expressions have the following syntax:

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

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

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

_ expression
The result is the negation of the expression.

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

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

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

The following is the list of operators:

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

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

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

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

+ − Add and subtract.

* / Multiply and divide.

^ Exponentiation.

The following is a list of builtin functions:

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

exp(x) is the exponential function of *x*.

log(x) is the natural logarithm of *x*.

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

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

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

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

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

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

abs(x) is the absolute value of *x*.

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

FILES

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

DIAGNOSTICS

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

RESTRICTIONS

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

NAME

basename – strip filename affixes

SYNOPSIS

basename string [suffix]

DESCRIPTION

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

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

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

SEE ALSO

sh(1)

NAME

bc − arbitrary-precision arithmetic language

SYNOPSIS

bc [−**c**] [−**l**] [file ...]

DESCRIPTION

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

Comments
    are enclosed in /* and */.

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

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

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

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

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

Functions in −**l** math library
    s(x) sine
    c(x) cosine
    e(x) exponential
    l(x) log
    a(x) arctangent
    j(n,x) Bessel function

All function arguments are passed by value.

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

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

For example

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

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

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

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

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

FILES

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

SEE ALSO

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

RESTRICTIONS

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

NAME

bdiff – big diff

SYNOPSIS

bdiff file1 file2 [n] [–s]

DESCRIPTION

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

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

FILES

/tmp/bd?????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cal – print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

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

Try September 1752.

RESTRICTIONS

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

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

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

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

FILES

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

SEE ALSO

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

RESTRICTIONS

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

NAME

cat − catenate and print

SYNOPSIS

cat [**−s**] [**−u**] file ...

DESCRIPTION

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

```
cat file
```

prints the file and

```
cat file1 file2 >file3
```

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

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

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

SEE ALSO

pr(1), cp(1)

BUGS

cat a b > a and **cat a b > b** cause strange results (because of *sh(1)*).

NAME

cat − catenate and print

SYNOPSIS

cat [**−u**] file ...

DESCRIPTION

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

```
cat file
```

prints the file and

```
cat file1 file2 >file3
```

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

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

SEE ALSO

pr(1), cp(1)

RESTRICTIONS

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

NAME

cb – C program beautifier

SYNOPSIS

cb

DESCRIPTION

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

NAME

cc, pcc – C compiler

SYNOPSIS

cc [option] ... file ...

pcc [option] ... file ...

DESCRIPTION

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

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

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

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

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

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

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

–O Invoke an object-code optimizer.

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

–P Run only the macro preprocessor and place the result for each '.c' file in a corresponding '.i' file and has no '#' lines in it.

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

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

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

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

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

–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 directories on a standard list.

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

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

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

RESTRICTIONS

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

NAME

cd – change working directory

SYNOPSIS

cd directory

DESCRIPTION

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

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

SEE ALSO

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

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

cdc –r*SID* [–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 *EXAMPLES*). 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 *Source Code Control System User's Guide.* Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc −r1.6 −m"bl78-12345 !bl77-54321 bl79-00001" −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(5).
Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cec, sec, cec68, sec68, cec09, sec09 – Concurrent Euclid compiler

SYNOPSIS

cec [**–help**] [**–m***MMK*] [**–S**] [**–c**] [**–O**] [**–l***LIB*]
[**–o** *file*] [**–k**] [**–r**] [**–R**] [**–i**] [**–n**]
[**–a** *addr*] [**–M**] [**–T**] [**–q***N*] [**–t***N*] [**–x***N arg*]
[**–L** *dir*] file ...

DESCRIPTION

Cec is the Concurrent Euclid compiler, which implements the Concurrent Euclid language as defined in "Specification of Concurrent Euclid", technical report CSRG-133. Concurrent Euclid (or CE) consists of a core subset of the Euclid language called Sequential Euclid (SE) plus concurrency features which implement processes, monitors and condition variables.

Up to 20 input files are accepted, each of which must be a Euclid source file ('.e'), an assembly source file ('.s', '.t', '.u'), an object file ('.o') or an object library ('.a'). The first file is assumed to be the main program. The output load module is put in 'file.out' (PDP-11), 'file.sout' (MC68000) or 'file.tout' (M6809). If the -S option is specified, the output assembly sources are saved in 'file.s' (PDP-11), 'file.t' (MC68000) or 'file.u' (M6809). If the -c option is specified, the output object modules are saved in 'file.o'. Error messages are sent to the standard output. Disasters (such as not finding a pass of the compiler) are logged on the diagnostic output.

'Sec' is the version of cec for sequential programs, equivalent to 'cec -m11q'. 'Cec68' and 'cec09' are equivalent to 'cec -m68b' and 'cec -m09b' respectively. 'Sec68' and 'sec09' are the equivalents for sequential programs.

The following options are recognized by *cec*.

-h Help! A description of cec is printed on the terminal.

-mMMK Generate code for target machine MM and link with kernel K. MM may be 11 (PDP-11), 68 (MC68000) or 09 (M6809). K may be 's' (simulation kernel under Unix), 'b' (standalone kernel) or 'q' (sequential program, no kernel). The default is -m11s. (Note: Only -m11s, -m11b, -m68b, -m09b and -mMMq are available so far.)

-S Produce assembly code only (do not assemble or link). Saves the output assembly sources in 'file.s' (PDP-11), 'file.t' (MC68000), or 'file.u' (M6809).

-c Compile and assemble only (do not link). Saves the output object modules in 'file.o'.

-O Do not emit run-time line numbering and checking code. (Equivalent to -k -r.)

-lLIB Link with the specified standard library ('/usr/lib/libLIB.a').

-o file Put the output load module in the specified file.

-k Do not emit range and assertion checking code. The compiler normally emits code to check all subscript and case tag ranges and user assertions at run time. This is very useful in debugging (see cedb(I)).

-r Do not emit line numbering code. The compiler normally emits code to keep track of the source line number being executed in a register. This is very useful in debugging (see cedb(I)).

-i Link using split I and D space (PDP-11 only).

-n Link using shared text (PDP-11 only).

-a addr Link at the specified address (MC68000 and M6809 only).

-M Produce a load map on the standard output (MC68000 and M6809 only).

The following options are recognized for compiler maintenance:

-T Save compiler temporary files in the user's directory.

*-q*N Run only the first N passes of the compiler.

*-t*N Trace S/SL execution of pass N of the compiler.

*-x*N arg Pass 'arg' as an argument to pass N of the compiler.

-L dir Run the compiler passes from the specified directory.

FILES

file.e	source input file
file.s (.t,.u)	output assembly code
file.o	output object code
file.out	output load module
/tmp/ce??a$$	compiler temporaries
/usr/lib/coneuc/*	compiler passes

SEE ALSO

Cedb(1TU) { Only on-line, not in this manual }, as(1), ld(1)
Various manual pages in /usr/src/thd/euclid/man
Documents can be found in /usr/src/thd/euclid/doc
Specification of Concurrent Euclid
(CSRG-133)
A Short Introduction to Concurrent Euclid
The Euclid Report.

DIAGNOSTICS

The diagnostics produced by the Concurrent Euclid compiler are intended to be self-explanatory. Programs which fail at run-time can be debugged using cedb(I).

U OF T INFO

Concurrent Euclid was designed at CSRG by R.C. Holt and J.R. Cordy.

NAME

checknews – check to see if user has news

SYNOPSIS

checknews [**-ynqevvN**] [*newsgroup list*] [*readnews options*]

DESCRIPTION

Checknews reports whether or not the user has news.

–y reports "There is news" if the user has news to read. If the **–N** flag is given, then the newsgroups requested are also printed.

–n reports "No news" if there isn't any news to read.

–q causes *checknews* to be quiet. Instead of printing a message, the exit status indicates news. A status of 0 means no news, 1 means there is news.

–v alters the **–y** message to show the name of the first newsgroup containing unread news. Doubling **v** (e.g. **–vv**) will cause an explanation of *any* claim of new news, and is useful if *checknews* and *readnews*(1) disagree on whether there is news.

–e executes *readnews* if there is news.

–N causes the next argument to be read and interpreted as a comma-separated list of newsgroups to be checked.

If there are no options, **–y** is the default.

FILES

~/.newsrc	Active newsgroups
/usr/lib/news/active	Options and list of previously read articles

SEE ALSO

inews(8), postnews(1), readnews(1), vnews(1), news(5), newsrc(5), expire(8), recnews(8), sendnews(8), uurec(8)

BUGS

The **–N** flag should really be named **–n** to be consistent with other news programs, but **–n** was already used. If the **–v** flag is used with the **–N** flag, the first newsgroup in the list where there is news should be printed instead of the entire list. If the **–N** flag is used and *readnews* is invoked (with **–e**) it does not restrict news reading to those groups checked, but reads all newsgroups where there is new news.

NAME

chfn – change finger entry

SYNOPSIS

chfn [loginname]

DESCRIPTION

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

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

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

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

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

FILES

/etc/passwd, /etc/ptmp

SEE ALSO

finger(1), passwd(5)

RESTRICTIONS

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

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

NAME

chghist change the history entry of an SCCS delta

SYNOPSIS

chghist —rSID name ...

DESCRIPTION

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

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

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

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

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

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

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

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

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

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

Only the owner of a file (or the superuser) may change its mode.

SEE ALSO

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

NAME

chog – change owner and group

SYNOPSIS

chog owner files...

DESCRIPTION

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

Only the superuser is allowed to change owner and group ID's.

FILES

/etc/passwd
/etc/group

SEE ALSO

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

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

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

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

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

FILES

/etc/passwd
/etc/group

SEE ALSO

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

NAME

chroot – change root directory

SYNOPSIS

chroot newdir

DESCRIPTION

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

Only the superuser can change the root directory.

FILES

/bin/sh

SEE ALSO

chroot(2)

NAME

chsh – change default login shell

SYNOPSIS

chsh name [shell]

DESCRIPTION

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

An example use of this command would be

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

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

SEE ALSO

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

NAME

Clisp – lisp interpreter

SYNOPSIS

Clisp

DESCRIPTION

Clisp is a C-written lisp interpreter.

help (subject) gives information on a built in function; help (" ") gives all help available. Extensive documentation is to be found in /usr/man/doc/Clisp.

The way to stop Clisp is ^D.

FILES

/usr/thd/Clisp
/usr/man/doc/Clisp

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] file1 file2

DESCRIPTION

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

Options:

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

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

SEE ALSO

diff(1), comm(1)

DIAGNOSTICS

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

NAME

col − filter reverse line feeds

SYNOPSIS

col [− bfx]

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

RESTRICTIONS

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

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [–] [–**2**] [file ...]

DESCRIPTION

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

The optional – suppresses all underlining. It is especially useful for previewing *allboxed* tables from *tbl*(1).

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

A typical use of *colcrt* would be

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

SEE ALSO

nroff/troff(1), col(1), more(1), ul(1)

AUTHOR

William Joy

BUGS

Should fold underlines onto blanks even with the '–' option so that a true underline character would show; if we did this, however, *colcrt* wouldn't get rid of *cu'd* underlining completely.

Can't back up more than 102 lines.

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

Lines are trimmed to 132 characters.

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

NAME

comb – combine SCCS deltas

SYNOPSIS

comb [–**o**] [–**s**] [–psid] [–clist] files

DESCRIPTION

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

The generated shell procedure is written on the standard output.

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

–**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.

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

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

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] file1 file2

DESCRIPTION

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

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

SEE ALSO

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

NAME

compact, uncompact, ccat – compress and uncompress files, and cat them

SYNOPSIS

compact [name ...]
uncompact [name ...]
ccat [file ...]

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular,

... | compact | uncompact | ...

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C; file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file compressed by *compact*. If no file names are given, the standard input is uncompacted to the standard output.

Ccat cats the original file from a file compressed by *compact*, without uncompressing the file.

RESTRICTION

The last segment of the filename must contain fewer than thirteen characters to allow space for the appended '.C'.

FILES

*.C compacted file created by compact, removed by uncompact

SEE ALSO

Gallager, Robert G., 'Variations on a Theme of Huffman', *I.E.E.E. Transactions on Information Theory*, vol. IT-24, no. 6, November 1978, pp. 668 - 674.

AUTHOR

Colin L. Mc Master

NAME

copy – copy a subtree of a directory

SYNOPSIS

copy <dir1> <dir2>

DESCRIPTION

copies all files (including subdirectories infinitely deep) in <dir1> to <dir2>. The originals are not destroyed.

BUGS

A copy to a subtree of <dir1> is endless.

NAME

cp – copy

SYNOPSIS

cp file1 file2

cp file ... directory

DESCRIPTION

File1 is copied onto *file2*. The mode and owner of *file2* are preserved if it already existed; the mode of the source file is used otherwise.

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

Cp refuses to copy a file onto itself.

SEE ALSO

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

NAME

cpio − copy file archives in and out

SYNOPSIS

cpio −o [**acBv**]

cpio −i [**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.

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

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

The meanings of the available options are:

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

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

d Directories are to be created as needed.

c Write header information in ASCII character form for portability.

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

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

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

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

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

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. PDP-11/73 only.

b Swap both bytes and halfwords. Use only with the **−i** option. PDP-11/73 only.

6 Process an old (i.e., UNIX System Sixth Edition format) file. Only useful with **−i** (copy in).

EXAMPLES

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

```
ls | cpio −o >/dev/ht0

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

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

```
find . −cpio /dev/rmt0
```

SEE ALSO

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

RESTRICTIONS

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

NAME

cptree – copy a subtree of a filesystem

SYNOPSIS

cptree dir1 dir2

DESCRIPTION

cptree copies a complete subtree of a filesystem onto another; dir1 is a directory specifying the source, dir2 a directory specifying the target.
Note that dir2 has to be an existing directory.

This command is especially usefull for copying user directories from one (mountable) file system to another.

FILES

/usr/thd/cptree (PDP)
/usr/bin/thd/cptree (GMX)

NAME

croff, diroff – formatters for the Canon Laserprinter

SYNOPSIS

croff [options] filename
diroff [options] filename

DESCRIPTION

The functionality of *croff* is the same as that of nroff (see *nroff(1)*). Its output however is suitable to be printed on the Canon Laserprinter, which is currently attached to the PDP-11/73. The options and deviations from nroff are as follows:

- *mc* replaces the *mm*-macropackage which can be used with nroff.

- *Croff* generates 67 lines output per page.

- Mathematical symbols can be printed when available.

- The graphic features of nroff can no longer be used in a standard way. The first character of a sequence of graphical characters must be '\200'.

- The are some new fonts available:
  - .U or \fU Underline font
  - .D or \fD Dotted font
  - .W or \fW White on black font

Diroff acts the same as troff but its output is device independent; it consists of normal ascii-characters. At the moment 3 typesetters are available (to be selected with the –T option):
- can Canon typesetter
- elite Elite typesetter
- crr Courier typesetter

Dint (for the typesetters Elite and Courier) and *dcan* (for the canon typesetter) are used to print the output of *diroff* on the laserprinter (see *dint(1TU)*). *Lbp* (see *lbp(1)*) can be used to print *croff* – output.

SEE ALSO

nroff(1), troff(1)
lbp(1), dint(1TU)

NOTES

This manual was printed using *diroff -Tcan*

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 cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are 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 lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

BUGS

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

NAME

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

SYNOPSIS

csh [**–cefinstvVxX**] [arg ...]

DESCRIPTION

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

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

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

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

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

Lexical structure

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

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

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

Commands

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

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

Jobs

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

[1] 1234

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

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the *bg* command, or run some other commands and then eventually bring the job back into the foreground with the foreground command *fg*. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key ^Y which does not generate a STOP signal until a program attempts to *read*(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

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

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

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

Status reporting

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

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

Substitutions

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

History substitutions

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

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

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

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

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

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!-2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a *redo*.

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

0	first (command) word
n	*n*'th argument
^	first argument, i.e. '1'
$	last argument
%	word matched by (immediately preceding) ?*s*? search
x-*y*	range of words
-*y*	abbreviates '0-*y*'

*	abbreviates '^−$', or nothing if only 1 word in event
*x**	abbreviates '*x*−$'
x−	like '*x**' but omitting word '$'

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

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

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

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

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

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

Quotations with ' and "

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

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

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

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

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

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

Variable substitution

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

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

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

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '$' characters. This expansion can be prevented by preceding the '$' with a '\' except within '"'s where it **always** occurs, and within '`'s where it **never** occurs. Strings quoted by '`' are interpreted later (see *Command substitution* below) so '$' substitution does not occur there until later, if at all. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

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

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

prevent later command or filename substitution.

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

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

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

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

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

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

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

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

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

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

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

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

$<
Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

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

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

Filename substitution

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

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

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

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus '~source/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, its is truncated, its previous contents being lost.

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

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

>> name
>>& name
>>! name
>>&! name

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

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

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

Expressions

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

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

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side

is a *pattern* (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

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

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

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

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

Control flow

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

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

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

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

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

alloc
Shows the amount of dynamic core in use, broken down into used and free core, and address of the last location in the heap. With an argument shows each used and free block on the internal dynamic memory chain indicating its address, size, and whether

it is used or free. This is a debugging command and may not work in production versions of the shell; it requires a modified version of the system memory allocator.

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

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

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

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

cd
cd name
chdir
chdir name
Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.
If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

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

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

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

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

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

eval arg ...
(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

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

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

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

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

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

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

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

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

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

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**
...
else if (expr2) **then**
...
else
...
endif

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

jobs
jobs −l

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

kill %job
kill −sig **%**job ...
kill pid
kill −sig pid ...
kill −l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h,* stripped of the prefix 'SIG'). The signal names are listed by 'kill −l'. There is no default, saying just 'kill' does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

limit
limit *resource*
limit *resource maximum-use*

(Vax only.) Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource.* If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

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

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

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

login

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

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

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

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

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

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

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

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

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

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

rehash

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

repeat count command

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

set
set name
set name=word
set name[index]=word
set name=(wordlist)

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

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

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *csh* variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift
shift variable

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

source name

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

stop
stop %job ...

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

suspend

Causes the shell to stop in its tracks, much as if it had been sent a stop signal with ^Z. This is most often used to stop shells started by *su*(1).

switch (string)
case str1:

```
  ...
 breaksw
...
default:
  ...
 breaksw
endsw
```

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command

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

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern

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

unhash

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

unlimit *resource*
unlimit

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

unset pattern

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

unsetenv pattern

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

wait

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

while (expr)
...
end

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

% job

Brings the specified job into the foreground.

% job **&**

Continues the specified job in the background.

@
@ name = expr
@ name[index] = expr

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

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

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

Pre-defined and environment variables

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

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

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

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

cwd The full pathname of the current directory.

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

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

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

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

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

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

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

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

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

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

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

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

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

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

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

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

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

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

Non-builtin command execution

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

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

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

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

Argument list processing

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

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

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

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

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

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

−s Command input is taken from the standard input.

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

−v Causes the *verbose* variable to be set, with the effect that command input is echoed

after history substitution.

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

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

−X Is to **−x** as **−V** is to **−v**.

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

Signal handling

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

AUTHOR

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

FILES

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

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters (5120 characters on most PDP11 systems). The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

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

RESTRICTIONS

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the shell will then immediately execute 'c'. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

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

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

Commands within loops, prompted for by '?', are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '$' substitutions.

NAME

ctags – create a tags file

SYNOPSIS

ctags [**–BFatuwvx**] name ...

DESCRIPTION

Ctags makes a tags file for *ex*(1) from the specified C, Pascal and Fortran sources. A tags file gives the locations of specified objects (in this case functions and typedefs) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, *ex* can quickly find these object name definitions.

If the **–x** flag is given, *ctags* produces a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

If the **–v** flag is given, an index of the form expected by *vgrind*(1) is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through **sort –f**. Sample use:

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

Files whose name ends in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

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

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

–a append to tags file.

–t create tags for typedefs.

–u causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

–w suppressing warning diagnostics.

The tag *main* is treated specially in C programs. The tag formed is created by prepending *M* to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

FILES

tags	output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

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

RESTRICTIONS

Recognition of *functions*, *subroutines* and *procedures* for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

ctags does not know about #ifdefs.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of **−tx** shows only the last line of typedefs.

NAME

cu_v7 − call UNIX (original V7 version)

SYNOPSIS

cu_v7 telno [**−t**] [**−s** speed] [**−a** acu] [**−l** line]

DESCRIPTION

Cu_v7 is the original version of CU from UNIX V7, used with the DN11 and the Bell 801 auto call unit. *Cu_v7* is obsolete and is not supported by the ULTRIX-11 system.

Cu_v7 calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of text files. *Telno* is the telephone number, with minus signs at appropriate places for delays. The **−t** flag is used to dial out to a terminal. *Speed* gives the transmission speed (110, 134, 150, 300, 1200); 300 is the default value.

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

−a /dev/cua0 **−l** /dev/cul0

After making the connection, *cu_v7* runs as two processes: the send process reads the standard input and passes most of it to the remote system; the receive process reads from the remote system and passes most data to the standard output. Lines beginning with '~' have special meanings.

The send process interprets the following:

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

The receive process handles output diversions of the following form:

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

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

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

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

FILES

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

SEE ALSO

dn(4), tty(4)

DIAGNOSTICS

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

RESTRICTIONS

The syntax is unique.

NAME

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*.

EXAMPLES

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

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

SEE ALSO

grep(1), paste(1).

NAME

das – das compiler

SYNOPSIS

das [[options]* [files]] +

DECRIPTION

Das is the UNIX Das compiler.

This manual first describes the overall structure of Das programs and the facilities provided for separate compilation. Storing of output files, processing of options and other *das* related topics will be described later.

Compilation Units - Library Units.

The syntax of the overall structure of Das programs is similar to that of the Ada language. A (partial) BNF discription is given below. The semantical restrictions/similarities with respect to the Ada language will now be discussed.

A Das program is a collection of one or more *compilation units* submitted to the compiler in one or more compilations. *Compilation units* of a program are said to belong to a program library. The designator of a separately compiled subprogram (whether a *library unit* or *subunit)* is the *unit_simple_name* given in the specification of the *library unit*. Within a *program library* the simple name of all *library units* are distinct.

The effect of compiling a *library unit* is to define (or redefine) this unit as one that belongs to the program library.

The effect of compiling a *secondary unit* is to define the body of a *library unit,* or in the case of a *subunit,* to define the proper body of a program unit that is declared within another *compilation unit.*

A subprogram that is a *library unit* can be used as a *main program* in the usual sense. *Main programs* are parameterless procedures, and every *main program* must be a subprogram that is a *library unit.*

Order of Compilation.

The *compilation units* of a compilation must be compiled in a given order. The rules defining the order in which units can be compiled are direct consequences of the visibility rules and, in particular, of the fact that any *library unit* that is mentioned by the *context clause* of a *compilation unit* is visable in the *compilation unit.*

A *compilation unit* must be compiled after all *library units* named by its context clause. A *secondary unit* must be compiled after its corresponding *library unit.*

The order in which the *compilation units* of a program are compiled must be consistent with the partial ordening defined by the above rules.

Similar rules apply for recompilations. A *compilation unit* is potentially affected by a change in any *library unit* named by its *context clause.* A *secondary unit* is potentially affected by a change in the corresponding *library unit.* The *subunits* of a parent compilation unit are potentially affected by a change of the parent *compilation unit.* If a *compilation unit* is succeccfully recompiled, the *compilation units* potentially affected by this change are obsolete and must be recompiled unless they are no longer needed.

The *subunits* of a unit can be recompiled without affecting the unit itself. Similarly, changes in a *package body* do not affect other *compilation units* since the *package body* only has access to the *package specification.*

If any error is detected while attempting to compile a *compilation unit,* then the attempted compilation is rejected and it has no effect whatsoever on the program library; the same holds for recompilations (no compilation can become obsolete because of such a recompilation).

Elaboration of Library Units.

Before the execution of a *main program,* all *libray units* needed by the *main program* are elaborated, as well as the corresponding *library unit* bodies, if any. The *library units* needed by the *main program* are:

those named by *with clauses* applicable to the *main program,* and to its *subunits.*

those named by the *with clauses* applicable to these *library units* themselves, to the corresponding *library unit* bodies, and to their *subunits*; and so on, in a transitive manner.

The elaboration of these *library units* and of the corresponding *library unit* bodies is performed in an order consistent with the partial ordening defined by the *with clauses.* In addition, a *library unit* mentioned by the *context clause* of a *subunit* will be elaborated before the body of the ancestor unit of the *subunit.*

The program is illegal if no consistant order can be found (that is, if a circularity exists). The elaboration of the *compilation units* of the program is performed in some order that is otherwise not defined by the language.

Calling Das

To compile a *compilation unit,* submit (or re-submit) the UNIX file belonging to the unit to *das. Das* will maintain a library file containing information on the *compilation units* of the program library. This information includes intermediate filenames, output filename and other information pertaining to the order of previous compilations. The later is used for checks and is updated for each *compilation unit* successfully compiled. If any needed information is not found or found to be inconsistant, then *das* will abort execution.

A single program library is implied for the *compilation units* of the compilation. No linking and/or sharing program libraries is (as yet) available.

Output files.

The output will be put by default in file.t, file.o and if an executable is made, file.x. The .t, .o and .x files will be necessary for further compilations.

Option list processing.

Options which are capitalized act as toggles whereas other options must have an argument which follows. Options do not have to preceed the UNIX filenames.

Options are:

- **C:** compileonly. Normally *das* will compile and load the specified file. This option will supress the loader.
- **K:** keep intermediate files. After the compilation intermediate files will not be removed. Intermediate files are made by the preprocessor, frontend, backend, pc1, and optimizer. These should not be confused with the .t and .o files.
- **L:** loadonly. All .o files must exist.
- **N:** no run flag on. This option will print the calculated compilation / elaboration order of the unit. The compiler will not be executed.
- **O:** optimizer on. By default the optimizer will not be invoked.

- **P:** preprocessor on. By default the preprocessor will not be invoked.
- **S:** compile with a non default package *standard*. After a library file is created a package *standard* is specified to be the default. This option supresses the default package *standard* and will interactivly ask for a new one.
- **T:** print the attributed tree of the compilation unit
- **V:** verbose. Each pass of the compiler with its arguments is printed.
- **a** file:
  use *file* as the assembler instead of the default assembler.
- **b** file:
  use *file* as the backend instead of the default backend.
- **e** file: use *file* as the loader instead of the default loader.
- **f** file: use *file* as the frontend instead of the default frontend.
- **i** dir: use *dir* as the directory to place intermediate files instead of the default directory "/tmp".
- **l** file: link library *file* with the program. This file must be a .a file. The order of libraries which are to be linked can be relavent. The libraries daslib.a and libc.a are linked by default.
- **m** file:
  use *file* as the library file instead of the default ".master".
- **n** file:
  make *file* as the executable instead of the default .x file.
- **o** file:
  use *file* as the optimizer. This option implies the -O option.
- **p** file:
  use *file* as the preprocessor. This option implies the -P option.
- **t** dir: use *dir* as the tree directory as the directory to store the .t and .o files instead of the default directory ".".
- **s** file: use *file* as a *startup* file. When loading a unit a number of startup files are placed before the specified file. This option supresses the default startup file and uses *file* instead.

SYNTAX

compilation ::=
  {*compilation_unit*}

compilation_unit ::=
  context_clause library_unit |
  context_clause secondary_unit

context_clause ::=
  {*with_clause* {*use_clause*}}

with_clause ::=
  with unit_simple_name {, unit_simple_name};

library_unit ::=
  subprogram_specification |
  subprogram_body |
  package_specification

secondary_unit ::=
package_body | *subunit*

subprogram_body ::=
subprogram_specification **is** ...

package_specification ::=
package unit_simple_name **is** ...

package_body ::=
package body unit_simple_name **is** ...

subunit ::=
separate (unit_simple_name) *proper_body*

subprogram_specification ::=
procedure unit_simple_name *formal_part* |
function unit_simple_name *formal_part* **return** *type_mark*

EXAMPLES

1. das file1 file2 file3
will compile and load the named files.

2. das -CV -t dir file1 file2
will (verbose on) only compile the named files. The trees will be stored in the named directory.

3. das file1 -n file
will compile and load *file1*. The result will be *file*.

AUTHOR

Robert van Liere

FILES

file: input file
file.t: das tree file
file.o: object file
file.x: loaded output
/lib/cpp: optional preprocesser
~adacomp/lib/front: frontend
~adacomp/lib/backend: backend
~adacomp/lib/c1: backend of portable C compiler
/lib/c2: optional optimizer
~adacomp/lib/as: assembler
/bin/ld: loader
~adacomp/lib/t2: tree for package standard
~adacomp/lib/o2.o: object for package standard
~adacomp/lib/dasrt0.o: runtime startup
~adacomp/lib/dasexit.o: runtime exit
~adacomp/lib/daslib.a: standard das library
/lib/libc.a: standard C library
/tmp/????{a,b,c,d,e}: intermediate files

SEE ALSO

"ADA Reference Manual"
ANSI/MIL-STD-1815A; United States Department of Defense; January 1983.

"DAS Reference Manual"
Jan v. Katwijk, Delft University of Technology.

BUGS

When compiling a unit, *das* will lock the library file in order to disallow multiaccess. If *das* is aborted in an odd manner (such as: UNIX down) then the library will stay locked.

If a file with the same unit name as an existing compilation unit is entered in the library, the first compilation unit will be overwritten without warning.

NAME

date – print and set the date

SYNOPSIS

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

DESCRIPTION

If no argument is given, the current date and time are printed. If an argument is given, the current date is set. *yy* is the last two digits of the year; the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* is optional and is the seconds. For example:

```
date 10080045
```

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

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

utmp(5)

DIAGNOSTICS

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

NAME

dc – desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

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

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

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

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 and in registers are printed.

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

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

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

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

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

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

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

c
: All values on the stack are popped.

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

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

O pushes the output base on the top of the stack.

k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

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

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

; : are used by *bc* for array operations.

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

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

SEE ALSO

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

DIAGNOSTICS

'x is unimplemented' where x is an octal number.
'stack empty' for not enough elements on the stack to do what was asked.
'Out of space' when the free list is exhausted (too many digits).
'Out of headers' for too many numbers being kept around.
'Out of pushdown' for too many items on the stack.
'Nesting Depth' for too many levels of nested execution.

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

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

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

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

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

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

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

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

To skip over a file before copying from magnetic tape do

```
(dd of=/dev/null; dd of=x) </dev/rmt0
```

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

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

RESTRICTIONS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The 'ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

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

CAUTION

Image copying disks using *dd* requires careful selection of the block size (bs, ibs, obs). Using the default block size of 512 bytes (1b) will always work, however this is not very efficient. In most cases the optimum block size is the number of 512 byte sectors per track. For example: to copy an RL02 disk to another RL02 disk use 'bs=20b', or to copy an RP06 to an RM03 use 'ibs=22b obs=32b'. If the size of the input disk is larger that the size of the output disk a portion of the last record will not be transferred. An disk image copy using *dd* will always terminate with the message 'No such device or address', this indicates that the end of the disk has been reached.

NAME

ded – display editor

SYNOPSIS

ded [-b] [-d] [-s] [-t] [-u] [name]

DESCRIPTION

Ded is a display oriented text editor.

If a *name* argument is given, *ded* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ded's* buffer so that it can be edited. Ded takes a few switches, of which some are as follows:

-b : set auto-break mode (default on non-program-text files),
-d : replay dlog file,
-s : turn bell off (silence),
-t : set auto-tab mode (default on program-text files) and
-u : unset auto-break mode.

Commands are typed on the **command line** (type ESCAPE or DIAGONAL to enter this line) and are executed by typing RETURN, NEWLINE or <SEND,SEND>. In the commands below, a<sep> is either a slash ('/'), which initiates a search **down** the file from the current position, or a semicolon (';'),' which initiates a search **up** the file. All searches wrap-around the top/bottom of the file.

The *regular expressions* in ded are complex and powerful. The prime character introduces a special interpretation of a character, otherwise characters stand for themselves.

1. A regular expression (<re>) is a sequence of one or more match expressions (<me>). The <me>s are:

<character>	matches that character
''	matches a prime
'/	matches a slash
';	matches a semicolon
'^	matches beginning of line
'$	matches end of line (trailing blanks ignored)
'.	matches anything EXCEPT end of line
'<space>	matches any number of spaces and end-of-line
'[<set>']	matches any element of <set>
'[-<set>']	matches anything EXCEPT those in <set>
'(<re>')	matches <re> - useful for repetition of multiple characters and for labeling parts of an <re>
<me>'*	matches as many as possible (zero or more) occurrences of <me>
<me>'**	as '*, except it looks for as few as possible
<me>'+	as '*, except on or more occurrences
<me>'++	as '+, except it looks for as few as possible
<me>'?	looks for zero or one occurrence of <me>

2. A <set> is a *sequence* of set elements, which may be:

<char>	representing that character
<char>-<char>	representing a range of ASCII characters e.g. a-z, 0-9 etc.
'^	representing beginning of line
'$	representing ed of line
'/	representing slash
';	representing semicolon

Ranges and positions.

When a line is labele with a (lower case) letter, the letter appears in the left margin. One may refer to the range in later commands merely by giving the letter. A range may also be a dot ('.') meaning the current line, an up-arrow ('^') meaning the first line in the file or a dollar ('$') meaning the last line in the file.

A position is a range (meaning by default the LAST line in the range or the position after the last line in the range), or a range followed by '+' (meaning then same as the unadorned range) or a range followed by '-' (meaning the first line in the range or the position before the first line in the range).

The following commands are available:

tof,^:
go to the first line in file

eof,$:
go to last line in file

":
goto next screenful of file

&:
go to previous screenful of text

%:
move screen so that current line is in middle

d:
delete current text line (and the d command as well)

d <range>:
delete lines in <range>

<sep><re><sep>:
search for occurrence of <re> (the <sep> at the end is optional)

<sep>:
search again for last <re> mentioned

x<sep><re><sep><string><sep>:
find an occurrence of <re> in the file, replace it by <string>, ask for user confirmation (the <sep> at the end is optional)

x:
repeat the last <re>,<string> combination

s<sep><re><sep><string><sep>
replace every occurrence of <re> in the file by <string> (the <sep> at the end is optional)

s:
repeat last <re>, <string> combination

w:
write the edited file and make a .old backup copy of the original (unedited) file

reallywrite:
force a 'w' even though the edited file hasn't changed

w <range> <filename>:
write lines from <range> into file <filename>

wd <range> <filename>:
write lines, then delete them from the edited file

dw <range> <filename>:
same as wd

a <range> <filename>:
append lines from range to file <filename>

ad <range> <filename>:
append lines, then delete them

da <range> <filename>:
same as ad

r <position> <filename>:
read text from file <filename> and place it at <position> in the edited file

ok:
write the file (as for 'w') and exit

q:
just exit, if the file hasn't changed

reallyquit:
force a 'q', even though the file may have changed

b+, b-:
set/unset auto-break mode

t+, t-:
set/unset auto-tab mode

bell+ , bell-:
turn bell on/off

!<text>:
try to execute <text> as a UNIX command

!:
execute the last UNIX command mentioned

-:
shell out into sh

@:
shell out into newsh

.<letter>:
label current line as <range <letter>

f <letter>:
label current line as first of range <letter>

l <letter>:
label current line as last of range <letter>

= <range>:
give information about <range>

<number>:
go to line number <number>

go to position <position>

m <range> <position>:
move lines in <range> to <position>

c <range> <position>:
copy lines from <range> to <position>

p:
redraw the screen

The string used in an 's' or 'x' command is normally a sequence of characters, but may contain any of the following sequences:

'$ insert a newline
'& insert a copy of the text matched by the <re>
'(<n> (where <n> is a digit in the range 1-9) insert a copy of the text matched by the <n>'th bracketed <re> in the matched expression

After an 'x' command has found an instance of <re> *ded* will show the effect of the replacement and ask 'ok?'. You may reply

y accept the replacement
n reject the replacement
<ESC> accept the replacement and continue to search for another
<DEL> reject the replacement but continue to search for another

The following *cursor* control keys are available:

wordright / ^A:
move to start of next word

wordleft / ^B:
move to start of current/previous word

^C:
interrupt search, s or x commands

down / ^E:
move down a line

^F:
erase tail of line

^G:
erase up to start of next word

^I:
move to next tab position, pushing tail of line

linefeed / ^J:
move to start of next line

^K:
move to first non-blank character on line

^L:
move to afer last non-blank character on line

return / ^M:
split line, insert a line break

right / ^Q:
move right one character

^R:
delete character under cursor

up / ^S:

move up a line

left / ^U:
move left a character

send / home / ^V:
special interpretation of next control key

```
send,wordrightsame as ^L
send,wordleft  same as ^K
send,down              goto bottom line
send,right      goto right edge of screen
send,left              goto left edge of screen
send,up                goto top line of screen
send,send              execute command shown in command line
```

^W:
delete up to start of current word

^X:
delete head of line

diagonal / ^Y:
switch from text -> command line or vice versa

^Z:
insert a control mark

ESC / ^[:
erase current contents of command line, move cursor to start of command line

DEL:
rubout character before cursor, or if at left of screen join to previous line (rubout newline)

BUGS

Ded has a habit of replacing tabs by spaces; this is annoying in program sources and fatal in makefiles.

NAME

delta − make a delta (change) to an SCCS file

SYNOPSIS

delta [−rSID] [−s] [−n] [−glist] [−m[mrlist]] [−y[comment]] [−p] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see **−m** and **−y** keyletters below).

Keyletter arguments apply independently to each named file.

−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 *get*s for editing (**get −e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **−r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

−s Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

−n Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

−g*list* Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

−m[*mrlist*] If the SCCS file has the **v** flag set (see *admin*(1)) then a Modification Request (**MR**) number *must* be supplied as the reason for creating the new delta.

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

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

Note that if the **v** flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

−y[*comment*] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

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

−**p** Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff*(1) format.

FILES

All files of the form *?*-file are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the 'gotten' file and the *g-file*.

WARNINGS

Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is escaped. This character has special meaning to SCCS (see *sccsfile*(5)) and will cause an error.

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

If the standard input (−) is specified on the *delta* command line, the −**m** (if necessary) and −**y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff − remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**−w**] file ...

DESCRIPTION

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

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

SEE ALSO

troff(1), eqn(1), tbl(1)

RESTRICTIONS

Deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.

NAME

df – disk free space

SYNOPSIS

df [**–l**] [**–d**] [filesystems] ...

DESCRIPTION

Df prints out the number of free blocks available on the *filesystems*. If no filesystem is specified, a tabled listing of all known filesystems is produced, containing the size and free counts for blocks and inodes. Also the mount information is given with the comment from the */etc/checklist* file.

Options are:

–l Give free inodes on *filesystems* to.

–d Recalculate the free count by following the freelist on *filesystems*. Normally the free count is taken from the superblock.

FILES

/etc/checklist table of known devices
/etc/mtab table of mounted filesystems
/dev/* devices

SEE ALSO

quot(1), du(1), checklist(5)

BUGS

Very deep mounted directories give garbled output.

NAME

df – disk free

SYNOPSIS

df [filesystem] ...

DESCRIPTION

Df prints out the number of free blocks available on the *filesystems*. If no file system is specified, the free space on all of the normally mounted file systems is printed.

FILES

Default file system names obtained from the */etc/fstab* file.

SEE ALSO

icheck(1), fsck(1), fstab(5)

NAME

diction – print wordy sentences
suggest – interactive thesaurus for diction

SYNOPSIS

diction [**–ml**] [**–mm**] [**–n**] [**–f** pfile] file ...
suggest

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **–ms** may be overridden with the flag **–mm**. The flag **–ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **–f** *pfile*. If the flag **–n** is also supplied the default file will be supressed.

Suggest is an interactive thesaurus for the phrases found by diction.

FILES

/usr/etc/diction.d – diction phrases
/usr/etc/suggest.d – suggest phrases

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

diff − differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is '−', the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* (*file1*) is used. The normal output contains lines of these forms:

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

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

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

The **−b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **−e** option produces a script of *a*, *c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **−f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

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

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **−h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **−e** and **−f** are unavailable with **−h**.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

cmp(1), comm(1), ed(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

RESTRICTIONS

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

NAME

diff3 − 3-way differential file comparison

SYNOPSIS

diff3 [−**ex3**] file1 file2 file3

DESCRIPTION

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

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

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

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

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

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

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

FILES

/tmp/d3?????
/usr/lib/diff3

SEE ALSO

diff(1)

RESTRICTIONS

Text lines that consist of a single '.' will defeat −**e**.
Files longer than 64K bytes won't work.

NAME

dint, dcan – print diroff-output on the Canon Laserprinter

SYNOPSIS

dint
dcan

DESCRIPTION

Dint prints the output of a diroff-job on the Canon Laserprinter when the elite or courier typesetters are used.

Dcan does the same for diroff-output ment for the canon typesetter.

Both *dint* and *dcan* read from the standard input.

SEE ALSO

croff(1TU), lbp(1)

NAME

du – summarize disk usage

SYNOPSIS

du [**–s**] [**–a**] [name ...]

DESCRIPTION

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

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

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

RESTRICTIONS

Non-directories given as arguments (not under **–a** option) are not listed.
If there are too many distinct linked files, *du* counts the excess files multiply.

NAME

echo – echo arguments

SYNOPSIS

echo [**–n**] [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a newline on the standard output. If the flag **–n** is used, no newline is added to the output.

Echo is useful for producing diagnostics in shell programs and for writing constant data on pipes. To send diagnostics to the standard error file, do 'echo ... 1>&2'.

NAME

ed – text editor

SYNOPSIS

ed [–] [name]

red [–] [name]

DESCRIPTION

Ed is the standard text editor.

If a *name* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed's* buffer so that it can be edited. The optional – suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command.*

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

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command.* Attempts to bypass these restrictions result in an error message (*restricted shell*).

Commands to *ed* have a simple and regular structure: zero or more *addresses* followed by a single character *command,* possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Missing addresses are supplied by default.

In general, only one command may appear on a line. Certain commands allow the addition of text to the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period '.' alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. In the following specification for regular expressions the word 'character' means any character but newline.

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

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

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character regular expression that matches itself.

1.2 A backslash (\) followed by any special character is a one-character regular expression that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* regular expression (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (currency symbol), which is special at the *end* of an entire regular expression (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire regular expression, which is special for that regular expression (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character regular expression that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character regular expression that matches any *one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character regular expression matches any character *except* newline and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (−) may be used to indicate a range of consecutive ASCII characters; for example, **[0−9]** is equivalent to **[0123456789]**. The − loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial , if any); e.g., **[]a−f]** matches either a right square bracket (]) or one of the letters **a** through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

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

2.1 A one-character regular expression is a regular expression that matches whatever the one-character regular expression matches.

2.2 A one-character regular expression followed by an asterisk (*) is a regular expression that matches *zero* or more occurrences of the one-character regular expression. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character regular expression followed by \{*m*\}, \{*m*,\}, or \{*m*,*n*\} is a regular expression that matches a *range* of occurrences of the one-character regular expression. The values of *m* and *n* must be non-negative integers less than 256; \{*m*\} matches *exactly m* occurrences; \{*m*,\} matches *at least m* occurrences; \{*m*,*n*\} matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the regular expression matches as many occurrences as possible.

2.4 The concatenation of regular expressions is a regular expression that matches the concatenation of the strings matched by each component of the regular expression.

2.5 A regular expression enclosed between the character sequences \(and \) is a regular expression that matches whatever the unadorned regular expression matches.

2.6 The expression *n* matches the same string of characters as was matched by an expression enclosed between \(and \) *earlier* in the same regular expression. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of \(counting from the left. For example, the expression **\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

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

3.1 A circumflex (^) at the beginning of an entire regular expression constrains that regular expression to match an *initial* segment of a line.

3.2 A currency symbol ($) at the end of an entire regular expression constrains that regular expression to match a *final* segment of a line.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

The construction *entire regular expression*$ constrains the entire regular expression to match the entire line.

The null regular expression (e.g., //) is equivalent to the last regular expression encountered. See also the last paragraph before *FILES* below.

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

1. The character '.' addresses the current line.
2. The character '$' addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the name *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.
5. A regular expression enclosed in slashes '/' addresses the line found by searching forward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the beginning of the buffer. See also the last paragraph before *FILES* below.
6. A regular expression enclosed in queries '?' addresses the line found by searching backward from the current line and stopping at the first line containing a string that matches the regular expression. If necessary the search wraps around to the end of the buffer. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign '+' or a minus sign '−' followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with '+' or '−' the addition or subtraction is taken with respect to the current line; e.g. '−5' is understood to mean '.−5'.
9. If an address ends with '+' or '−', then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character in addresses is entirely equivalent to −.) Moreover, trailing '+' and '-' characters have a cumulative effect, so '− −' refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair **1,$**, while a semicolon (;) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient ones are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are

the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n** or **p**, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a
<text>
.

The append command reads the given text and appends it after the addressed line. '.' is left on the last line input, if there were any, otherwise at the addressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer. See also the last paragraph before *FILES* below.

(.,.)c
<text>
.

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

(.,.)d

The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no 'filename' is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; 'filename' is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If 'filename' is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E filename

This command is the same as *e*, except that no diagnostic results when no *w* has been given since the last buffer alteration.

f filename

The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

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

In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. *A*, *i*, and *c* commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The *g*, *G*, *v*, and *V* commands are *not* permitted in the command list. See also *RESTRICTIONS* and the last paragraph before *FILES* below.

(1,$)G/ regular expression /

In the interactive *G*lobal command, the first step is to mark every line that matches the given regular expression. Then, for every such line, that line is printed, '.' is

changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an '&' causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The *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>
.

This command inserts the given text before the addressed line. '.' is left at the last line input, or, if there were none, at the line before the addressed line. This command differs from the *a* command only in the placement of the text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j
This command joins the addressed lines into a single line; intermediate newlines simply disappear. '.' is left at the resulting line.

(.)k*x*
The mark command marks the addressed line with name *x*, which must be a lowercase letter. The address '*x* then addresses this line; '.' is unchanged.

(.,.)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in two-digit octal, and long lines are folded. The *l* command may be placed on the same line after any non-i/o command.

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

(.,.)n
The number command prints the addressed lines, preceding each line by its line number and a tab character; '.' is left at the last line printed. The number command may placed on the same line after any non-i/o command.

(.,.)p
The print command prints the addressed lines. '.' is left at the last line printed. The *p* command may be placed on the same line after any non-i/o command.

P The editor will prompt with an asterisk (*) for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

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

Q This command is the same as *q*, except that no diagnostic results when no *w* has been

given since the last buffer alteration.

($)r filename
The read command reads in the given file after the addressed line. If no 'filename' is given, the remembered file name, if any, is used (see *e* and *f* commands). The file name is remembered if there was no remembered file name already. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file. If 'filename' is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

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

The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '.' is left at the last line substituted. See also the last paragraph before *FILES* below.

An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. The characters '*n*' where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left.

Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'. Such substitution cannot be done as part of a *g*, *G*, *v*, or *V* command list.

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

(1,$)V/ regular expression /
This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the regular expression.

(1,$)w filename
The *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 'filename' is *not* changed unless *filename* is the very first file name mentioned since *ed* was invoked. If no 'filename' is given, the currently remembered 'filename', if any, is used (see *e* and *f* commands); '.' is unchanged. If the command is successful, the number of characters written is typed. If filename is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

(1,$)W filename

This command is the same as *w*, except that the addressed lines are appended to the file.

($)=

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

!*shell command*

The remainder of the line after the ! is sent to the UNIX System shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; '.' is unchanged.

(.+1)<newline>

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

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

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

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a regular expression or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e*

ed.hup work is saved here if terminal hangs up

SEE ALSO

grep(1), sed(1), sh(1), stty(1)

B. W. Kernighan, *A Tutorial Introduction to the ED Text Editor*

B. W. Kernighan, *Advanced editing on UNIX*

DIAGNOSTICS

'?name' for inaccessible file; '?' for errors in commands; '?TMP' for temporary file overflow.
(use the *h*elp and *H*elp commands for detailed explanations).

To protect against throwing away valuable work, a *q* or *e* command is considered to be in error, unless a *w* has occurred since the last buffer change. A second *q* or *e* will be obeyed regardless.

RESTRICTIONS

The *l* command mishandles DEL.
A *!* command cannot be subject to a *g* or a *v* command.
The *!* command and the ! escape from the *e*, *r*, and *w* commands cannot be used if the the editor is invoked from a restricted shell (see *sh*(1)).
Because 0 is an illegal address for a *w* command, it is not possible to create an empty file with *ed*.
Characters are masked to 7 bits on input.

NAME

em editor for mortals

SYNOPSIS

em [ ] [-] [-elpqr] [name]

DESCRIPTION

Em is a Queen Mary College development of the standard UNIX text editor - ed. *Em* is a complete superset of *ed* except that it normally prompts command lines with a '>'. For those who prefer *ed's* silence, on the QMC UNIX system the *ed* command just invokes *em* without prompts. Other ways of controlling prompts are described below.

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

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

Commands to *em* have a simple and regular structure: zero or more *addresses* followed by a single character *command*, possibly followed by parameters to the command. These addresses specify one or more lines in the buffer. Every command which requires addresses has default addresses, so that the addresses can often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *em* 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.

The principal advantage of *em* over the original *ed* is that it also has an additional mode of text input and editing called open mode. This includes a form of editing editing within lines and while entering new text. This is described below under the *o* command. On fast terminals (anything faster than 1200 baud) open mode is generally more effective than input mode. (i.e. it can be used instead of the *a* , *c* and *i* commands which are retained in *em* mainly for use on slower terminals.) In open mode input is terminated not by a period but by an ESCAPE or a ^D code.

Em supports a limited form of *regular expression* notation. A regular expression specifies a set of strings of characters. A member of this set of strings is said to be *matched* by the regular expression. The regular expressions allowed by *em* are constructed as follows:

1. An ordinary character (not one of those discussed below) is a regular expression and matches that character.

2. A circumflex '^' at the beginning of a regular expression matches the empty string at the beginning of a line.

3. A currency symbol '$' at the end of a regular expression matches the null character at the end of a line.

4. A period '.' matches any character except a new-line character.

5. A regular expression followed by an asterisk '*' matches any number of adjacent

occurrences (including zero) of the regular expression it follows.

6. A string of characters enclosed in square brackets '[ ]' matches any character in the string but no others. If, however, the first character of the string is a circumflex '^' the regular expression matches any character except new-line and the characters in the string.

7. The concatenation of regular expressions is a regular expression which matches the concatenation of the strings matched by the components of the regular expression.

8. A regular expression enclosed between the sequences '\(' and '\)'is identical to the unadorned expression; the construction has side effects discussed under the *s* command.

9. The null regular expression standing alone is equivalent to the last regular expression encountered.

Regular expressions are used in addresses to specify lines and in one command (see *s* below) to specify a portion of a line which is to be replaced. If it is desired to use one of the regular expression metacharacters as an ordinary character, that character may be preceded by '\'. This also applies to the character bounding the regular expression (often '/') and to '\' itself.

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

1. The character '.' addresses the current line.

2. The character '$' addresses the last line of the buffer.

3. A decimal number *n* addresses the *n*-th line of the buffer.

4. ''*x*' addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.

5. A regular expression enclosed in slashes '/' addresses the first line found by searching toward the end of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the beginning of the buffer.

6. A regular expression enclosed in queries '?' addresses the first line found by searching toward the beginning of the buffer and stopping at the first line containing a string matching the regular expression. If necessary the search wraps around to the end of the buffer.

7. An address followed by a plus sign '+' or a minus sign '' followed by a decimal number specifies that address plus (resp. minus) the indicated number of lines. The plus sign may be omitted.

8. If an address begins with '+' or '−' the addition or subtraction is taken with respect to the current line; e.g. '−5' is understood to mean '.−5'.

9. If an address ends with '+' or '−', then 1 is added (resp. subtracted). As a

consequence of this rule and rule 8, the address '−' refers to the line before the current line. Moreover, trailing '+' and '−' characters have cumulative effect, so '− −' refers to the current line less 2.

10. To maintain compatibility with earlier version of the editor, the character '^' in addresses is entirely equivalent to '−'.

Commands may require zero, one, or two addresses. Commands which require no addresses regard the presence of an address as an error. Commands which accept one or two addresses assume default addresses when insufficient are given. If more addresses are given than such a command requires, the last one or two (depending on what is accepted) are used.

Addresses are separated from each other typically by a comma ','. They may also be separated by a semicolon ';'. In this case the current line '.' is set to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward searches ('/', '?'). The second address of any two-address sequence must correspond to a line following the line corresponding to the first address.

In the following list of *em* commands, the default addresses are shown in parentheses. The parentheses are not part of the address, but are used to show that the given addresses are the default.

As mentioned, it is generally illegal for more than one command to appear on a line. However, any command may be suffixed by 'p' or by 'l', in which case the current line is either printed or listed respectively in the way discussed below. Although the command letters are listed below in lower-case, commands to *em* may be typed as upper- or lower-case letters.

(.)a <text> **The append command reads the given text and appends it after the addressed line. if there were any, otherwise at the adressed line. Address '0' is legal for this command; text is placed at the beginning of the buffer.**

(. , .)c <text> **The change command deletes the addressed lines, then accepts input text which replaces these lines.** '.' is left at the last line input; if there were none, it is left at the first line not deleted.

(. , .) d
The delete command deletes the addressed lines from the buffer. The line originally after the last line deleted becomes the current line; if the lines deleted were originally at the end, the new last line becomes the current line.

e filename
The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in. '.' is set to the last line of the buffer. The number of characters read is typed. 'filename' is remembered for possible use as a default file name in a subsequent *r* or *w* command.

f filename
The filename command prints the currently remembered file name. If 'filename' is given, the currently remembered file name is changed to 'filename'.

(1,$)g/regular expression/command list
In the global command, the first step is to mark every line which matches the given regular expression. Then for every such line, the given command list is executed with '.' initially set to that line. A single command or the first of multiple commands appears on the same line with the global command. All lines of a multi-line list except the last line must be ended with '\'. *A*, *i*, and *c* commands and associated input are permitted; the '.' terminating input mode may be omitted if it would be on the last line of the command list. The (global) commands, *g*, and *v*, are not permitted in the command list.

h
The help command *h* displays a summary of the commands available in *em*. Help on certain commands is also available by typing *h x* where *x* is the command letter. The texts reside in /usr/lib/emhelp, and are designed to fill the screen of ITT terminals.

(.)i <text>
This command inserts the given text before the addressed line. '.' is left at the last line input; if there were none, at the addressed line. This command differs from the *a* command only in the placement of the text.

(. , .+1)j
The join command converts several lines into a single line containing the concatenation of the original lines. A space is interposed between the contents of each line. The resulting line may be up to 512 characters long. If *j* is used without line addresses, then the current line is joined to the following one.

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

(. , .)l
The list command prints the addressed lines in an unambiguous way: non-graphic characters are printed in octal, and long lines are folded. An *l* command may follow any other on the same line.

(. , .)m *a*
The move command repositions the addressed lines after the line addressed by *a*. The last of the moved lines becomes the current line. (. , .)o (. , .)o/regular expression/ (. , .)o+ (. , .)o; (. , .)o- (. , .)o:
The open command provides an additional level of editing for interactive corrections during input of new lines of text and for alterations to existing lines. The first two forms of the *o* command operate on an existing line or lines. Each line in the range is opened successively for modification with the cursor positioned at the first match of the regular if one is given, otherwise at the start of the line.

The commands *o+* , *o-* and *o:* are equivalent to the *a*, *i* and *c* commands respectively, except in their use of the control keys listed below, and in that they are not terminated by a line containing a single '.'. *o;* is equivalent to *o+* . In all variants of the *o* command it is possible to position the cursor within the line, to delete characters, words or portions of the line, and to insert new text at any point, using control keys as follows:

char: advance ^Q [->] line: display to end ^E [ ]

```
     backup ^U [<-]          reset to start ^S [^]
     delete  RUBOUT            delete backward ^X
word:  advance ^A [>>]           delete forward  ^F
     backup  ^B [<<]          release margin  ^G
     delete  ^W              re-type       ^R
     spell   ^V [SEND]   exit:  with new text   ESCAPE or ^D
                         unchanged text  ^C
```

Other characters (including RETURN) are inserted as typed; the symbols in [] are the equivalent BLUE keys on QMC ITT terminals.

In open mode, diagnostic information is provided on ITT terminals in the form of a backward '?' when editing control keys are used inappropriately. When the automatic break facility is operative, a bell is transmitted to a few characters before a line break is made. The break may be temporarily inhibited by the '^G' key (*cf* the 'margin release' key on typewriter keyboards.)

(. , .)p
The print command prints the addressed lines. '.' is left at the last line printed. The *p* command may be placed on the same line after any command.

q
The quit command causes *em* to exit. No automatic write of a file is done, but if a write of the whole buffer has not occured since the file was altered, a warning 'sure?' is printed on the first attempt to quit, and the quit does not occur.

($)r filename
The read command reads in the given file after the addressed line. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). The remembered file name is not changed unless 'filename' is the very first file name mentioned. Address '0' is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed. '.' is left at the last line read in from the file.

(. , .)s/regular expression/replacement/ or, (. , .)s/regular expression/replacement/g
The substitute command searches each addressed line for an occurrence of the specified regular expression. On each line in which a match is found, all matched strings are replaced by the replacement specified, if the global replacement indicator 'g' appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of '/' to delimit the regular expression and the replacement. '.' is left at the last line substituted. An ampersand '&' appearing in the replacement is replaced by the string matching the regular expression. The special meaning of '&' in this context may be suppressed by preceding it by '\'. 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 enclosed between '\(' and '\)'. When nested, parenthesized subexpressions are present, *n* is determined by counting occurrences of '\(' starting from the left. Lines may be split by substituting new-line characters into them. The new-line in the replacement string must be escaped by preceding it by '\'. It is possible to obtain a count of the number of replacements performed by postfixing the command with an 'n'.

(. , .) 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 on the last line of the copy.

(1,$)v/regular expression/command list
This command is the same as the global command except that the command list is executed with '.' initially set to every line *except* those matching the regular expression.

(1,$)w filename
The write command writes the addressed lines onto the given file. If the file does not exist, it is created mode 666 (readable and writeable by everyone). The remembered file name is *not* changed unless 'filename' is the very first file name mentioned. If no file name is given, the remembered file name, if any, is used (see *e* and *f* commands). '.' is unchanged. If the command is successful, the number of characters written is typed.

(. , .)x/regular expression/replacement/
The exchange command performs exactly the same functions as an equivalent substitute command 's' with the global postfix *g* , except that it enables the user to determine interactively whether or not each replacement is to be performed. Each line containing a matched string is displayed, with the matching characters underscored with '^' characters. The user must type a response line containing a '.' to perform the replacement. An empty response line will omit it.

It is possible to obtain a count of the number of replacements performed by postfixing the command with an 'n'.

(.)=
The line number of the addressed line is typed. '.' is unchanged by this command.

!UNIX command
The remainder of the line after the '!' is sent to UNIX to be interpreted as a command. '.' is unchanged.

Two additional facilities are available under the '!' command: '!!'repeats the last command executed in ! mode and '%' anywhere in a '!' line is replaced by the currently selected filename.

> <
> turns prompting off, < turns it on again.

(.) "
The " command is equivalent to '.+1,.+16p'

(.) %
The % command is equivalent to *.-6,.+ 6p* , except that the value of '.' is unchanged by it, and the current line is separated from its neighbours on the screen by a line of '-'s.

(.) &
This is equivalent to *.-16,.p*

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

Interrupts (^C at QMC) generally abort the current command, with the text of the current line as it was before the command started, but any preceding lines retain their alterations. End of file (^D) characters from the terminal are ignored (except in open mode), but hangup signals (such as are generated when a dial-up connection is accidentally broken) cause a quit. When *em* is used with a standard input other than the terminal (e.g. a file containing editor commands), end of file causes a quit from the editor.

Recovery: Whenever you are using *em* all of the keyboard input is saved in a log file. This is done in order to enable you to recover from system crashes and from your own errors. You can suppress the log file by invoking the editor with a -l flag. The name of the log file is guaranteed to be unique, is derived from the saved file name, and contains the string '.elog'. Whenever you write to the current saved file or change the current edit file the content of the log is erased. When you quit the editor normally, the log file is removed, but you can save it by giving a filename in the 'q' command. After a system crash or modem hangup the log file remains intact. To recover your edited workfile, invoke the editor with '-r' as the first argument and the name of the log file as the second. The log file will be 'replayed', leaving you ready to continue editing. If you wish to achieve a different result, you can edit the log file first. It is often a good idea to remove all of the lines in the log file that commence with a '|'. The log file is present except when the editor is invoked with a -l or -e flag, or with standard input other than from a terminal.

The -e flag is useful when the editor is invoked from another program, see for example the *send* command. The principal effects of -e are to inhibit the 'w' and 'e' commands, and to cause an automatic write when 'q' is invoked.

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

FILES

/usr/lib/emhelp/* help files available with *h*.
/tmp/#, temporary; '#' is the process number (in octal).
xxxx.elog, the log of keyboard input while file 'x' is being edited.

DIAGNOSTICS

Unsuccessful searches for strings are indicated by '??'. '?' for other errors in commands; 'TMP' for temporary file overflow.

SEE ALSO

An Introduction to the QMC Unix Editor - 'em' (George Coulouris), A Tutorial Introduction to the ED Text Editor (B. W. Kernighan), grep(I), spell(I)

BUGS

The *s* command causes all marks to be lost on lines changed. The underscoring in the 'x' command is sometimes incorrect on ITT terminals, especially when the line contains characters represented by compounds with '\'.

NAME

eqn (PDP only), neqn, checkeq (PDP only) typeset mathematics

SYNOPSIS

eqn [**dxy**] [**pn**] [**sn**] [**fn**] [file] ...
checkeq [file] ...

DESCRIPTION

Eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

eqn file ... troff
neqn file ... nroff

If no files are specified, these programs reads from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **d***xy* or (more commonly) with delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde represents a full space in the output, circumflex half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup**. Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 + bx + c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with *lim from {n -> inf} sum from 0 to n x sub i*.

Left and right brackets, braces, etc., of the right height are made with **left** and **right**: *left [x sup 2 + y sup 2 over alpha right] ~ = ~ 1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and for nothing at all (useful for a right-side-only bracket).

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

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol {1 above 2} }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot, dotdot, hat, tilde, bar, vec, dyad,** and **under**: *x dot* = *f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar* ~ = ~ *n under* is $\ddot{\underline{y}} = \underline{n}$, and *x vec* ~ = ~ *y dyad* is $\vec{x} = \vec{y}$.

Sizes and font can be changed with **size** *n* or **size** ± *n*, **roman, italic, bold,** and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **-s***n* and **-f***n*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like *sum* ($\sum$) *int* ($\int$) *inf* (∞) and shorthands like >= ($\geq$) -> ($\rightarrow$), and != ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *Troff*(1) four-character escapes like \(bs(Ø) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

troff(1), tbl(1), , eqnchar(7)

B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*

J. F. Ossanna, *NROFF/TROFF User's Manual*

NAME

ex, edit – text editor

SYNOPSIS

ex [–] [–**v**] [–**r**] [+*command*] [–**l**] name ...
edit [ex options]

DESCRIPTION

Ex is the root of a family of editors: *edit, ex* and *vi. Ex* is a superset of *ed,* with the most notable extension being a display editing facility. Display based editing is the focus of *vi.*

If you have not used *ed,* or are a casual user, you will find that the editor *edit* is convenient for you. It avoids some of the complexities of *ex* used mostly by systems programmers and persons very familiar with *ed.*

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex.*

DOCUMENTATION

The document *Edit: A tutorial* provides a comprehensive introduction to *edit* assuming no previous knowledge of computers or the UNIX system.

The *Ex Reference Manual – Version 3.5* is a comprehensive and complete manual for the command mode features of *ex,* but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed;* the material in the introductory and advanced documents works also with *ex.*

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi.* All of these documents can be found in volume 2c of the Programmer's Manual. In addition, the *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction.*

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
~/.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), grep(1), vi(1), termcap(5), environ(5)

AUTHOR

Originally written by William Joy
Mark Horton has maintained the editor since version 2.7, adding macros, support for many unusual terminals, and other features such as word abbreviation mode.

RESTRICTIONS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '−' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files, and cannot appear in resultant files.

NAME

expand – expand tabs to spaces

SYNOPSIS

expand [–tabstop] [–tab1,tab2,...,tabn] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

AUTHOR

Bill Joy

BUGS

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arg ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | *expr*
: yields the first *expr* if it is neither null nor '0', otherwise yields the second *expr*.

expr & *expr*
: yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr relop expr
: where *relop is one of* < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

expr + *expr*
expr - *expr*
: addition or subtraction of the arguments.

expr * *expr*
expr / *expr*
expr % *expr*
: multiplication, division, or remainder of the arguments.

expr : *expr*
: The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(*expr*)
: parentheses for grouping.

Examples:

To add 1 to the Shell variable *a*:

```
a=`expr $a + 1`
```

To find the filename part (least significant part) of the pathname stored in variable *a*, which may or may not contain '/':

```
expr $a : '.*/\(.*\)' '|' $a
```

Note the quoted Shell metacharacters.

SEE ALSO

ed(1), sh(1), test(1)

DIAGNOSTICS

Expr returns the following exit codes:

0 if the expression is neither null nor '0',
1 if the expression is null or '0',
2 for invalid expressions.

NAME

f77 – Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

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

The following options have the same meaning as in *cc*(1). See *ld*(1) for load-time options.

–c Suppress loading and produce '.o' files for each source file.

–p Prepare object files for profiling, see *prof*(1).

–O Invoke an object-code optimizer.

–S Compile the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

–f Use a floating point interpreter (for PDP11's that lack 11/70-style floating point).

–o output
Name the final output file *output* instead of 'a.out'.

–V Produce code suitable for using in overlaid programs. This is a default flag.

–V7 Turn off the **–V** flag. This is provided for backwards compatibility with older versions of *f77*, but in general should not be used.

The following options are peculiar to *f77*.

–onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

–u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

–C Compile code to check that subscripts are within declared array bounds.

–w Suppress all warning messages. If the option is '–w66', only Fortran 66 compatibility warnings are suppressed.

–F Apply EFL and Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.

–m Apply the M4 preprocessor to each '.r' or '.e' file before transforming it with the Ratfor or EFL preprocessor.

–E*x* Use the string *x* as an EFL option in processing '.e' files.

–R*x* Use the string *x* as a Ratfor option in processing '.r' files.

–N[qxscn]*nnn*
Make static tables in the compiler bigger. The compiler will complain if it overflows

its tables and suggest you apply one or more of these flags. These flags have the following meanings:

q Maximum number of equivalenced variables. Default is 150.

x Maximum number of external names (common block names, subroutine and function names). Default is 200.

s Maximum number of statement numbers. Default is 201.

c Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.

n Maximum number of identifiers. Default is 401.

Programs do not always require the full default space assigned to them. Accordingly, one of these default values may be reduced in order to accommodate an increased value for some other flag (see table below). In the case of much larger programs, static table space is at a premium. Therefore, experimentation with different values for these flags is sometimes required so that the compiler will not complain.

Storage costs associated with these optional flags are:

flag	default value	cost ea.	default cost
s	201	8 bytes	1608 bytes
q	150	12 bytes	1800 bytes
x	200	20 bytes	4000 bytes
c	20	18 bytes	360 bytes
n	401	4 bytes	1604 bytes

−T[12alFM]*file*

Use alternate programs for various passes of compilation.

1*file* Use *file* for the back end of the compiler, instead of */lib/c1*.

2*file* Use *file* as the optimizer, instead of */lib/c2*.

a*file* Use *file* as the assembler, instead of */bin/as*.

l*file* Use *file* as the loader, instead of */bin/ld*.

F*file* Use *file* as the footname (startup code), instead of */lib/crt0*.

M*file* Use *file* as the macro processor with the **−m** flag, instead of */usr/bin/m4*.

Other arguments are taken to be either loader option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name 'a.out'.

FILES

file.[fresc]	input file
file.o	object file
a.out	loaded output
/usr/lib/f77pass1	compiler
/lib/c1	pass 2
/lib/c2	optional optimizer
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3

SEE ALSO

S. I. Feldman, P. J. Weinberger, *A Portable Fortran 77 Compiler*
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

RESTRICTIONS

The Fortran 66 subset of the language has been exercised extensively; the newer features have not.

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

factor [number]

primes

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 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 1 minute to factor a prime near 10^{14} on a PDP11.

When *primes* is invoked, it waits for a number to be typed in. If you type in a positive number less than 2^{56} it will print all primes greater than or equal to this number.

DIAGNOSTICS

'Ouch.' for input out of range or for garbage input.

NAME

file – determine file type

SYNOPSIS

file file ...

DESCRIPTION

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

RESTRICTIONS

It often makes mistakes. In particular it often suggests that command files are C programs.

NAME

find – find files

SYNOPSIS

find pathname-list expression

DESCRIPTION

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

–name filename
: True if the *filename* argument matches the current file name. Normal Shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

–perm onum
: True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared: *(flags&onum)= = onum.*

–type c
: True if the type of the file is *c*, where *c* is **b**, **c**, **d** or **f** for block special file, character special file, directory or plain file.

–links n
: True if the file has *n* links.

–user uname
: True if the file belongs to the user *uname* (login name or numeric user ID).

–group gname
: True if the file belongs to group *gname* (group name or numeric group ID).

–size n
: True if the file is *n* blocks long (512 bytes per block).

–inum n
: True if the file has inode number *n*.

–atime n
: True if the file has been accessed in *n* days.

–mtime n
: True if the file has been modified in *n* days.

–exec command
: True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

–ok command
: Like **–exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

–print
: Always true; causes the current pathname to be printed.

–newer file
: True if the current file has been modified more recently than the argument *file*.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).
2) The negation of a primary ('!' is the unary *not* operator).

3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

4) Alternation of primaries ('**-o**' is the *or* operator).

EXAMPLE

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

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

FILES

/etc/passwd
/etc/group

SEE ALSO

sh(1), test(1), filsys(5)

RESTRICTIONS

The syntax is painful.

NAME

finger – user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by *finger* whenever a list of peoples names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

Finger options include:

–b Slightly briefer version (long format).

–f Suppress the printing of the header line (short format).

–h Suppress printing of the *.project* files.

–i Quick list (–q) with idle times.

–l Force long output format.

–p Suppress printing of the *.plan* files.

–q Quick list (similar to *who*(1)).

–s Force short output format.

–w Narrow format output (short format).

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, phones, directories, and shells
~/.plan	plans
~/.project	projects

SEE ALSO

who(1)

AUTHOR

Earl T. Cohen

RESTRICTIONS

Only the first line of the *.project* file is printed.

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [–width] [file ...]

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1TU) before coming to *fold*.

SEE ALSO

expand(1TU)

AUTHOR

Bill Joy

BUGS

If underlining is present it may be messed up by folding.

NAME

forth – forth interpreter

SYNOPSIS

forth

DESCRIPTION

This *forth* is the FIG-FORTH release for PDP-11, except for some minor changes in KEY and XCOUT and a file mechanism.

This *forth* does not know about screens; a file may be loaded by

FILE name LOAD

The maximum source file size is 4k. LOAD commands do not nest.

The way to stop *forth* is BYE.

Extensive documentation is to be found in /usr/man/doc/forth/h.r.

BUGS

In /usr/man/doc/forth/h.r some changes are suggested; I don't know wich part of them has been implemented.

NAME

from – who is my mail from?

SYNTAX

from [–**f** [mailbox]] [–**s** sender]

DESCRIPTION

From prints out the mail header lines in a mailbox file to show you who your mail is from. The *-f* option causes *from* to examine the mail header lines in your mbox (or the specified file). If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/spool/mail/*

SEE ALSO

mail(1)

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. *Getopt* will place – – in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments ($1 $2 . . .) are reset so that each option is preceded by a – and in its own shell argument; each option argument is also in its own shell argument.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** and **b**, and 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; shift;;
        --)         shift;  break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1), getopt(3C).

NAME

graph – draw a graph

SYNOPSIS

graph [option] ...

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1) filters.

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

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

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **–x**).

–b Break (disconnect) the graph after each label in the input.

–c Character string given by next argument is default label for each point.

–g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

–l Next argument is label for graph.

–m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.

–s Save screen, don't erase before plotting.

–x [l]
If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.

–y [l]
Similarly for *y*.

–h Next argument is fraction of space for height.

–w Similarly for width.

–r Next argument is fraction of space to move right before plotting.

–u Similarly to move up before plotting.

–t Transpose horizontal and vertical axes. (Option **–x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **–s** option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1), plot(1)

RESTRICTIONS

Graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

grep, egrep, fgrep – search a file for a pattern

SYNOPSIS

grep [option] ... expression [file] ...

egrep [option] ... [expression] [file] ...

fgrep [option] ... [strings] [file]

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output; unless the **–h** flag is used, the file name is shown if there is more than one input file.

Grep patterns are limited regular expressions in the style of *ed*(1); it uses a compact nondeterministic algorithm. *Egrep* patterns are full regular expressions; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed strings; it is fast and compact.

The following options are recognized.

–v All lines but those matching are printed.

–c Only a count of matching lines is printed.

–l The names of files with matching lines are listed (once) separated by newlines.

–n Each line is preceded by its line number in the file.

–b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.

–s No output is produced, only status.

–h Do not print filename headers with output lines.

–y Lower case letters in the pattern will also match upper case letters in the input (*grep* only).

–e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a **–**.

–f *file* The regular expression (*egrep*) or string list (*fgrep*) is taken from the *file*.

–x (Exact) only lines matched in their entirety are printed (*fgrep* only).

Care should be taken when using the characters $ * [^ | ? ´ " ( ) and \ in the *expression* as they are also meaningful to the Shell. It is safest to enclose the entire *expression* argument in single quotes ´ ´.

Fgrep searches for lines that contain one of the (newline-separated) *strings*.

Egrep accepts extended regular expressions. In the following description 'character' excludes newline:

A \ followed by a single character matches that character.

The character ^ ($) matches the beginning (end) of a line.

A . matches any character.

A single character not otherwise endowed with special meaning matches that character.

A string enclosed in brackets [] matches any single character from the string.

Ranges of ASCII character codes may be abbreviated as in 'a-z0-9'. A] may occur only as the first character of the string. A literal - must be placed where it can't be mistaken as a range indicator.

A regular expression followed by * (+, ?) matches a sequence of 0 or more (1 or more, 0 or 1) matches of the regular expression.

Two regular expressions concatenated match a match of the first followed by a match of the second.

Two regular expressions separated by | or newline match either a match for the first or a match for the second.

A regular expression enclosed in parentheses matches a match for the regular expression.

The order of precedence of operators at the same parenthesis level is [] then *+? then concatenation then | and newline.

SEE ALSO

ed(1), sed(1), sh(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files.

RESTRICTIONS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs.

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

NAME

help – ask for help

SYNOPSIS

help [args]

DESCRIPTION

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

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

type 1 — Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., **ge6**, for message 6 from the *get* command).

type 2 — Does not contain numerics (as a command, such as **get**)

type 3 — Is all numeric (e.g., **212**)

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

When all else fails, try 'help stuck'.

FILES

/usr/lib/help — directory containing files of message text.

DIAGNOSTICS

Use *help*(1) for explanations.

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 superuser can set the hostname by giving an argument; this is usually done in the startup script /etc/rc.

SEE ALSO

ghostname(2)

NAME

hul – filter for highlighting and underlining

SYNOPSIS

hul

DESCRIPTION

Hul is used to filter highlighting (bold) and underlining send to a terminal. It is terminal-independent for it is *termcap(5)*.

The type of terminal is determined by the environnement of the *CSH(1)*. It can be changed by **setenv TERM <terminal>** to fit a certain terminal.

The bold lettering can be generated by NROFF using the fonttype T and/or the macro packages mmb and manb (instead of mm and man).

FILES

/etc/bin/hul	filter
/etc/termcap	terminal capabilities

SEE ALSO

csh(1), qhul(L), termcap(5)

BUGS

Hul has no paranormal sighting to determine the type of terminal used.

AUTHOR

P. Spee

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.

SEE ALSO

logname(1), getuid(2), getgid(2).

NAME

iie – interactive ingres editor

SYNOPSIS

iie databasename

DESCRIPTION

Iie is used to edit databases maintained by ingres (see *ingres(1TU)*). Ingres offers possibilities to make changes in databases. Its syntax is quite painful, however. Therefore *iie* was designed. The reference manual for *iie* can be found in /usr/man/doc/iie/iie.txt. You have to be a valid ingres user if you want to use *iie*.

FILES

/usr/man/doc/iie/*
/usr/thd/iie

SEE ALSO

ingres(1TU)
Handleiding voor het gebruik van de Interactieve Ingres Editor by P. Spee.
/usr/man/doc/iie/*

BUGS

Yes, there are.

AUTHOR

P. Spee
N. Plat

NAME

ingres – relational data base management system

SYNOPSIS

ingres [*flags*] dbname [process_table]

DESCRIPTION

This is the command which is used to invoke . Dbname is the name of an existing data base. The optional flags have the following meanings (a "" means the flag may be stated "+ *x*" to set option *x* or "– *x*" to clear option *x*. "–" alone means that "– *x*" must be stated to get the *x* function):

U Enable/disable direct update of the system relations and secondary indicies. You must have the 000004 bit in the status field of the users file set for this flag to be accepted. This option is provided for system debugging and is strongly discouraged for normal use.

– u *name*
Pretend you are the user with login name *name* (found in the users file). If *name* is of the form *:xx*, *xx* is the two character user code of a user. This may only be used by the DBA for the database or by the superuser.

– c N Set the minimum field width for printing character domains to *N*. The default is 6.

– i lN Set integer output field width to *N*. *l* may be 1, 2, or 4 for i1's, i2's, or i4's repectively.

– f lxM.N
Set floating point output field width to *M* characters with *N* decimal places. *l* may be 4 or 8 to apply to f4's or f8's respectively. *x* may be **e, E, f, F, g, G, n,** or **N** to specify an output format. **E** is exponential form, **F** is floating point form, and **G** and **N** are identical to **F** unless the number is too big to fit in that field, when it is output in **E** format. **G** format guarantees decimal point alignment; **N** does not. The default format for both is **n10.3.**

– v X Set the column seperator for retrieves to the terminal and print commands to be *X*. The default is vertical bar.

– r M Set modify mode on the *retrieve into* command to *M*. *M* may be **isam, cisam, hash, chash, heap, cheap, heapsort,** or **cheapsort,** for ISAM, compressed ISAM, hash table, compressed hash table, heap, compressed heap, sorted heap, or compressed sorted heap. The default is "cheapsort".

– n M Set modify mode on the *index* command to *M*. *M* can take the same values as the **– r** flag above. Default is "isam".

a Set/clear the autoclear option in the terminal monitor. It defaults to set.

b Set/reset batch update. Users must the 000002 bit set in the status field of the users file to clear this flag. This flag is normally set. When clear, queries will run slightly faster, but no recovery can take place. Queries which update a secondary index automatically set this flag for that query only.

d Print/don't print the dayfile. Normally set.

s Print/don't print any of the monitor messages, including prompts. This flags is normally set. If cleared, it also clears the **– d** flag.

w Wait/don't wait for the database. If the **+ w** flag is present,
will wait if certain processes are running (purge,restore, and/or sysmod) on the

given data base. Upon completion of those processes will proceed. If the −w flag is present, a message is returned and execution stopped if the data base is not available. If the w flag is omitted and the data base is unavailable, the error message is returned if is running in foreground (more precisely if the standard input is from a terminal), otherwise the wait option is invoked.

Process_table is the pathname of a file which may be used to specify the run-time configuration of . This feature is intended for use in system maintenance only, and its unenlightened use by the user community is strongly discouraged.

Note: It is possible to run the monitor as a batch-processing interface using the '<', '>' and '' operators of the shell, provided the input file is in proper monitor-format.

EXAMPLE

```
ingres demo
ingres −d demo
ingres −s demo < batchfile
ingres −f4g12.2 −i13 +b −rhash demo
```

FILES

/usr/ingres/files/users − valid users
/usr/ingres/data/base/* − data bases
/usr/ingres/datadir/* − for compatability with previous versions
/usr/ingres/files/proctab6.3 − runtime configuration file

SEE ALSO

iie(1TU)
A Tutorial on Ingres by Robert Epstein
Ingres Reference Manual by K. Youssefi e.o.
Extensive documentation can be found in /usr/ingres/doc/*

DIAGNOSTICS

Too many options to − you have stated too many flags as options.
Bad flag format − you have stated a flag in a format which is not intelligible, or a bad flag entirely.
Too many parameters − you have given a database name, a process table name, and "something else" which doesn't know what to do with.
No database name specified
Improper database name − the database name is not legal.
You may not access database *name* − according to the users file, you do not have permission to enter this database.
You are not authorized to use the *flag* flag − the flag specified requires some special authorization, such as a bit in the users file, which you do not have.
Database *name* does not exist
You are not a valid user − you have not been entered into the users file, which means that you may not use at all.
You may not specify this process table − special authorization is needed to specify process tables.
Database temporarily unavailable − someone else is currently performing some operation on the database which makes it impossible for you to even log in. This condition should disappear shortly.

NAME

iostat – report I/O statistics

SYNOPSIS

iostat [option] ... [drive] ... [interval [count]]

DESCRIPTION

Iostat delves into the system and reports certain statistics kept about input-output activity. Information is kept about up to nine different types of disks (HP, HM, HJ, HK, RP, RL, RK, HS, RA) and about typewriters. The RA disks include the RX50 floppy disk (RX), the RD51 winchester disk (RD), and the RC25 disk.

For each disk drive, I/O completions and number of words transferred are counted; for typewriters collectively, the number of input and output characters are counted. Also, each sixtieth of a second, the state of each disk drive is examined and a tally is made if the disk drive is active. The processor state is also examined, this tally goes into one of four categories, depending on whether the system is executing in user mode, in 'nice' (background) user mode, in system mode, or idle.

The *iostat* reports are for all types of activity, seeks as well as data transfers, on all drives that have had any I/O activity since the system was booted, inactive and nonexistent drives are ignored.

The optional *drive* argument allows the reports to be limited to a specified subset of the available drives. Up to six drive names, of the form; hp0, hm1, rl3, rp4, rd1, ra2, etc., may be specified. Reports will be generated for only those drives which exist and have been active.

The optional *interval* argument causes *iostat* to report once each *interval* seconds. The first report is for all time since a reboot and each subsequent report is for the last interval only.

The optional *count* argument restricts the number of reports. If *count* is given, then *interval* must also be specified.

With no option argument *iostat* reports for each disk the number of transfers per minute, the milliseconds per average seek, and the milliseconds per data transfer exclusive of seek time. It also gives the percentage of time the system has spend in each of the four categories mentioned above.

The following options are available:

–t Report the number of characters of terminal IO per second as well.

–i Report the percentage of time spend in each of the four categories mentioned above, the percentage of time each disk controller was active (seeking or transferring, the percentage of time any disk drive was active, and the percentage of time spent in 'IO wait:' idle, but with a disk drive active.

–s Report the raw timing information for each active disk drive. The information consists of; the disk controller name and drive number, the disk's transfer rate (microseconds/word), the percentage of the total system time that the drive had I/O activity, the number of transfers on that drive, and the number of words transferred by the drive.

–b Report on the usage of I/O buffers. The report gives; the number of buffers in the pool, the number of buffered reads, number of read-ahead blocks, number of buffer cache hits, number of buffered writes, and the number of I/O operations on each buffer starting with the first one.

–d Print the date and time at the head of the report.

−a Print the total time in minutes at the end of the report.

FILES

/dev/mem - system memory
/unix - namelist

RESTRICTIONS

The accuracy of the *iostat* reports is subject to the sixtieth of a second granularity of the system clock.

All disk I/O statistics produced by *iostat* are approximate, the RA disk statistics are more approximate than others.

NAME

join – relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is '–', the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

–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.

–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.

–t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

RESTRICTIONS

With default field separation, the collating sequence is that of *sort – b*; with –t, the sequence is that of a plain sort.

The conventions of *join, sort, comm, uniq, look* and *awk*(1) are wildly incongruous.

NAME

kermit – kermit file transfer

SYNOPSIS

kermit [option ...] [file ...]

DESCRIPTION

Kermit is a file transfer program that allows files to be moved between machines of many different operating systems and architectures. This man page describes version 4C of the program.

Arguments are optional. If *Kermit* is executed without arguments, it will enter command mode. Otherwise, *kermit* will read the arguments off the command line and interpret them.

The following notation is used in command descriptions:

fn A Unix file specification, possibly containing either of the "wildcard" characters '*' or '?' ('*' matches all character strings, '?' matches any single character).

fn1 A Unix file specification which may not contain '*' or '?'.

rfn A remote file specification in the remote system's own syntax, which may denote a single file or a group of files.

rfn1 A remote file specification which should denote only a single file.

n A decimal number between 0 and 94.

c A decimal number between 0 and 127 representing the value of an ASCII character.

cc A decimal number between 0 and 31, or else exactly 127, representing the value of an ASCII control character.

[ ] Any field in square braces is optional.

{*x*,*y*,*z*} Alternatives are listed in curly braces.

Kermit command line options may specify either actions or settings. If *Kermit* is invoked with a command line that specifies no actions, then it will issue a prompt and begin interactive dialog. Action options specify either protocol transactions or terminal connection.

COMMAND LINE OPTIONS

–s *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters, the Unix shell expands it into a list. If *fn* is '-' then *Kermit* sends from standard input, which must come from a file:

```
kermit -s - < foo.bar
```

or a parallel process:

```
ls -l | kermit -s -
```

You cannot use this mechanism to send terminal typein. If you want to send a file whose name is "-" you can precede it with a path name, as in

```
kermit -s ./-
```

–r Receive a file or files. Wait passively for files to arrive.

–k Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

```
kermit -k
```

Displays the incoming files on your screen; to be used only in "local mode" (see below).

```
kermit -k > fn1
```

Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

```
kermit -k | command
```

Pipes the incoming data (single or multiple files) to the indicated command, as in

```
kermit -k | sort > sorted.stuff
```

−a *fn1* If you have specified a file transfer option, you may specify an alternate name for a single file with the **-a** option. For example,

```
kermit -s foo -a bar
```

sends the file foo telling the receiver that its name is bar. If more than one file arrives or is sent, only the first file is affected by the **−a** option:

```
kermit -ra baz
```

stores the first incoming file under the name baz.

−x Begin server operation. May be used in either local or remote mode.

Before proceeding, a few words about remote and local operation are necessary. *Kermit* is "local" if it is running on a PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line — not your job's controlling terminal or console. *Kermit* is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line, connected to your PC or workstation.

If you are running *Kermit* on a PC, it is in local mode by default, with the "back port" designated for file transfer and terminal connection. If you are running *Kermit* on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection. The following command sets *Kermit*'s "mode":

−l *dev* Line — Specify a terminal line to use for file transfer and terminal connection, as in

```
kermit -l /dev/ttyi5
```

When an external line is being used, you might also need some additional options for successful communication with the remote system:

−b *n* Baud — Specify the baud rate for the line given in the **−l** option, as in

```
kermit -l /dev/ttyi5 -b 9600
```

This option should always be included with the **−l** option, since the speed of an external line is not necessarily what you expect.

−p *x* Parity — **e**, **o**, **m**, **s**, **n** (even, odd, mark, space, or none). If parity is other than none, then the 8th-bit prefixing mechanism will be used for transferring 8-bit binary data, provided the opposite *Kermit* agrees. The default parity is none.

−t Specifies half duplex, line turnaround with XON as the handshake character.

The following commands may be used only with a *Kermit* which is local — either by default or else because the **−l** option has been specified.

−g *rfn* Actively request a remote server to send the named file or files; *rfn* is a file

specification in the remote host's own syntax. If *fn* happens to contain any special shell characters, like '*', these must be quoted, as in

kermit -g x*.\?

−f Send a 'finish' command to a remote server.

−c Establish a terminal connection over the specified or default communication line, before any protocol transaction takes place. Get back to the local system by typing the escape character (normally Control-Backslash) followed by the letter 'c'.

−n Like **−c**, but after a protocol transaction takes place; **−c** and **−n** may both be used in the same command. The use of **−n** and **−c** is illustrated below.

On a timesharing system, the **−l** and **−b** options will also have to be included with the **−r**, **−k**, or **−s** options if the other *Kermit* is on a remote system.

If *kermit* is in local mode, the screen (stdout) is continously updated to show the progress of the file transer. A dot is printed for every four data packets, other packets are shown by type (e.g. 'S' for Send-Init), 'T' is printed when there's a timeout, and '%' for each retransmission. In addition, you may type (to stdin) certain "interrupt" commands during file transfer:

Control-F: Interrupt the current File, and go on to the next (if any).

Control-B: Interrupt the entire Batch of files, terminate the transaction.

Control-R: Resend the current packet

Control-A: Display a status report for the current transaction.

These interrupt characters differ from the ones used in other *Kermit* implementations to avoid conflict with Unix shell interrupt characters. With System III and System V implementations of Unix, interrupt commands must be preceeded by the escape character (e.g. control-\).

Several other command-line options are provided:

−i Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used to slightly boost efficiency in Unix-to-Unix transfers of text files by eliminating CRLF/newline conversion.

−w Write-Protect — Avoid filename collisions for incoming files.

−q Quiet — Suppress screen update during file transfer, for instance to allow a file transfer to proceed in the background.

−d Debug — Record debugging information in the file debug.log in the current directory. Use this option if you believe the program is misbehaving, and show the resulting log to your local *Kermit* maintainer.

−h Help — Display a brief synopsis of the command line options.

The command line may contain no more than one protocol action option.

INTERACTIVE OPERATION

Kermit's interactive command prompt is "C-Kermit>". In response to this prompt, you may type any valid command. *Kermit* executes the command and then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally an English verb, such as "send". You may omit trailing characters from any keyword, so long as you specify sufficient characters to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as "send", "receive", "connect") have special non-unique abbreviations ("s" for "send", "r" for "receive", "c" for "connect").

Certain characters have special functions in interactive commands:

? Question mark, typed at any point in a command, will produce a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.

ESC (The Escape or Altmode key) — Request completion of the current keyword or filename, or insertion of a default value. The result will be a beep if the requested operation fails.

DEL (The Delete or Rubout key) — Delete the previous character from the command. You may also use BS (Backspace, Control-H) for this function.

^W (Control-W) — Erase the rightmost word from the command line.

^U (Control-U) — Erase the entire command.

^R (Control-R) — Redisplay the current command.

SP (Space) — Delimits fields (keywords, filenames, numbers) within a command. HT (Horizontal Tab) may also be used for this purpose.

CR (Carriage Return) — Enters the command for execution. **LF** (Linefeed) or **FF** (formfeed) may also be used for this purpose.

**** (Backslash) — Enter any of the above characters into the command, literally. To enter a backslash, type two backslashes in a row (\\). A single backslash immediately preceding a carriage return allows you to continue the command on the next line.

You may type the editing characters (DEL, ^W, etc) repeatedly, to delete all the way back to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If you make any mistakes, you will receive an informative error message and a new prompt — make liberal use of '?' and ESC to feel your way through the commands. One important command is "help" — you should use it the first time you run *Kermit*.

Interactive *Kermit* accepts commands from files as well as from the keyboard. When you enter interactive mode, *Kermit* looks for the file .kermrc in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not Unix command-line format. A "take" command is also provided for use at any time during an interactive session. Command files may be nested to any reasonable depth.

Here is a brief list of *Kermit* interactive commands:

! Execute a Unix shell command.

bye Terminate and log out a remote *Kermit* server.

close Close a log file.

connect Establish a terminal connection to a remote system.

cwd Change Working Directory.

dial	Dial a telephone number.
directory	Display a directory listing.
echo	Display arguments literally.
exit	Exit from the program, closing any open logs.
finish	Instruct a remote *Kermit* server to exit, but not log out.
get	Get files from a remote *Kermit* server.
help	Display a help message for a given command.
log	Open a log file — debugging, packet, session, transaction.
quit	Same as 'exit'.
receive	Passively wait for files to arrive.
remote	Issue file management commands to a remote *Kermit* server.
script	Execute a login script with a remote system.
send	Send files.
server	Begin server operation.
set	Set various parameters.
show	Display values of 'set' parameters.
space	Display current disk space usage.
statistics	Display statistics about most recent transaction.
take	Execute commands from a file.

The 'set' parameters are:

block-check	Level of packet error detection.
delay	How long to wait before sending first packet.
duplex	Specify which side echoes during 'connect'.
escape-character	Character to prefix "escape commands" during 'connect'.
file	Set various file parameters.
flow-control	Communication line full-duplex flow control.
handshake	Communication line half-duplex turnaround character.
line	Communication line device name.
modem-dialer	Type of modem-dialer on communication line.
parity	Communication line character parity.
prompt	Change the *Kermit* program's prompt.
receive	Set various parameters for inbound packets.

send Set various parameters for outbound packets.

speed Communication line speed.

The 'remote' commands are:

cwd Change remote working directory.

delete Delete remote files.

directory Display a listing of remote file names.

help Request help from a remote server.

host Issue a command to the remote host in its own command language.

space Display current disk space usage on remote system.

type Display a remote file on your screen.

who Display who's logged in, or get information about a user.

FILES

$HOME/.kermrc *Kermit* initialization commands
./.kermrc more *Kermit* initialization commands

SEE ALSO

cu(1C), uucp(1C)
Frank da Cruz and Bill Catchings, *Kermit User's Guide*, Columbia University, 6th Edition

DIAGNOSTICS

The diagnostics produced by *Kermit* itself are intended to be self-explanatory.

BUGS

See recent issues of the Info-Kermit digest (on ARPANET or Usenet), or the file ckuker.bwr, for a list of bugs.

NAME

kermit – file transfer, virtual terminal over tty link

SYNOPSIS

kermit csrdilbe [*line*] [*baud rate*] [*escape char*]

DESCRIPTION

Kermit provides reliable file transfer and primitive virtual terminal communication between machines. It has been implemented on many different computers, including microprocessors (see below). The files transferred may be arbitrary ASCII data (7-bit characters) and may be of any length. The file transfer protocol uses small (96 character) checksummed packets, with ACK/NACK responses and timeouts. Kermit currently uses a five second timeout and ten retries.

The new command syntax isn't as user friendly as that of the other implementations, but it is closer to what most Unix users will expect.
Roughly, this is how it works. You can only do one thing with any one invocation of Kermit; either connect, send, or receive. The format is basically like that of the "dump" or "tar" tape utilities, in that the first command line argument is a word made up of letters which specify the desired function and options. This "word" is then followed by the arguments for the options, in the order in which the options appear in the command "word," and then a list of files to be sent, if the send command is being invoked. The commands are:

c	connect
s	send
r	receive

and the options are:

d	debug mode (no argument)
i	image (8-bit) mode; no mapping from LF to CRLF, and all eight bits of each byte are sent or received. if this option is not specified, LF is mapped to CRLF, and only 7 bit bytes are sent or received.
l	line (the next arg is a tty name, e.g. /dev/tty01)
b	baud rate (the next arg is a number, e.g. 1200)
e	escape char (next arg is a decimal number, interpreted as the ascii code for the desired escape character)

The defaults are no debug mode, no external tty line (i.e. "host" mode operation), the system default baud rate, and ^D as the escape character. For example, assume that a user on a micro running Unix wants to transfer files between her machine and a DEC20 over a hardwire line, say, /dev/tty13, at 4800 baud, using ^A (ascii code 1) as her escape character. Note that the order in which arguments are given must match the order in which corresponding letters appear in the first "word":

```
% kermit clbe /dev/tty13 4800 1
Kermit: connected
@
@; now we're on the -20
@kermit
Kermit-20>send *.c
```

```
                                    (user types ^A to get back to Unix)
Kermit: disconnected
%
% kermit rlb /dev/tty13 4800       (no escape character needed here)
Receiving CONNECT.C
Receiving MAIN.C
Receiving SEND.C
Receiving RECEIVE.C
Receiving UTILS.C
OK
%                                   (now, let's send something to the -20)
% kermit clbe /dev/tty13 4800 1
Kermit: connected.

Kermit-20>receive
                                    (user types ^A again to get back)
Kermit: disconnected.
%
% kermit slb /dev/tty13 4800 *.pl
Sending learn.pl
Sending parse.pl
OK
%
```

While the format shown above might seem verbose, it's easy to set up shell scripts or command macros so that, for a given tty line, all one needs to type is "connect", "send file1 file2 ..." or "receive."

"Host" mode operation is much simpler. When connected to a Unix host, nothing about the tty line need be specified, so the format reduces to

```
% kermit r
```

to receive files sent by the "remote" Kermit, and

```
% kermit s file1 file2 file3 ...
```

to send files back to the "remote" system (the "i" option also applies here). Unfortunately, this command syntax doesn't allow the user to own the tty line from the "remote" machine to the "host" continuously, but this probably won't be a problem for most users.

The current version has been used to transfer files between Vaxes running Berkeley 4.1bsd Unix, a Sun workstation running Unisoft Unix (basically Unix Version 7 with Berkeley extensions), and a DEC20, over dialups at 1200 baud, and hardwire lines at 9600 baud, all with no noticable problems. None of the special features of Berkeley Unix are used, so only minimal, straightforward changes should be necessary to get Kermit up and running on other flavors of Unix.

Detail on other implementations and on the protocol is given in the Kermit Users Guide, and the Kermit Protocol Handbook.

FILES

/dev/tty??

SEE ALSO

fx(1), p(1), g(1), tm(1)
Kermit Users Guide, Fourth Edition (4 May 83), Frank da Cruz, Daphne Tzoar, Bill Catchings

Kermit Protocol Manual, Protocol Version 3 (29 April 83), Frank da Cruz, Bill Catchings
Both of the above documents are from the Columbia University Center for Computing Activities, New York, New York, 10027.

NAME

kill – terminate a process with extreme prejudice

SYNOPSIS

kill [–signo] processid ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. If a signal number preceded by '–' is given as first argument, that signal is sent instead of terminate (see *signal*(2)). This will kill processes that do not catch the signal; in particular 'kill –9 ...' is a sure kill.

By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled.

The killed processes must belong to the current user unless he is the superuser. To shut the system down to single user mode use the *operator services* program. Refer to Chapter 5 of the *ULTRIX-11 System Management Guide* for operator services program information.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using *ps*(1).

SEE ALSO

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

NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [–N] [name ...] [tty ...]

DESCRIPTION

Last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *Last* will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

last reboot

will give an indication of mean time between reboot.

Last with no arguments prints a record of all logins and logouts, in reverse order. The –N option limits the report to N lines.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

FILES

/usr/adm/wtmp login data base
/usr/adm/shutdownlog which records shutdowns and reasons for same

SEE ALSO

wtmp(5), ac(8), lastcomm(1)

AUTHOR

Howard Katseff

NAME

ld – loader

SYNOPSIS

ld [option] file ...

DESCRIPTION

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

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named '__.SYMDEF', then it is understood to be a dictionary for the library such as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for **–l**, they should appear before the file names.

–s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).

–u Take the following argument as a symbol and enter it as 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 '/usr/lib/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 *cc*(1) to discard internally generated labels while retaining symbols local to routines.

–r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

–d Force definition of common storage even if the **–r** flag is present.

–n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.

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

−D The next argument is a decimal number that sets the size of the data segment.

−R The next argument gives one or two hexadecimal numbers in the form:
org1/org2
These numbers specify the text origin and the data origin. One of these two numbers may be left out.

−f The rest of this argument is taken to be a filename. This file should contain more arguments for. *ld*

−i The rest of this argument is taken to be an identification that is inserted in the executable. This identification can be found later with aid of *adb(1)*,the debugger.

−S Discard all symbols except globals and locals.

−m Produces a "MAP OUTPUT" i.e. a symbol table.

FILES

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

SEE ALSO

adb(1), as(1), ar(1), cc(1), ranlib(1)

REMARKS

−t Specially for TNO-use;produces files with old TIMO-header.

−e The following argument is taken to be the name of the entry point of the loaded program;location 0 is the default.
THIS OPTION IS CURRENTLY IGNORED.

BUGS

NAME

ld – loader

SYNOPSIS

ld [option] file ...

DESCRIPTION

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

The overlay loader is used for the generation of the overlay text unix kernel and user overlay programs.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named '__.SYMDEF', then it is understood to be a dictionary for the library such as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols '_etext', '_edata' and '_end' ('etext', 'edata' and 'end' in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for **–l**, they should appear before the file names.

–s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).

–u Take the following argument as a symbol and enter it as 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/lib*x*.a', where *x* is a string. If that does not exist, *ld* tries '/usr/lib/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 *cc*(1) to discard internally generated labels while retaining symbols local to routines.

–r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

–d Force definition of common storage even if the **–r** flag is present.

−n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 4K word boundary following the end of the text.

−i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and **−n** is that here the data starts at location 0.

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

−e The following argument is taken to be the name of the entry point of the loaded program; location 0 is the default.

−O This is an overlay file. This option will cause the entire text segment of a running process to be overlaid , retaining the current data segment, when *exec*(2) is called. Shared data must have the same layout as in the program overlaid.

−D The next argument is a decimal number that sets the size of the data segment.

−Z Marks the beginning of an overlay text segment. The object modules listed up to the next **−Z** or **−L** option are placed in the next overlay (numerically).

−L Marks the end of all overlays. Any further routines or libraries go into the base segment.

FILES

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

SEE ALSO

as(1), ar(1), cc(1), f77(1), ranlib(1)

RESTRICTIONS

The **−O** option is untested and probably does not work.

NAME

learn – computer aided instruction about UNIX

SYNOPSIS

learn [–directory] [subject [lesson [speed]]]

DESCRIPTION

Learn gives CAI courses and practice in the use of UNIX. To get started simply type 'learn'. The program will ask questions to find out what you want to do. The questions may be bypassed by naming a *subject,* and the last *lesson* number that *learn* told you in the previous session. You may also include a *speed* number that was given with the lesson number (but without the parentheses that *learn* places around the speed number). If *lesson* is '–', *learn* prompts for each lesson; this is useful for debugging.

The *subjects* presently handled are

editor
eqn
files
macros
morefiles
C

The special command 'bye' terminates a *learn* session.

The – *directory* option allows one to exercise a script in a nonstandard place.

FILES

/usr/lib/learn and all dependent directories and files

RESTRICTIONS

The main strength of *learn,* that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped, but it takes some sophistication to recognize the situation.

NAME

lex − generator of lexical analysis programs

SYNOPSIS

lex [**−tvfn**] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c −ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The following *lex* program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%
[A−Z] putchar(yytext[0]+'a'−'A');
[ ]+$
[ ]+    putchar(' ');
```

The options have the following meanings.

−t Place the result on the standard output instead of in file 'lex.yy.c'.

−v Print a one-line summary of statistics of the generated analyzer.

−n Opposite of **−v**; **−n** is default.

−f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

SEE ALSO

yacc(1)

M. E. Lesk and E. Schmidt, *LEX − Lexical Analyzer Generator*

NAME

link link to a remote computer via a terminal line/modem

SYNOPSIS

[-speed] computername

DESCRIPTION

is a program to enable you to use a Unix terminal as if it was a terminal another computer to which your Unix is connected by modem or terminal line connection. You can cause Unix files to be transmitted (appearing like keyboard input to the remote machine), and the output from the remote machine can be collected in a Unix file. If the remote machine is another Unix system, then you should use the companion program rt(I) (q.v.) to receive and transmit files on the remote Unix. uses the special device table to interpret the argument and associate it with a Unix device (normally a terminal line). It verifies that the line is available, then locks it against use by others, sets the line to transmit raw 8-bit characters, and the line speed (on multiplexer lines) according to the speed item in the entry, unless there is a flag argument (-110, -300, -600, -1200, -2400, -4800 or -9600), which overides the speed setting. While the link program is active, every character received from the line is displayed on the screen, and with two exceptions, every character typed at the terminal is transmitted immediately to the line. The normal Unix terminal input editing conventions do not apply. The first exception is that the Unix line terminator ('\n', or '\r') is mapped to the line termination character given by sixth item in the entry. The second exception is the '!' character. When this occurs as the first character in a line, it introduces a command to the link program. The remainder of the line is interpreted by and is not transmitted. commands are specified by their initial letter, the remainder of the command word is then supplied by and any parameters (normally a file name) are prompted, with the normal input editing conventions. The Commands are:

!a abort current file transmission
!b baud rate change
!c close current receiving file
!e echo terminal input locally (i.e. in Unix)
!f files in use (displays name of receiving file and/or transmitting file, if any)
!g get a text file (e.g. one transmitted by 'rt' on a remote machine) line-by-line, transmitting a prompt for each line.
!h help (displays this summary)
!n no local echo
!q quit the link program
!r receive into file (all displayed characters are written to the specified file until a subsequent 'close')
!s send contents of file (without mappings or synchronisation)
!t transfer text file (e.g. to 'rt' on another Unix) line-by-line, waiting for prompts between lines
!u temporarily return to Unix (until a subsequent ^D). Link remains active.
!v change verbosity (of file transfers in !g and !t operations)
!! transmit a single '!'

Only the !g and !t commands require further explanation. !t is designed to simulate the interactions required to input a text file to the remote machine. After sending each line from the specified Unix file, it waits for the 'prompt string' to be received. The prompt string is the seventh item in the entry mentioned above. When the whole file has been transmitted, the tenth item in the entry is transmitted. !g does the converse operation; a file is received line-by-line. The prompt string is transmitted when each line has been received. When an an end of file string is received, the file is closed. When the remote machine is

another Unix system, the companion program rt should be used to receive and transmit files. The default prompt string for rt is ">". Both !g and !t normally operate silently, i.e. the file is not displayed as it is being transfered. not displayed. A verbose mode of transfer, in which the transfered information is displayed, can be obtained by executing the !v command.

NOTE

In the PDP version of link, '!' is replaced by '~'. Furthermore the ~p command has been added. This sends a break, wich causes the portselector to hang up.

FILES

/etc/spdevices contains an entry giving protocol for each remote system

SEE ALSO

avail (I), lock (I), setup(I), unlock (I), rt(I), wrap(I), spdevices(V), linkgetty(8)

BUGS

NAME

lint − a C program verifier

SYNOPSIS

lint [**−abchnpuvx**] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as '-lm', in the style of *ld*(1).

Any number of the options in the following list may be used. The **−D**, **−U**, and **−I** options of *cc*(1) are also recognized as separate arguments.

p Attempt to check portability to the *IBM* and *GCOS* dialects of C.

h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.

b Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)

v Suppress complaints about unused arguments in functions.

x Report variables referred to by extern declarations, but never used.

a Report assignments of long values to int variables.

c Complain about casts which have questionable portability.

u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).

n Do not check compatibility against the standard library.

Exit(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code.

/*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.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the −v option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

FILES

/usr/lib/lint[12] programs
/usr/lib/llib-lc declarations for standard functions
/usr/lib/llib-port declarations for portable functions

SEE ALSO

cc(1)
S. C. Johnson, *Lint, a C Program Checker*

NAME

ln – make a link

SYNOPSIS

ln [**-f**] name1 [name2]

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There is no way to distinguish a link to a file from its original directory entry; any changes in the file are effective independently of the name by which the file is known.

The **-f** option allows the superuser to link to a directory.

Ln creates a link to an existing file *name1*. If *name2* is given, the link has that name; otherwise it is placed in the current directory and its name is the last component of *name1*.

It is forbidden to link across file systems.

SEE ALSO

rm(1)

NAME

lock lock a special device

SYNOPSIS

lock name

DESCRIPTION

lock is a function associated with the *link* system for communication with satellite computers.

lock name locks the special device named against use by the link command until a subsequent:

unlock name To define a new special device, make an entry in the file /etc/spdevices mentioned below.

FILES

All of the information to enable this command to work for any device or remote satellite is contained in the /etc/spdevices file (documented in *spdevices(5)*).

SEE ALSO

link(1), lock(1), unlock(1), avail(1), setup(1), spdevices(5)

BUGS

NAME

login – sign on

SYNOPSIS

login [username]

DESCRIPTION

The *login* command is used when a user initially signs on, or it may be used at any time to change from one user to another. The latter case is the one summarized above and described here. See 'How to Get Started' for how to dial up initially.

If *login* is invoked without an argument, it asks for a user name, and, if appropriate, a password. Echoing is turned off (if possible) during the typing of the password, so it will not appear on the written record of the session.

After a successful login, accounting files are updated and the user is informed of the existence of *.mail* and message-of-the-day files. *Login* initializes the user and group IDs and the working directory, and sets the HOME, PATH, TERM, SHELL and USER environment variables. It then executes a command interpreter (usually *sh*(1) or *csh(1)*) according to specifications found in a password file. If the command interpreter is */bin/csh*, the new line discipline is entered, otherwise the terminal stop characters are set to be undefined (see *stty(1)*). Argument 0 of the command interpreter starts with a '–'.

Login is recognized by *sh*(1) and *csh*(1) and executed directly (without forking).

FILES

/etc/utmp	accounting
/usr/adm/wtmp	accounting
/usr/mail/*	mail
/etc/motd	message-of-the-day
/etc/passwd	password file

SEE ALSO

init(8), newgrp(1), getty(8), mail(1), passwd(1), passwd(5)

DIAGNOSTICS

'Login incorrect,' if the name or the password is bad.
'No Shell', 'cannot open password file', 'no directory': consult a programming counselor.

NAME

look − find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort*(1):

d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence **−df.**

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see *ar*(1)) *files*. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1).

This brash one-liner intends to build a new library from existing '.o' files.

```
ar cr library `lorder *.o | tsort`
```

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

RESTRICTIONS

The names of object files, in and out of libraries, must end with '.o'; nonsense results otherwise.

NAME

lpq – show lineprinter queue

SYNOPSIS

lpq [option] [job_numbers ...] [user_names ...]

DESCRIPTION

Lpq displays the *files* that are queued for printing on a line printer. Any job numbers or user names entered with the command will limit the output to jobs with the appropriate job numbers or that have been printed by the users whose names were given. The following *option*s are available:

–l Display the queue with a long format.

–Pprinter
Display the queue for printer *printer*.

FILES

/usr/spool/lpd/lock	
/usr/spool/lpd/cf*	data file
/usr/spool/lpd/df*	daemon control file
/usr/spool/lpd/tf*	temporary version of control file

SEE ALSO

lpq(1), lprm(1), printcap(5), lpd(8), ulf(8)

NAME

lpr – line printer spooler

SYNOPSIS

lpr [option] ... [file] ...

DESCRIPTION

Lpr causes the *file*s to be queued for printing on a line printer. If no files are named, the standard input is read. The following *option*s are available:

–r Remove the file when it has been queued.

–c Copy the file to insulate against changes that may happen before printing.

–m Report by *mail*(1) when printing is complete.

–Jname Print *name* instead of file name on header page.

–p Pipe each file through *pr* before printing it.

–hname Use *name* instead of file name on *pr's* header.

–Cname Use *name* instead of *hostname* on header.

–inn Indent output *nn* spaces (8 if just -i).

–#nn Print *nn* copies of each file.

–Pprinter
Route output to alternate printer *printer*.

FILES

/usr/spool/lpd/lock	
/usr/spool/lpd/cf*	data file
/usr/spool/lpd/df*	daemon control file
/usr/spool/lpd/tf*	temporary version of control file
/usr/adm/lpacct	accounting records

SEE ALSO

lpq(1), lprm(1), printcap(5), lpd(8), ulf(8)

NAME

lprm – delete jobs from the lineprinter queue

SYNOPSIS

lprm [option] [job_numbers ...] [user_names ...]

DESCRIPTION

Lprm deletes the *files* from the print queue. *Lprm* will attempt to delete members whose job numbers or user names have been entered with the command. No one except the superuser may remove another user's files. The following *options* are available:

– Delete all jobs owned by the issuer of the command.

– Pprinter
Delete jobs from the queue of the alternate printer *printer.*

FILES

/usr/spool/lpd/lock	
/usr/spool/lpd/cf*	data file
/usr/spool/lpd/df*	daemon control file
/usr/spool/lpd/tf*	temporary version of control file

SEE ALSO

lpq(1), lpr(1), printcap(5), lpd(8), ulf(8)

NAME

ls − list contents of directory

SYNOPSIS

ls [**−abcdfgilmqrstux1CFR**] name ...

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The format chosen depends on whether the output is going to a teletype, and may also be controlled by option flags. The default format for a teletype is to list the contents of directories in multi-column format, with the entries sorted down the columns. (Files which are not the contents of a directory being interpreted are always sorted across the page rather than down the page in columns. This is because the individual file names may be arbitrarily long.) If the standard output is not a teletype, the default format is to list one entry per line. Finally, there is a stream output format in which files are listed across the page, separated by ',' characters. The **−m** flag enables this format; when invoked as *l* this format is also used.

There are an unbelievable number of options:

−l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.

−t Sort by time modified (latest first) instead of by name, as is normal.

−a List all entries; usually '.' and '..' are suppressed.

−s Give size in blocks, including indirect blocks, for each entry.

−d If argument is a directory, list only its name, not its contents (mostly used with **−l** to get status on directory).

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−u Use time of last access instead of last modification for sorting (**−t**) or printing (**−l**).

−c Use time of file creation for sorting or printing.

−i Print i-number in first column of the report for each file listed.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

−g Give group ID instead of owner ID in long listing.

−m force stream output format

−1 force one entry per line output format, e.g. to a teletype

−C force multi-column output, e.g. to a file or a pipe

−q force printing of non-graphic characters in file names as the character '?'; this normally happens only if the output device is a teletype

−b force printing of non-graphic characters to be in the \ddd notation, in octal.

−x force columnar printing to be sorted across rather than down the page; this is the default if the last character of the name the program is invoked with is an 'x'.

−F cause directories to be marked with a trailing '/' and executable files to be marked with a trailing '*'; this is the default if the last character of the name the program is invoked with is a 'f'.

−R recursively list subdirectories encountered.

The mode printed under the −l option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;
b if the entry is a block-type special file;
c if the entry is a character-type special file;
m if the entry is a multiplexor-type character special file;
− if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
− if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise the user-execute permission character is given as **s** if the file has set-user-ID mode.

The last character of the mode (normally 'x' or '−') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user ID's for 'ls −l'.
/etc/group to get group ID's for 'ls −g'.

RESTRICTIONS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide.

The option setting based on whether the output is a teletype is undesirable as "ls −s" is much different than "ls −s | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

Column widths choices are poor for terminals which can tab.

NAME

m4 – macro processor

SYNOPSIS

m4 [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no arguments, or if an argument is '–', the standard input is read. The processed text is written on the standard output.

Macro calls have the form

name(arg1,arg2, . . . , argn)

The '(' must immediately follow the name of the macro. If a defined macro name is not followed by a '(', it is deemed to have no arguments. Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore '_', where the first character is not a digit.

Left and right single quotes (`´) are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define — The second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string.

undefine — removes the definition of the macro named in its argument.

ifdef — If the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX versions of *m4*.

changequote — Change quote characters to the first and second arguments. *Changequote* without arguments restores the original values (i.e., `´).

divert — *M4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert — causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum — returns the value of the current output stream.

dnl — reads and discards characters up to and including the next newline.

ifelse — has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7.

Otherwise, the value is either the fourth string, or, if it is not present, null.

incr returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

eval evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation); relationals; parentheses.

len returns the number of characters in its argument.

index returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

substr returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include returns the contents of the file named in the argument.

sinclude is identical to *include*, except that it says nothing if the file is inaccessible.

syscmd executes the UNIX command given in the first argument. No value is returned.

maketemp fills in a string of XXXXX in its argument with the current process id.

errprint prints its argument on the diagnostic output file.

dumpdef prints current names and definitions, for the named items, or for all if no arguments are given.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The M4 Macro Processor*

NAME

m11 – Macro-11 assembler for UNIX

SYNOPSIS

m11 [option1 option2 ...] file1 file2 ... filen

DESCRIPTION

M11 assembles the concatenation of the specified files (*file1*, etc.) and terminates when an '.end' statement is encountered. The resulting object file is usually named *filen.obj* (see below). If a file argument, *filei* does not contain a '.' in its name, the file *filei.m11* will be sought before *filei* itself.

Options, if desired, may appear anywhere in the command, and are chosen from the following list. All options are interpreted before any files are read.

– ls Produce an assembly listing and place in *filen*.lst

– lt Produce an assembly listing on the standard output.

– fl If coupled with the – *ls* or – *lt* directives, makes the listing have a shortened format. It is shorthand for – *nl:seq:loc:bin:bex:me:meb:ttm:toc:sym*.

– uc Simulate an initial *.dsabl lc* directive. Force all characters in macro definitions to be upper case. This flag makes lower-case handling in *m11* compatible with the DEC Macro-11 assemblers.

– um Force all characters in macro definitions to be upper case. This flag makes lower-case handling in this release of *m11* compatible with previous versions of *m11*.

– de Make all option choices needed to make assembly mimic DEC Macro-11. Implies *(inter alia)* the – *uc* flag. This includes the Johns Hopkins *asm* assembler.

– ha Make all option choices needed to make assembly mimic earlier (Harvard) releases of m11. This implies the – *um* flag. Default .psect and .csect attributes are set up in the funny Harvard way.

– mx Produce a listing of the source program as it appears *after* macro expansion. Macro calls, conditional directives and so on appear in the listing as comments. Listing appears on standard output. No machine code is generated or listed. This option is meant to correspond to the – E or – P options of the C compiler *cc(1)*.

– my Like – *mx*, except that macro calls and conditional directives do not show up in the listing.

– 10 Generate an error whenever op codes not in the PDP-11 'standard instruction set' are encountered. Programmers writing for a PDP 11/10 can catch instructions illegal for that machine by using this argument.

– dp:args
The default attributes for a *.psect* or unnamed *.csect* are redefined, using the colon-separated list *args* of valid .psect attributes.

– da:args
The default attributes for an *.asect* are redefined.

– dp:c The default attributes for a named *.csect* are redefined.

– li:arglist
Simulate an initial *.list arglist* directive. All *.list* and *.nlist* directives in the program text which attempt to change the settings established with the – *li* flag will be

ignored.

– nl:arglist
Like – *li:arglist*, but for the *.nlist* directive.

– en:arglist
Similarly, for the *.enabl* directive.

– ds:arglist
Similarly, but for the *.dsabl* directive.

– cr:arglist
Produces a cross-reference listing. If the – *ls* option is also included, the cross-reference listing will follow the assembly listing in *filen*.lst. References which are tagged with the symbol # are definitions. References tagged with * are destructive references: the value of the symbol or variable in question is changed. *Arglist* consists of colon-separated keywords from the following set. The keywords may be prefix abbreviated:

sym All user-defined symbols are indexed.

mac All macro names are indexed.

per All uses of permanent symbols – op codes, directives, etc – are indexed.

pse All psect names are indexed. For compatibility with the RT-11 *CREF* program, the argument *cse* is synonymous with *pse*.

err All errors are indexed.

reg All register uses are indexed.

If no *arglist* is specified the default *sym:mac:err* is used. In the listing page and line numbers for uses of symbols are followed by a # sign if the symbol is defined and by a * sign if the symbol is modified.

– lp Same as – ls, but also spools *filen*.lst for printing upon completion.

– no No object file is produced. This is useful for syntax checking or list producing.

– xs:n Allots nK words of extra space for symbol table and macro storage. NOTE: This option is currently inoperative: *m11* automatically allots core for its tables as needed.

– xx Debug flag: generate all kinds of weird hack flack.

– ns No symbol table is included in the object file (thus ddt knows of no symbols from this assembly).

– sx Make the symbol table contain 'local symbols' as well as ordinary symbols.

– u Treat form feed characters as spaces. This will make *m11*'s idea of line numbers coincide with the UNIX text editors. Macro-11 statements containing imbedded form feed characters will be parsed differently when the – *u* flag is in effect.

– na:file
Override the convention of using last name as file name. Instead, use names *file.obj* and *file.lst* for object and listing files.

NOTES

This implementation of Macro-11 is a distant hand-me down from an old DEC Macro-11 modified at Harvard University in the early 1970's. It is grubby with smudges by Brent Byer, F. J. Howard, Bob Bowering, and Jim Reeds. It does not implement keyword arguments such as are described in section 7.3.6 of the DEC manual. The .enabl abs option

does not work. Listing control is by default .list ttm. Unlike earlier editions of m11 at UCB and at Harvard, it does treat immediate constants of floating point operations correctly: see the last paragraph of section 6.4.2 on the middle of page 6-27 of the DEC manual. M11 has several directives not described in the DEC manual. See the *New UCB M11 Manual*. The default attributes for *.psects* are different from those described in the DEC manual, but may be changed by the *– dp* flag. The *.enabl glb* feature is implemented: undefined symbols are taken as undefined global externals.

FILES

/usr/lib/sysmac.sml	system macro library (for .mcall directive)
filen.xrf	intermediate cross-reference temporary file
lpr(1)	spooler
/usr/ucb/macxrf	cross-reference post-processor

SEE ALSO

PDP-11 MACRO-11 Language Reference Manual , Digital Equipment Corp. Order No. AA-5075A-TC, August 1977.

New UCB M11 Manual , notional document by Jim Reeds.

NAME

mail, rmail − send mail to users or read mail

SYNOPSIS

mail [−**rpq**] [−**f** file]

mail persons

rmail 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:

−**r** causes messages to be printed in first-in, first-out order.
−**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.
−**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 >. 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 **dead.letter** will be saved to allow editing and resending.

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**.

The *mailfile* may be manipulated in two ways to alter the function 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.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in he is informed of the presence of mail, if any.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*	incoming mail for user *; *mailfile*
$HOME/mbox	saved mail
$MAIL	*mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race 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

mail – send and receive mail

SYNTAX

mail [**–v**] [**–i**] [**–n**] [**–s** subject] [user ...]
mail [**–v**] [**–i**] [**–n**] **–f** [name]
mail [**–v**] [**–i**] [**–n**] **–u** user

INTRODUCTION

Mail is a intelligent mail processing system, which has a command syntax reminiscent of *ed* with lines replaced by messages.

The **–v** flag puts mail into verbose mode; the details of delivery are displayed on the users terminal. The **–i** flag causes tty interrupt signals to be ignored. This is particularly useful when using *mail* on noisy phone lines. The **–n** flag inhibits the reading of /usr/lib/Mail.rc.

Sending mail. To send a message to one or more other people, *mail* can be invoked with arguments which are the names of people to send to. You are then expected to type in your message, followed by an EOT (control–D) at the beginning of a line. A subject may be specified on the command line by using the **–s** flag. (Only the first argument after the **–s** flag is used as a subject; be careful to quote subjects containing spaces.) The section below, labeled *Replying to or originating mail,* describes some features of *mail* available to help you compose your letter.

Reading mail. In normal usage *mail* is given no arguments and checks your mail out of the post office, then prints out a one line header of each message there. The current message is initially the first message (numbered 1) and can be printed using the **print** command (which can be abbreviated **p**). You can move among the messages much as you move between lines in *ed,* either with the commands '+' and '–' moving backwards and forwards or with simple numbers.

Disposing of mail. After examining a message you can **delete** (**d**) the message or **reply** (**r**) to it. Deletion causes the *mail* program to forget about the message. This is not irreversible; the message can be **undeleted** (**u**) by giving its number, or the *mail* session can be aborted by giving the **exit** (**x**) command. Deleted messages will, however, usually disappear never to be seen again.

Specifying messages. Commands such as **print** and **delete** can be given a list of message numbers as arguments to apply to a number of messages at once. Thus "delete 1 2" deletes messages 1 and 2, while "delete 1–5" deletes messages 1 through 5. The special name "*" addresses all messages, and "$" addresses the last message; thus the command **top** which prints the first few lines of a message could be used in "top *" to print the first few lines of all messages.

Replying to or originating mail. You can use the **reply** command to set up a response to a message, sending it back to the person who it was from. Text you then type in, up to an end-of-file, defines the contents of the message. While you are composing a message, *mail* treats lines beginning with the character '~' specially. For instance, typing "~m" (alone on a line) will place a copy of the current message into the response right shifting it by a tabstop. Other escapes will set up subject fields, add and delete recipients to the message and allow you to escape to an editor to revise the message or to a shell to run some commands. (These options are given in the summary below.)

Ending a mail processing session. You can end a *mail* session with the **quit** (**q**) command. Messages which have been examined go to your *mbox* file unless they have been deleted in which case they are discarded. Unexamined messages go back to the post office. The **–f** option causes *mail* to read in the contents of your *mbox* (or the specified file) for processing; when you **quit**, *mail* writes undeleted messages back to this file. The **–u** flag is a short way

of doing "mail −f /usr/spool/mail/user".

Personal and systemwide distribution lists. It is also possible to create a personal distribution lists so that, for instance, you can send mail to "cohorts" and have it go to a group of people. Such lists can be defined by placing a line like

alias cohorts bill ozalp jkf mark kridle@ucbcory

in the file .mailrc in your home directory. The current list of such aliases can be displayed with the **alias** (**a**) command in *mail.* System wide distribution lists can be created by editing /usr/lib/aliases, see *aliases*(5) and *sendmail*(8); these are kept in a different syntax. In mail you send, personal aliases will be expanded in mail sent to others so that they will be able to **reply** to the recipients. System wide *aliases* are not expanded when the mail is sent, but any reply returned to the machine will have the system wide alias expanded as all mail goes through *sendmail.*

Network mail (ARPA, UUCP, Berknet) See *mailaddr(7)* for a description of network addresses.

Mail has a number of options which can be set in the *.mailrc* file to alter its behavior; thus "set askcc" enables the "askcc" feature. (These options are summarized below.)

SUMMARY

(Adapted from the 'Mail Reference Manual')

Each command is typed on a line by itself, and may take arguments following the command word. The command need not be typed in its entirety − the first command which matches the typed prefix is used. For commands which take message lists as arguments, if no message list is given, then the next message forward which satisfies the command's requirements is used. If there are no messages forward of the current message, the search proceeds backwards, and if there are no good messages at all, *mail* types "No applicable messages" and aborts the command.

− Goes to the previous message and prints it out. If given a numeric argument *n*, goes to the *n*-th previous message and prints it.

? Prints a brief summary of commands.

! Executes the UNIX shell command which follows.

Print (**P**) Like **print** but also prints out ignored header fields. See also **print** and **ignore.**

Reply (R) Reply to originator. Does not reply to other recipients of the original message.

Type (T) Identical to the **Print** command.

alias (**a**) With no arguments, prints out all currently-defined aliases. With one argument, prints out that alias. With more than one argument, creates an new or changes an on old alias. These aliases are in effect for the current mail session only.

alternates (**alt**) The **alternates** command is useful if you have accounts on several machines. It can be used to inform *mail* that the listed addresses are really you. When you **reply** to messages, *mail* will not send a copy of the message to any of the addresses listed on the *alternates* list. If the **alternates** command is given with no argument, the current set of alternate names is displayed.

chdir (**ch**) Changes the user's working directory to that specified. If no directory is given, then changes to the user's login directory.

copy (**co**) The **copy** command does the same thing that **save** does, except that it does

not mark the messages it is used on for deletion when you quit.

delete (**d**) Takes a list of messages as argument and marks them all as deleted. Deleted messages will not be saved in *mbox*, nor will they be available for most other commands.

dp (also **dt**) Deletes the current message and prints the next message. If there is no next message, *mail* says "at EOF."

edit (**e**) Takes a list of messages and points the text editor at each one in turn. On return from the editor, the message is read back in.

exit (**ex** or **x**) Effects an immediate return to the Shell without modifying the user's system mailbox, his *mbox* file, or his edit file in −**f**.

file (**fi**) The same as **folder**.

folders List the names of the folders in your folder directory.

folder (**fo**) The **folder** command switches to a new mail file or folder. With no arguments, it tells you which file you are currently reading. If you give it an argument, it will write out changes (such as deletions) you have made in the current file and read in the new file. Some special conventions are recognized for the name. # means the previous file, % means your system mailbox, %user means user's system mailbox, & means your ~/mbox file, and +folder means a file in your folder directory.

from (**f**) Takes a list of messages and prints their message headers in the order that they appear in the mail directory, not in the order given in the list.

headers (**h**) Lists the current range of headers, which is an 18 message group. If a "+" argument is given, then the next 18 message group is printed, and if a "−" argument is given, the previous 18 message group is printed.

help A synonym for ?

hold (**ho**, also **preserve**) Takes a message list and marks each message therein to be saved in the user's system mailbox instead of in *mbox*. Does not override the **delete** command.

ignore Add the list of header fields named to the *ignored list*. Header fields in the ignore list are not printed on your terminal when you print a message. This command is very handy for suppression of certain machine-generated header fields. The **Type** and **Print** commands can be used to print a message in its entirety, including ignored fields. If **ignore** is executed with no arguments, it lists the current set of ignored fields.

mail (**m**) Takes as argument login names and distribution group names and sends mail to those people.

mbox Indicate that a list of messages be sent to *mbox* in your home directory when you quit. This is the default action for messages if you do *not* have the *hold* option set.

next (**n** like + or CR) Goes to the next message in sequence and types it. With an argument list, types the next matching message.

preserve (**pre**) A synonym for **hold**.

print (**p**) Takes a message list and types out each message on the user's terminal.

quit (**q**) Terminates the session, saving all undeleted, unsaved messages in the user's *mbox* file in his login directory, preserving all messages marked with

hold or **preserve** or never referenced in his system mailbox, and removing all other messages from his system mailbox. If new mail has arrived during the session, the message "You have new mail" is given. If given while editing a mailbox file with the **-f** flag, then the edit file is rewritten. A return to the Shell is effected, unless the rewrite of edit file fails, in which case the user can escape with the **exit** command.

reply (**r**) Takes a message list and sends mail to the sender and all recipients of the specified message. The default message must not be deleted.

respond A synonym for **reply**.

save (**s**) Takes a message list and a filename and appends each message to the end of the file. The messages are saved in the order in which they appear in the mail directory, not in that given in the message list. The filename in quotes, followed by the line count and character count is echoed on the user's terminal.

set (**se**) With no arguments, prints all variable values. Otherwise, sets option. Arguments are of the form "option=value" or "option."

shell (**sh**) Invokes an interactive version of the shell.

size Takes a message list and prints out the size (in characters) of each message. The size of the messages are printed in the order that they appear in the mail directory, not in that given in the list.

source (**so**) The **source** command reads *mail* commands from a file.

top Takes a message list and prints the top few lines of each. The number of lines printed is controlled by the variable **toplines** and defaults to five.

type (**t**) A synonym for **print**.

unalias Takes a list of names defined by **alias** commands and discards the remembered groups of users. The group names no longer have any significance.

undelete (**u**) Takes a message list and marks each one as *not* being deleted.

unset Takes a list of option names and discards their remembered values; the inverse of **set**.

visual (**v**) Takes a message list and invokes the display editor on each message.

write (**w**) A synonym for **save**.

xit (**x**) A synonym for **exit**.

z *Mail* presents message headers in windowfuls as described under the **headers** command. You can move *mail*'s attention forward to the next window with the **z** command. Also, you can move to the previous window by using **z-**.

Here is a summary of the tilde escapes, which are used when composing messages to perform special functions. Tilde escapes are only recognized at the beginning of lines. The name "tilde escape" is somewhat of a misnomer since the actual escape character can be set by the option **escape**.

~!command Execute the indicated shell command, then return to the message.

~c name ... Add the given names to the list of carbon copy recipients.

~d Read the file "dead.letter" from your home directory into the message.

~e Invoke the text editor on the message collected so far. After the editing session is finished, you may continue appending text to the message.

~f messages Read the named messages into the message being sent. If no messages are specified, read in the current message.

~h Edit the message header fields by typing each one in turn and allowing the user to append text to the end or modify the field by using the current terminal erase and kill characters.

~m messages
Read the named messages into the message being sent, shifted right one tab. If no messages are specified, read the current message.

~p Print out the message collected so far, prefaced by the message header fields.

~q Abort the message being sent, copying the message to "dead.letter" in your home directory if **save** is set.

~r filename Read the named file into the message.

~s string Cause the named string to become the current subject field.

~t name ... Add the given names to the direct recipient list.

~v Invoke an alternate editor (defined by the VISUAL option) on the message collected so far. Usually, the alternate editor will be a screen editor. After you quit the editor, you may resume appending text to the end of your message.

~w filename Write the message onto the named file.

~|command Pipe the message through the command as a filter. If the command gives no output or terminates abnormally, retain the original text of the message. The command *fmt*(1) is often used as *command* to rejustify the message.

~~string Insert the string of text in the message prefaced by a single ~. If you have changed the escape character, then you should double that character in order to send it.

Options are controlled via the **set** and **unset** commands. Options may be either binary, in which case it is only significant to see whether they are set or not, or string, in which case the actual value is of interest. The binary options include the following:

append Causes messages saved in *mbox* to be appended to the end rather than prepended. (This is set in /usr/lib/Mail.rc on version 7 systems.)

ask Causes *mail* to prompt you for the subject of each message you send. If you respond with simply a newline, no subject field will be sent.

askcc Causes you to be prompted for additional carbon copy recipients at the end of each message. Responding with a newline indicates your satisfaction with the current list.

autoprint Causes the **delete** command to behave like **dp** – thus, after deleting a message, the next one will be typed automatically.

debug Setting the binary option *debug* is the same as specifying **–d** on the command line and causes *mail* to output all sorts of information useful for debugging *mail*.

dot The binary option *dot* causes *mail* to interpret a period alone on a line as the terminator of a message you are sending.

hold This option is used to hold messages in the system mailbox by default.

ignore Causes interrupt signals from your terminal to be ignored and echoed as

@'s.

ignoreeof An option related to *dot* is *ignoreeof* which makes *mail* refuse to accept a control-d as the end of a message. *Ignoreeof* also applies to *mail* command mode.

msgprompt When sending mail, prompts you for the message text and indicates how to terminate the message.

metoo Usually, when a group is expanded that contains the sender, the sender is removed from the expansion. Setting this option causes the sender to be included in the group.

nosave Normally, when you abort a message with two RUBOUT, *mail* A copies the partial letter to the file "dead.letter" in your home directory. Setting the binary option *nosave* prevents this.

quiet Suppresses the printing of the version when first invoked.

verbose Setting the option *verbose* is the same as using the −v flag on the command line. When mail runs in verbose mode, the actual delivery of messages is displayed on he users terminal.

The following options have string values:

EDITOR Pathname of the text editor to use in the **edit** command and ~e escape. If not defined, then a default editor is used.

SHELL Pathname of the shell to use in the **!** command and the ~! escape. A default shell is used if this option is not defined.

VISUAL Pathname of the text editor to use in the **visual** command and ~v escape.

crt The valued option *crt* is used as a threshold to determine how long a message must be before *more* is used to read it.

escape If defined, the first character of this option gives the character to use in the place of ~ to denote escapes.

folder The name of the directory to use for storing folders of messages. If this name begins with a '/', *mail* considers it to be an absolute pathname; otherwise, the folder directory is found relative to your home directory.

record If defined, gives the pathname of the file used to record all outgoing mail. If not defined, then outgoing mail is not so saved.

toplines If defined, gives the number of lines of a message to be printed out with the **top** command; normally, the first five lines are printed.

FILES

/usr/spool/mail/*	post office
~/mbox	your old mail
~/.mailrc	file giving initial mail commands
/tmp/R#	temporary for editor escape
/usr/lib/Mail.help*	help files
/usr/lib/Mail.rc	system initialization file
Message*	temporary for editing messages

SEE ALSO

binmail(1), from(1)
'The Mail Reference Manual'

NAME

make − maintain program groups

SYNOPSIS

make [−**f** makefile] [option] ... file ...

DESCRIPTION

Make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no −**f** option is present, 'makefile' and 'Makefile' are tried in order. If *makefile* is '−', the standard input is taken. More than one −**f** option may appear

Make updates a target if it depends on prerequisite files that have been modified since the target was last modified, or if the target does not exist.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated list of targets, then a colon, then a list of prerequisite files. Text following a semicolon, and all following lines that begin with a tab, are shell commands to be executed to update the target.

Sharp and newline surround comments.

The following makefile says that 'pgm' depends on two files 'a.o' and 'b.o', and that they in turn depend on '.c' files and a common file 'incl'.

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
a.o: incl a.c
        cc −c a.c
b.o: incl b.c
        cc −c b.c
```

Makefile entries of the form

```
string1 = string2
```

are macro definitions. Subsequent appearances of *$(string1)* are replaced by *string2*. If *string1* is a single character, the parentheses are optional.

Make infers prerequisites for files for which *makefile* gives no construction commands. For example, a '.c' file may be inferred as prerequisite for a '.o' file and be compiled to produce the '.o' file. Thus the preceding example can be done more briefly:

```
pgm: a.o b.o
        cc a.o b.o −lm −o pgm
a.o b.o: incl
```

Prerequisites are inferred according to selected suffixes listed as the 'prerequisites' for the special name '.SUFFIXES'; multiple lists accumulate; an empty list clears what came before. Order is significant; the first possible name for which both a file and a rule as described in the next paragraph exist is inferred. The default list is

```
.SUFFIXES: .out .o .c .e .r .f .y .l .s
```

The rule to create a file with suffix *s2* that depends on a similarly named file with suffix *s1* is specified as an entry for the 'target' *s1s2*. In such an entry, the special macro $* stands for the target name with suffix deleted, $@ for the full target name, $< for the complete list of prerequisites, and $? for the list of prerequisites that are out of date. For example, a rule for making optimized '.o' files from '.c' files is

```
.c.o: ; cc −c −O −o $@ $*.c
```

Certain macros are used by the default inference rules to communicate optional arguments to any resulting compilations. In particular, 'CFLAGS' is used for *cc* and *f77*(1) options, 'LFLAGS' and 'YFLAGS' for *lex* and *yacc*(1) options.

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the special target '.SILENT' is in *makefile*, or the first character of the command is '@'.

Commands returning nonzero status (see *intro*(1)) cause *make* to terminate unless the special target '.IGNORE' is in *makefile* or the command begins with <tab><hyphen>.

Interrupt and quit cause the target to be deleted unless the target depends on the special name '.PRECIOUS'.

Other options:

−i Equivalent to the special entry '.IGNORE:'.

−k When a command returns nonzero status, abandon work on the current entry, but continue on branches that do not depend on the current entry.

−n Trace and print, but do not execute the commands needed to update the targets.

−t Touch, i.e. update the modified date of targets, without executing any commands.

−r Equivalent to an initial special entry '.SUFFIXES:' with no list.

−s Equivalent to the special entry '.SILENT:'.

FILES

makefile, Makefile

SEE ALSO

sh(1), touch(1)
S. I. Feldman *Make − A Program for Maintaining Computer Programs*

RESTRICTIONS

Some commands return nonzero status inappropriately. Use **−i** to overcome the difficulty. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across newlines in *make*.

NAME

man − find manual information by keywords; print out the manual

SYNOPSIS

man −**k** keyword ...
man −**f** file ...
man [−] [−**t**] [section] title ...

DESCRIPTION

Man is a program which gives information from the programmers manual. It can be asked form one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option −**k** and a set of keywords, *man* prints out a one line synopsis of each manual sections whose listing in the table of contents contains that keyword.

When given the option −**f** and a list of file names, *man* attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither −**k** nor −**f** is specified, *man* formats a specified set of manual pages. If a section specifier is given *man* looks in the that section of the manual for the given *titles*. *Section* is an arabic section number, i.e. 3, which may be followed by a single letter classifier, i.e. 1g indicating a graphics program in section 1. If *section* is omitted, *man* searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

If the standard output is a teletype, or if the flag − is given, then *man* pipes its output through *cat*(1) with the option −**s** to crush out useless blank lines, *ul*(1) to create proper underlines for different terminals, and through *more*(1) to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

The −**t** flag causes *man* to arrange for the specified section to be *troff'ed* to a suitable raster output device; see *vtroff*(1).

FILES

/usr/man/man?/*
/usr/man/cat?/*

SEE ALSO

more(1), ul(1), whereis(1), catman(8)

BUGS

The manual is supposed to be reproducible either on the phototypesetter or on a typewriter. However, on a typewriter some information is necessarily lost.

NAME

man – print sections of this manual

SYNOPSIS

man [option ...] [chapter] title ... .

DESCRIPTION

Man locates and prints the section of this manual named *title* in the specified *chapter*. (In this context, the word 'page' is often used as a synonym for 'section'.) The *title* is entered in lower case. The *chapter* number does not need a letter suffix. If no *chapter* is specified, the whole manual is searched for *title* and all occurrences of it are printed.

Options and their meanings are:

–t Phototypeset the section using *troff*(1).

–n Print the section on the standard output using *nroff*(1).

–k Display the output on a Tektronix 4014 terminal using *troff*(1) and *tc*(1).

–e Appended or prefixed to any of the above causes the manual section to be preprocessed by *neqn* or *eqn*(1); **–e** alone means **–te**.

–w Print the path names of the manual sections, but do not print the sections themselves.

(default)
Copy an already formatted manual section to the terminal, or, if none is available, act as **–n**. It may be necessary to use a filter to adapt the output to the particular terminal's characteristics.

Further *options*, e.g. to specify the kind of terminal you have, are passed on to *troff*(1) or *nroff*. *Options* and *chapter* may be changed before each *title*.

For example:

man man

would reproduce this section, as well as any other sections named *man* that may exist in other chapters of the manual, e.g. *man*(7).

FILES

/usr/man/man?/*
/usr/man/cat?/*
/tmp/Man*

SEE ALSO

nroff(1), eqn(1), tc(1), man(7)

RESTRICTIONS

The manual is supposed to be reproducible either on a phototypesetter or on a terminal. However, on a terminal some information is necessarily lost.

NAME

memstat – print memory usage map

SYNOPSIS

memstat [ifn] [interval] [corefile] [namelist]

DESCRIPTION

The *memstat* program prints a map showing the utilization of all of memory. The optional argument 'i' specifies that the printout should be repeated every *interval* seconds. The 'f' option causes *corefile* to be used as memory instead of /dev/mem. The 'n' option declares that the system namelist should be obtained from *namelist* instead of /unix. Single or multiple options may be specified. The arguments interval, corefile, and namelist must occur in the same order as the options [ifn].

The following headings appear on the *memstat* printout:

Addr — The octal address range of the section of memory being mapped.

Command — The name of the entity occupying a section of memory. This may be the operating system, a process, a shared text segment, or free memory.

Pid — The process ID number of the process named in command.

Size — The total size in bytes of non-shared text processes or the size of the data segment of shared text processes.

Ublock — The address and size of the U block (per process data) for the process.

Text — Size and address of the processes text segment.

Data — Size and address of the processes data segment.

Stack — Size and address of the processes stack.

FILES

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

SEE ALSO

Refer to the ULTRIX-11 System Management Guide, Section 9.5.5 for an example of a memstat printout.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*
/dev

SEE ALSO

write(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777. Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

rm(1)

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

NAME

mkstr – create an error message file by massaging C source

SYNTAX

mkstr [–] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be:

mkstr pistrings xx *.c

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file *mkstr* keys on the string 'error("' in the input stream. Each time it occurs, the C string starting at the '"' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains an *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char    efilname[] =  "/usr/lib/pistrings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[BUFSIZ];

        if (efil < 0) {
                efil = open(efilname, FATT_RDONLY);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, FSEEK_ABSOLUTE)
          || read(efil, buf, sizeof buf) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

The optional "–" causes the error messages to be placed at the end of the specified message file for recompiling part of a large program upon which *mkstr* has already been run.

SEE ALSO

xstr(1), lseek(2)

RESTRICTIONS

All the arguments except the name of the file to be processed are unnecessary.

AUTHORS

Bill Joy and Charles Haley

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [-cdflsu] [-n] [+ *linenumber*] [+ */pattern*] [name ...]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

– *n* An integer which is the size (in lines) of the window which *more* will use instead of the default.

–c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

–d *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

–f This causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen postions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

–l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

–s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

–u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *– u* option suppresses this processing.

+ *linenumber*
Start up at *linenumber.*

+ */pattern*
Start up two lines before the line containing the regular expression *pattern.*

If the program is invoked as *page,* then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more* , including invocations by programs such as *man* and *msgs* , to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
: display *i* more lines, (or another screenful if no argument is given)

^D
: display 11 more lines (a 'scroll'). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control-D)

*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

q or Q
: Exit from *more*.

=
: Display the current line number.

v
: Start up the editor *vi* at the current line.

h
: Help command; give a description of all the *more* commands.

i/expr
: search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n
: search for the *i*-th occurrence of the last regular expression entered.

'
: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command
: invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n
: skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p
: skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a

file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat,* except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

AUTHOR

Eric Shienbrood, minor revisions by John Foderaro and Geoffrey Peck

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

NAME

mv – move or rename files and directories

SYNOPSIS

mv file1 file2

mv file ... directory

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, *mv* exits.

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

Mv refuses to move a file onto itself.

SEE ALSO

cp(1), chmod(2)

RESTRICTIONS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Mv should take **–f** flag, like *rm*, to suppress the question if the target exists and is not writable.

NAME

newgrp – log in to a new group

SYNOPSIS

newgrp group

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

A password is demanded if the group has a password and the user himself does not.

When most users log in, they are members of the group named 'other.' *Newgrp* is known to the shell, which executes it directly without a fork.

FILES

/etc/group, /etc/passwd

SEE ALSO

login(1), group(5)

NAME

nice, nohup – run a command at low priority

SYNOPSIS

nice [– *number*] command [arguments]

nohup command [arguments]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The superuser may run commands with priority higher than normal by using a negative priority, e.g. '– –10'.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. *Nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

nice(2)

DIAGNOSTICS

Nice returns the exit status of the subject command.

NAME

nm – print name list

SYNOPSIS

nm [**–gnopru**] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in 'a.out' are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters U (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **B** (bss segment symbol), or **C** (common symbol). If the symbol is local (non-external) the type letter is in lower case. If the 'a.out' file is an overlay text executable (0430 or 0431), all text segment symbols are followed by their overlay text segment number. The output is sorted alphabetically.

Options are:

–g Print only global (external) symbols.

–n Sort numerically rather than alphabetically.

–o Prepend file or archive element name to each output line rather than only once.

–p Don't sort; print in symbol-table order.

–r Sort in reverse order.

–u Print only undefined symbols.

SEE ALSO

ar(1), ar(5), a.out(5)

NAME

nroff, troff – text formatting and typesetting

SYNOPSIS

nroff [option] ... [file] ...

troff [option] ... [file] ...

DESCRIPTION

Troff formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; *nroff* for typewriter-like devices. Their capabilities are described in the *Nroff/Troff user's manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input. The options, which may appear in any order so long as they appear before the files, are:

–o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N*–*M* means pages *N* through *M*; an initial –*N* means from the beginning to page *N*; and a final *N*– means from *N* to the end.

–n*N* Number first generated page *N*.

–s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. *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.

–m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files.*

–r*aN* Set register *a* (one-character) to *N*.

–i Read standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **rd** request.

Nroff only

–T*name* Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).

–e Produce equally-spaced words in adjusted lines, using full terminal resolution.

–h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

Troff only

–t Direct output to the standard output instead of the phototypesetter.

–f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

–w Wait until phototypesetter is available, if currently busy.

–b Report whether the phototypesetter is busy or available. No text processing is done.

–a Send a printable ASCII approximation of the results to the standard output.

–pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

-g Prepare output for a GCOS phototypesetter and direct it to the standard output (see *gcat*(1)).

If the file */usr/adm/tracct* is writable, *troff* keeps phototypesetter accounting records there. The integrity of that file may be secured by making *troff* a 'set user-id' program.

FILES

/usr/lib/suftab	suffix hyphenation tables
/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for *nroff*
/usr/lib/font/*	font width tables for *troff*
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

SEE ALSO

J. F. Ossanna, *Nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*
croff(1)
eqn(1), tbl(1)
col(1), tk(1) (*nroff* only)
tc(1), gcat(1) (*troff* only)

NAME

od – octal dump

SYNOPSIS

od [**–bcdox**] [file] [[+]offset[.][**b**]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **–o** is default. The meanings of the format argument characters are:

b Interpret bytes in octal.

c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.

d Interpret words in decimal.

o Interpret words in octal.

x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If '.' is appended, the offset is interpreted in decimal. If 'b' is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded '+'.

Dumping continues until end-of-file.

SEE ALSO

adb(1)

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the superuser may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3)
Robert Morris and Ken Thompson, *Password Security: A Case History*

NAME

paste − merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste −**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* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if − is used in place of a file name.

The meanings of the options are:

−d Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use "" − *d*"\\\\").

−s Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

− May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls **|** paste −d" " −	list directory in one column
ls **|** paste − − − −	list directory in four columns
paste −s −d"\t\n" file	combine pairs of lines into lines

SEE ALSO

grep(1), cut(1),
pr(1): **pr −t −m...** works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **−s** option, no more than 12 input files may be specified.

NAME

plot – graphics filters

SYNOPSIS

plot [–Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot*(5)) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter $TERM (see *environ*(5)) is used. Known *terminals* are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of *plot* places a scan-converted image in '/usr/tmp/raster' and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster

SEE ALSO

plot(3), plot(5)

RESTRICTIONS

There is no lockout protection for /usr/tmp/raster.

NAME

pop2 interactive pop2 language

SYNOPSIS

[file ...]

DESCRIPTION

is an interactive object-oriented language developed by artificial intelligence researchers. The Unix version supports all of the Standard POP-2 set forth in *Programming in POP-2* by R M Burstall, J S Collins, and R J Popplestone. Many enhancements, including those implemented in the new DEC-10 "WonderPOP" system, are implemented. The system occupies 8K of pure code, and a minimum of 2K of data. The user's workspace may grow to 22K words of data.

FILES

Documentation, sources, library and error files can be found in /usr/lib/pop

AUTHOR

William Clocksin
Dept. of Artificial Intelligence, University of Edinburgh

NAME

postnews – submit news articles

SYNOPSIS

postnews [*article*]

DESCRIPTION

Postnews is a program that calls *inews*(8) to submit news articles to USENET. The commands should be self-explanatory, however you may type "?" to most prompts to get a list of the possible options (except for the "Keywords" of the article, etc). It will prompt the user for the title of the article (which should be a phrase suggesting the subject, so that persons reading the news can tell if they are interested in the article), for the newsgroup, and for the distribution.

The distribution is typically a geographic region or corporate region. Typing "?" will get you a list of the possible distributions. You should use the minimum distribution that will serve your purpose for posting the article. For example, if you are selling your car in New Jersey, it is doubtful that someone in California (or Europe) would be willing to buy it. If you don't restrict the distribution to your local area, you will cause this article to be transmitted unnecessarily around the world. Currently, with a distribution of **world**, the article will be seen in the United States, Canada, Europe, Japan, Korea and other places. A distribution header will, if given, be included in the headers of the article, affecting where the article is distributed.

After entering the title, newsgroup, and distribution, the user will be placed in an editor. If **EDITOR** is set in the environment, that editor will be used. Otherwise, *postnews* defaults to *vi*(1).

An initial set of headers containing the subject and newsgroups will be placed in the editor, followed by a blank line. The article should be appended to the buffer, after the blank line. The initial headers can be changed, or additional headers added, while in the editor, if desired.

After you have finished typing in your article, you have the option of sending it, listing it, quitting without sending it, editing the file again, or saving it in a file without sending it.

For posting news from a program, see *inews*(8).

If the file */usr/lib/news/recording* is present, it is taken as a list of "recordings" to be shown to users posting news. (This is named after the recording you hear when you dial "information" in some parts of the U.S., asking you to stop and think if you really want do do this, but not actually preventing you.) The recording file contains lines of the form:

newsgroup-specifier TAB filename

for example:

```
comp.all          comp.recording
local.all,!local.test local.recording
```

Any user posting an article to a newsgroup matching the pattern on the left will be shown the contents of the file on the right. The file is found in the **LIBDIR** directory (often */usr/lib/news*). The user is then told to hit DEL to abort or RETURN to proceed. The intent of this feature is to help companies keep proprietary information from accidently leaking out.

FILES

/usr/lib/news/active	List of known newsgroups and highest local article numbers in each.
/usr/lib/news/distributions	Suggested distribution code names

/usr/lib/news/newsgroups Descriptons of newsgroups

SEE ALSO

inews(8), readnews(1), vi(1), news(5), expire(8)

NAME

pr – print file

SYNOPSIS

pr [option] ... [file] ...

DESCRIPTION

Pr produces a printed listing of one or more *files*. The output is separated into pages headed by a date, the name of the file or a specified header, and the page number. If there are no file arguments, *pr* prints its standard input.

Options apply to all following files but may be reset between files:

– *n* Produce *n*-column output.

+ *n* Begin printing with page *n*.

–h Take the next argument as a page header.

–w*n* For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.

–l*n* Take the length of the page to be *n* lines instead of the default 66.

–t Do not print the 5-line header or the 5-line trailer normally supplied for each page.

–s*c* Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.

–m Print all *files* simultaneously, each in one column,.

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

FILES

/dev/tty? to suspend messages.

SEE ALSO

cat(1)

DIAGNOSTICS

There are no diagnostics when *pr* is printing on a terminal.

NAME

prep − prepare text for statistical processing

SYNOPSIS

prep [−**dio**] file ...

DESCRIPTION

Prep reads each *file* in sequence and writes it on the standard output, one 'word' to a line. A word is a string of alphabetic characters and imbedded apostrophes, delimited by space or punctuation. Hyphenated words are broken apart; hyphens at the end of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

−**d** Print the word number (in the input stream) with each word.

−**i** Take the next *file* as an 'ignore' file. These words will not appear in the output. (They will be counted, for purposes of the −**d** count.)

−**o** Take the next *file* as an 'only' file. Only these words will appear in the output. (All other words will also be counted for the −**d** count.)

−**p** Include punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the −**d** count.

Ignore and only files contain words, one per line.

SEE ALSO

deroff(1)

NAME

printenv – print out the environment

SYNOPSIS

printenv [name]

DESCRIPTION

Printenv prints out the values of the variables in the environment. If a *name* is specified, only its value is printed.

If a *name* is specified and it is not defined in the environment, *printenv* returns exit status 1, else it returns status 0.

SEE ALSO

csh(1), sh(1), environ(5)

NAME

prof − display profile data

SYNOPSIS

prof [−**a**] [−**l**] [file]

DESCRIPTION

Prof interprets the file *mon.out* produced by the *monitor(3)* subroutine. Under default modes, the symbol table in the named object file (*a.out* default) is read and correlated with the *mon.out* profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the −**a** option is used, all symbols are reported rather than just external symbols. If the −**l** option is used, the output is listed by symbol value rather than decreasing percentage.

In order for the number of calls to a routine to be tallied, the −**p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the *mon.out* file to be produced automatically.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

monitor(3), profil(2), cc(1)

RESTRICTIONS

Beware of quantization errors.

NAME

prof − display profile data

SYNOPSIS

prof [−**v**] [−**a**] [−**l**] [− *low* [− *high*]] [file]

DESCRIPTION

Prof interprets the file *mon.out* produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file (*a.out* default) is read and correlated with the *mon.out* profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the −**a** option is used, all symbols are reported rather than just external symbols. If the −**l** option is used, the output is listed by symbol value rather than decreasing percentage.

If the −**v** option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *plot*(1) filters. The numbers *low* and *high*, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the −**p** option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the *mon.out* file to be produced automatically.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

monitor(3), profil(2), cc(1), plot(1)

RESTRICTIONS

Beware of quantization errors.

NAME

prolog – interactive prolog language

SYNOPSIS

prolog

DESCRIPTION

There is no genuine manual page for prolog. Extensive information can be found in:
W.F. Clocksin and C.S. Mellish: The Unix Prolog System. This document can be obtained by the command:
roff /usr/src/thd/prolog/doc/doc.

FILES

/usr/src/thd/prolog/doc/doc

NAME

prs – print an SCCS file

SYNOPSIS

prs [–**d**[dataspec]] [–**r**[SID]] [–**e**] [–**l**] [–**a**] files

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(5)) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

–d[*dataspec*] Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

–r[*SID*] Used to specify the *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.

–e Requests information for all deltas created *earlier* than and including the delta designated via the **–r** keyletter.

–l Requests information for all deltas created *later* than and including the delta designated via the **–r** keyletter.

–a Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the **–a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(5)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec.*

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec.* The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n.

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D* or *R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th::Tm::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes* or *no*	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes* or *no*	S
:BF:	Branch flag	"	*yes* or *no*	S
:J:	Joint edit flag	"	*yes* or *no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes* or *no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y: :M: :I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

prs −d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

prs −d"Newest delta for pgm :M:: :I: Created :D: By :P:" −r s.file

may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case:*

prs s.file

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the 'D' type. The only keyletter argument allowed to be used with the *special case* is the −**a** keyletter.

FILES

/tmp/pr?????

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – process status

SYNOPSIS

ps [**alxvt#**] [namelist] [corefile]

DESCRIPTION

Ps prints certain indicia about active processes. The **a** option asks for information about all processes with terminals (ordinarily only one's own processes are displayed); **x** asks even about processes with no terminal; **l** asks for a long listing. The short listing contains the process ID, tty letter, the cumulative execution time of the process and an approximation to the command line. If **v** is given and **l** is not present, the sums of the child process's system and user times are printed following the cumulative execution time. The **t** option limits printouts to those processes associated with tty #. Specifying **?** as the tty number to the t option will limit printouts to those processes not associated with a tty.

The long listing is columnar and contains

F Flags associated with the process. 001: in core; 002: system process; 004: locked in core (e.g. for physical I/O); 010: being swapped; 020: being traced by another process; 040: another tracing flag; 100: user settable lock in core; 200: detached inheritaed by init; 400: using new signal mechanism.

S The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.

UID The user ID of the process owner.

PID The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.

PPID The process ID of the parent process.

CPU Processor utilization for scheduling.

PRI The priority of the process; high numbers mean low priority.

NICE Used in priority computation.

ADDR
If the process is resident, the memory address in octal expressed as 64 byte clicks. *ADDR* is left shifted six places to obtain the physical memory address. If the process is swapped out, the disk address in decimal, that is, the block number relative to the start of the swap area, where the image of the process resides.

SZ The size in 512 byte blocks of the core image of the process.

WCHAN
The event for which the process is waiting or sleeping; if blank, the process is running.

TTY The controlling tty for the process.

TIME The cumulative execution time for the process.

The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. *Ps* makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

The optional third argument specifies a corefile to be used in place of */dev/mem* . This is used for postmortem system debugging. If a second argument is given, it is taken to be the file containing the system's namelist.

FILES

/unix	default system namelist
/dev/mem	default core memory file
/usr/crash/core	alternate core file for crash dump analysis
/dev	searched to find swap device and tty names

SEE ALSO

kill(1)
ULTRIX-11 System Management Guide, Section 9.5.6

RESTRICTIONS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.
Some data printed for defunct processes is irrelevant

NAME

ptx − permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o only
Use as keywords only the words given in the *only* file.

−i ignore
Do not use as keywords any words given in the *ignore* file. If the *-i* and *-o* options are missing, use /usr/lib/eign as the *ignore* file.

−b break
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/bin/sort
/usr/lib/eign

RESTRICTIONS

Line length counts do not account for overstriking or proportional spacing.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

cd(1)

NAME

qhul – bold filter for the qume printer

SYNOPSIS

qhul

DESCRIPTION

Qhul is used to filter bold printing. The resulting bold printing is done by generating code neccessary to print a character, move one step, print the same character and move nine steps.

The bold lettering can be generated by NROFF using the fonttype T and/or the macro packages mmb and manb (instead of mm and man).

FILES

/usr/thd/qhul filter

SEE ALSO

hul(1TU), qume(1TU)

BUGS

Possibly.

AUTHOR

P. Spee

NAME

qume – print file on daisywheel printer

SYNOPSIS

qume [-fN] [-lN] [-sN] [-pN] filename

DESCRIPTION

The qume is a daisywheel printer in room 0.101, controlled by a dedicated terminal, the Eurocom. This Eurocom is connected by the portselector to a number of UNIX systems, and can be used as an ordinary terminal.

Normally the Eurocom should show a menu that allows you to set certain printer parameters or to become a UNIX terminal. If the power is off, this menu can be generated by the following key sequence (no returns needed):

<reset>
E01
GC000

While in terminal mode, this menu may be entered with ^E.

Typing 't' and a few returns will connect you to the portselector. Now type the machine name for the system required, followed by a return. Valid names are:

73
70
gmx
vax

A linefeed indicates contact has been established, and you may log in. Once logged onto UNIX, you can give the *qume* command. This command will print a file on the qume printer.

options are:

–fN Start printing at page N (default: 1).

–lN The number of lines per page is N (66).

–sN Stop printing every N pages (1).

–pN Print every line N times (1).

Qume may be interrupted with ^C.

SEE ALSO

qhul(1TU)

NAME

quot − summarize file system ownership

SYNOPSIS

quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, a default name is assumed. The default name is obtained from the file system information table (/etc/fstab) by searching for the file system associated with the /usr directory. The following options are available:

−n Cause the pipeline **ncheck filesystem | sort +0n | quot −n filesystem** to produce a list of all files and their owners.

−c Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

−f Print count of number of files as well as space owned by each user.

FILES

/dev/r??? - filesystem
/etc/fstab - default file system name
/etc/passwd - to get user names

SEE ALSO

ls(1), du(1)

DIAGNOSTICS

If no filesystem is given and no filesystem is mounted on the */usr* directory, an error message is printed.

RESTRICTIONS

Holes in files are counted as if they actually occupied space.

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named **__.SYMDEF** to the beginning of the archive. It uses *ar*(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

SEE ALSO

ld(1), ar(1)

RESTRICTIONS

Because generation of a library by *ar* and randomization by *ranlib* are separate, phase errors are possible. The loader *ld* warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

NAME

ratfor – rational Fortran dialect

SYNOPSIS

ratfor [option ...] [filename ...]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
switch (integer value) {
        case integer:   statement
        ...
        [ default: ]    statement
}
```

loops:

```
while (condition) statement
for (expression; condition; expression) statement
do limits statement
repeat statement [ until (condition) ]
break [n]
next [n]
```

and some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation

comments:

this is a comment

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return (expression)

returns expression to caller from function

define: define name replacement

include:

include filename

The option **–h** causes quoted strings to be turned into 27H constructs. **–C** copies comments to the output, and attempts to format it neatly. Normally, continuation lines are marked with a & in column 1; the option **–6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77*(1).

SEE ALSO

f77(1)

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

readnews − read news articles

SYNOPSIS

readnews [−**a** *date*] [−**n** *newsgroups*] [−**t** *titles*] [−**leprxhfuM**] [−**c** [*mailer*]]

readnews −**s**

readnews −**K**

DESCRIPTION

Readnews without argument prints unread articles. There are several interfaces available other than the default:

Flag	Interface
−**M**	An interface to *mailx*(1) or *Mail(1)*.
−**c**	A *binmail*(1)−like interface.
−**c** *"mailer"*	All selected articles written to a temporary file. Then the mailer is invoked. The name of the temporary file is referenced with a "%". Thus, "mail −f %" will invoke mail on a temporary file consisting of all selected messages.
−**p**	All selected articles are sent to the standard output. No questions asked.
−**l**	Only the titles output. The *.newsrc* file will not be updated.
−**e**	Like −**l** but also updates the *.newsrc* file.

The −**r** flag causes the articles to be printed in reverse order. The −**f** flag prevents any followup articles from being printed. The −**h** flag causes articles to be printed in a less verbose format, and is intended for terminals running at 300 baud. The −**u** flag causes the *.newsrc* file to be updated every 5 minutes, in case of an unreliable system. (Note that if the *.newsrc* file is updated, the **x** command will not restore it to its original contents.)

The following flags determine the selection of articles.

−**n** *newsgroups*	Select all articles that belong to *newsgroups*.
−**t** *titles*	Select all articles whose titles contain one of the strings specified by *titles*.
−**a** [*date*]	Select all articles that were posted past the given *date* (in *getdate*(3) format).
−**x**	Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

Readnews maintains a *.newsrc* file in the user's home directory that specifies all news articles already read. It is updated at the end of each reading session in which the −**x** or −**l** options weren't specified. If the environment variable **NEWSRC** is present, it should be the path name of a file to be used in place of *.newsrc*.

If the user wishes, an options line may be placed in the *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of standard options just as they would be typed on the command line. Such a list may include: the −**n** flag along with a newsgroup list; a favorite interface; and/or the −**r** or −**t** flag. Continuation lines are specified by following lines beginning with a space or tab character. Similarly, options can be specified in the **NEWSOPTS** environment parameter. Where conflicts exist, option on the command line take precedence, followed by the *.newsrc* **options** line, and lastly the **NEWSOPTS** parameter.

You can use the −s flag to print the newsgroup subscription list.

If you haven't read news in a while (or if you have never read news!) you can do *readnews* −*K* to Kill (mark as read) all of the articles in the groups to which you are subscribed.

When the user uses the reply command of the default or *binmail*(1) interfaces, the environment parameter **MAILER** will be used to determine which mailer to use. The default is *mail*(1).

The user may specify a particular paging progam for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with a "%", as in the −c option. If no "%" is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value. By default, the pager is *cat*(1).

COMMANDS

This section lists the commands you can type to the default and *binmail* interface prompts. The default interface will suggest some common commands in brackets. Just hitting return is the same as typing the first command. For example, "[ynq]" means that the commands "y" (yes), "n" (no), and "q" (quit) are common responses, and that "y" is the default.

Command — Meaning

− Go back to last article. This is a toggle, typing it twice returns you to the original article.

Report the name and size of the newsgroup.

! Shell escape.

< message ID>
Look for a particular article. (See *Standard for Interchange of Usenet Messages* for a description of message ID's).

b Back. Back up one article.

c Cancel the article. Only the author or the super user can do this.

d Read a digest. Breaks up a digest into separate articles and permits you to read and reply to each piece.

D Decrypt. Invokes a Caesar decoding program on the body of the message. This is used to decrypt rotated jokes posted to **rec.humor**. Such jokes are usually obscene or otherwise offensive to some groups of people, and so are rotated to avoid accidental decryption by people who would be offended. The title of the joke should indicate the nature of the problem, enabling people to decide whether to decrypt it or not.

An explicit *number* rotation (usually 13) may be given to force a particular shift.

e Erase. Forget that this article was read.

f [*title*] Submit a follow up article. Normally you should leave off the title, since the system will generate one for you. You will be placed in your **EDITOR** to compose the text of the followup.

fd Followup directly, without edited headers. This is like **f**, but the headers of the article are not included in the editor buffer.

h Header. Print a more verbose header.

H Print a very verbose header, containing all known information about the article.

K Kill. Mark all remaining articles in this newsgroup as read and skip to the next newsgroup.

n No. Goes on to next article without printing current one. In the *binmail* interface, this means "go on to the next article", which will have the same effect as **y** or just hitting return.

N [*newsgroup*]
Next Newsgroup. Go to the next newsgroup or named newsgroup.

p Print. Reprint previous article.

P Previous Newsgroup. Go back to previous newsgroup.

q Quit. The *.newsrc* file will be updated if **−l** or **−x** were not on the command line.

r Reply. Reply to article's author via *mail*. You are placed in your **EDITOR** (by default *vi*(1)) with a header specifying "To", "Subject", and "References" lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article.

rd Reply directly. You are placed in **MAILER** (*mail* by default) in reply to the author. Type the text of the reply and then control-D.

s [*file*] Save. The article is appended to the named file. The default is *Articles*. If the first character of the file name is "|", the rest of the file name is taken as the name of a program, which is executed with the text of the article as standard input. If the first character of the file name is "/", it is taken as a full path name of a file. If **NEWSBOX** (in the environment) is set to a full path name, and the file contains no "/", the file is saved in **NEWSBOX**. Otherwise, it is saved relative to **HOME**.

U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.

v Print the current version of the news software.

w Same as **s**.

x Exit. Like quit except that *.newsrc* is not updated.

X *system*
Transmit article to the named system.

y Yes. Prints current article and goes on to next.

number
Go to *number*.

+ [*n*] Skip *n* articles. The articles skipped are recorded as "unread" and will be offered to you again the next time you read news.

The commands **c, f, fd, r, rd, e, h, H,** and **s** can be followed by **−**'s to refer to the previous article. Thus, when replying to an article using the default interface, you should normally type **r−** (or **re-**) since by the time you enter a command, you are being offered the next article.

EXAMPLES

readnews Read all unread articles using the default interface. The *.newsrc* file is updated at the end of the session.

readnews −c "ed %" −l
Invoke the *ed*(1) text editor on a file containing the titles of all unread articles. The *.newsrc* file is **not** updated at the end of the session.

readnews −n all !talk −M −r
Read all unread articles except articles whose newsgroups begin with **talk.** via *mailx* in reverse order. The *.newsrc* file is updated at the end of the session.

readnews −p −n all −a last thursday
Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

readnews −K
Discard all unread news. This is useful after returning from a long trip.

ENVIRONMENT VARIABLES

EDITOR
Editor invoked by **f** command. (Default is */usr/ucb/vi.*)

MAILER
Mailing program invoked by the **r** command. (Default is */bin/mail.*)

NAME
Your full name used in header of articles posted by you. (Default is the comments field of your id in */etc/passwd.*)

NEWSBOX
File or directory where articles saved with the **s** command are stored. (Default is same as **HOME.**)

NEWSOPTS
Options for *readnews.*

ORGANIZATION
Full name of this site used header of articles posted by you.

PAGER
Paging program invoked by articles with more than 16 lines. (Default is */usr/ucb/more.*)

SHELL
The shell invoked by the **!** command. (Default is */bin/sh.*)

FILES

/usr/spool/news/*newsgroup*/*number*	News articles
/usr/lib/news/active	Active newsgroups and numbers of articles
/usr/lib/news/help	Help file for default interface
~/.newsrc	Options and list of previously read articles

SEE ALSO

binmail(1), checknews(1), inews(8), mail(1), mailx(1), news(5), newsrc(5) postnews(1), vnews(1), getdate(3), news(5), newsrc(5), expire(8), recnews(8), sendnews(8), uurec(8)
How to Read the Network News by Mark Horton.
Standard for Interchange of Usenet Messages by Mark Horton.

AUTHORS

Matt Glickman
Mark Horton
Stephen Daniel
Tom R. Truscott

NAME

refer, lookbib – find and insert literature references in documents

SYNOPSIS

refer [option] ...

lookbib [file] ...

DESCRIPTION

Lookbib accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

Refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between .[and .] command lines, which are assumed to contain keywords as for *lookbib,* and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference; by default the references are indicated by numbers.

The following options are available:

–a*r* Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted all author names are reversed.

–b Bare mode: do not put any flags in text (neither numbers nor labels).

–c*string*
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *string.*

–e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to the same source.

–k*x* Instead of numbering references, use labels as specified in a reference data line beginning *%x;* by default *x* is **L.**

–l*m,n*
Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *,n* is omitted the entire name or date respectively is used.

–p Take the next argument as a file of references to be searched. The default file is searched last.

–n Do not search the default file.

–s*keys*
Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies **–e.** The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with + taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort,

for example, on all authors and then title use **-sA+T**.

To use your own references, put them in the format described in *pubindex*(1) They can be searched more rapidly by running *pubindex*(1) on them before using *refer;* failure to index results in a linear search.

When *refer* is used with *eqn, neqn* or *tbl, refer* should be first, to minimize the volume of data passed through pipes.

FILES

/usr/dict/papers directory of default publication lists and indexes
/usr/lib/refer directory of programs

SEE ALSO

NAME

renice – oppress running processes

SYNOPSIS

renice newnice pid1 [pid2 ... pidn]

DESCRIPTION

Renice will assign *pid1* , ..., *pidn* a 'niceness' of *newnice*. Positive niceness means a lower scheduling priority. Negative niceness means a high priority. The superuser may renice any process to any value. Other users may renice their own processes to any value greater than the current value.

SEE ALSO

nice(1), nice(2), renice(2)

NAME

report – report generator for Ingres

SYNOPSIS

report formatfile [optional arguments ...]

DESCRIPTION

The *report* program extracts information from an Ingres database and outputs it, using a specified format. All necessary information is contained in the format file, which has the following layout:

B:<database name>
R:<assigned relation number><relation name>
F:<assigned field number>:<corresponding relation number>: <type>:<field name>:<field length>
W:<selection specification>
#
<optional header format>
#
<text specifying format>
#
<optional footer format>
#
<optional one line filler to footer>

Note: the pound symbols may be omitted, if no footer or filler is used. The <type> indicator can be either i for integer or s for string. Both <selection specification> and <text> can contain the following macros:

%<number> field selection
$<number> argument selection

DIAGNOSTICS

"no database specified"
"illegal arg count"
"argument not specified"
"can't open format file"
"relation number too large"
"relation already defined"
"relation not defined"
"fieldnumber too large"
"field not defined"
"illegal specifier"

SEE ALSO

ingres(1), iie(1TU)

AUTHOR

P. Spee

NAME

rev – reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rm, rmdir – remove (unlink) files

SYNOPSIS

rm [**–fri**] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked when the **–f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **–r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **–i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **–r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2)

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file '..' merely to avoid the antisocial consequences of inadvertently doing something like 'rm –r .*'.

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel –r*SID* files

DESCRIPTION

Rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get*(1)) exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

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

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide.* Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sa, accton – system accounting

SYNOPSIS

sa [**–abcjlnrstuv**] [file]

/etc/accton [file]

DESCRIPTION

With an argument naming an existing *file*, *accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system *acct* can grow by 100 blocks per day. The summary file is read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; *sha* is the default. There are zillions of options:

a Place all command names containing unprintable characters and those used only once under the name '***other.'

b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

c Besides total user, system, and real time for each command print percentage of total time over all commands.

j Instead of total minutes time for each category, give seconds per call.

l Separate system and user time; normally they are combined.

m Print number of processes and number of CPU minutes for each user.

n Sort by number of calls.

r Reverse order of sort.

s Merge accounting file into summary file */usr/adm/savacct* when done.

t For each command report ratio of real time to the sum of user and system times.

u Superseding all other flags, print for each command in the accounting file the user ID and command name.

v If the next character is a digit *n*, then type the name of each command used *n* times or fewer. Await a reply from the typewriter; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct	raw accounting
/usr/adm/savacct	summary
/usr/adm/usracct	per-user summary

SEE ALSO

ac(1), acct(2)

NAME

sact − print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the **−e** option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of − is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1	specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.
Field 2	specifies the SID for the new delta to be created.
Field 3	contains the logname of the user who will make the delta (i.e. executed a *get* for editing).
Field 4	contains the date that **get −e** was executed.
Field 5	contains the time that **get −e** was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff **–r**SID1 **–r**SID2 [**–p**] [**–s**n] 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.

–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.

FILES

/tmp/get????? Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1), sccsfile(5).
An Introduction to the Source Code Control System by *Eric* Allman

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**] [**−n**] [**−q**] [**−s**] [**−S** shell] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is saved in a file, and can be sent to the line printer later with *lpr*. If *file* is given, the typescript is saved there. If not, the typescript is saved in the file *typescript*.

To exit *script*, type control-D. This sends an end of file to all processes you have started up, and causes *script* to exit. For this reason, control-D behaves as though you had typed an infinite number of control-D's.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply. The options are:

−a Append to the typescript file instead of creating a new file.

−n Use the 'new' shell (interpretation of 'new' is installation dependent).

−q Suppress the 'script started' and 'script done' messages.

−s Use the 'standard' shell (usually *sh*(1)).

−S Use *shell*. If the requested shell is not available, *script* uses any shell it can find.

AUTHOR

Mark Horton

RESTRICTIONS

Since UNIX has no way to write an end-of-file down a pipe without closing the pipe, there is no way to simulate a single control-D without ending *script*.

The new shell has its standard input coming from a pipe rather than a tty, so *stty*(1) will not work, and neither will *ttyname*(1).

When the user interrupts a printing process, *script* attempts to flush the output backed up in the pipe for better response. Usually the next prompt also gets flushed.

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** script] [**–f** sfile] [file] ...

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f**'s, the flag **–e** may be omitted. The **–n** option suppresses the default output.

A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a 'D' command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

An *address* is either a decimal number that counts input lines cumulatively across files, a '$' that addresses the last line of input, or a context address, '/regular expression/', in the style of *ed*(1) modified thus:

The escape sequence '\n' matches a newline embedded in the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function '!' (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

An argument denoted *text* consists of one or more lines, all but the last of which end with '\' to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an 's' command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line.

An argument denoted *rfile* or *wfile* must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\
text

Append. Place *text* on the output before reading the next input line.

(2) b *label*

Branch to the ':' command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c\
text

Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2)d Delete the pattern space. Start the next cycle.

(2)D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2)g Replace the contents of the pattern space by the contents of the hold space.

(2)G Append the contents of the hold space to the pattern space.

(2)h Replace the contents of the hold space by the contents of the pattern space.

(2)H Append the contents of the pattern space to the hold space.

(1)i\
text Insert. Place *text* on the standard output.

(2)l List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two digit ascii, and long lines are folded.

(2)n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2)N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2)p Print. Copy the pattern space to the standard output.

(2)P Copy the initial segment of the pattern space through the first newline to the standard output.

(1)q Quit. Branch to the end of the script. Do not start a new cycle.

(2)r *rfile*
Read the contents of *rfile*. Place them on the output before reading the next input line.

(2)s/*regular expression/replacement/flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of '/'. For a fuller description see *ed*(1). *Flags* is zero or more of

g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.

p Print the pattern space if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2)t *label*
Test. Branch to the ':' command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a 't'. If *label* is empty, branch to the end of the script.

(2)w *wfile*
Write. Append the pattern space to *wfile*.

(2)x Exchange the contents of the pattern and hold spaces.

(2)y/*string1/string2/*
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*
Don't. Apply the *function* (or group, if *function* is '{') only to lines *not* selected by the address(es).

(0): *label*
This command does nothing; it bears a *label* for 'b' and 't' commands to branch to.

(1)= Place the current line number on the standard output as a line.

(2){ Execute the following commands through a matching '}' only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

ed(1), grep(1), awk(1)

NAME

sh, for, case, if, while, :, ., break, continue, cd, eval, exec, exit, export, login, newgrp, read, readonly, set, shift, times, trap, umask, wait – command language

SYNOPSIS

sh [**–ceiknrstuvx**] [arg] ...

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. See **invocation** for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non blank *words* separated by blanks (a blank is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a simple-command is its exit status if it terminates normally or 200+ *status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by |. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more *pipelines* separated by ;, &, && or || and optionally terminated by ; or &. ; and & have equal precedence which is lower than that of && and ||, && and || also have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding *pipeline* to be executed without waiting for it to finish. The symbol && (||) causes the *list* following to be executed only if the preceding *pipeline* returns a zero (non zero) value. Newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. The value returned by a command is that of the last simple-command executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
Each time a **for** command is executed *name* is set to the next word in the **for** word list If **in** *word* ... is omitted then **in "$@"** is assumed. Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**
A **case** command executes the *list* associated with the first pattern that matches *word*. The form of the patterns is the same as that used for file name generation.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
The *list* following **if** is executed and if it returns zero the *list* following **then** is executed. Otherwise, the *list* following **elif** is executed and if its value is zero the *list* following **then** is executed. Failing that the **else** *list* is executed.

while *list* [**do** *list*] **done**
A **while** command repeatedly executes the **while** *list* and if its value is zero executes the **do** *list;* otherwise the loop terminates. The value returned by a **while** command is that of the last executed command in the **do** *list*. **until** may be used in place of **while** to negate the loop termination test.

(*list*) Execute *list* in a subshell.

{ *list* } *list* is simply executed.

The following words are only recognized as the first word of a command and when not quoted.

if then else elif fi case in esac for while until do done { }

Command substitution.
The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing newlines are removed.

Parameter substitution.
The character **$** is used to introduce substitutable parameters. Positional parameters may be assigned values by **set**. Variables may be set by writing

name= *value* [*name*= *value*] ...

${*parameter***}**
A *parameter* is a sequence of letters, digits or underscores (a *name*), a digit, or any of the characters * @ # ? - $!. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is a digit then it is a positional parameter. If *parameter* is * or @ then all the positional parameters, starting with **$1**, are substituted separated by spaces. **$0** is set from argument zero when the shell is invoked.

${*parameter***-** *word***}**
If *parameter* is set then substitute its value; otherwise substitute *word*.

${*parameter***=** *word***}**
If *parameter* is not set then set it to *word;* the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***?***word***}**
If *parameter* is set then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter***+** *word***}**
If *parameter* is set then substitute *word;* otherwise substitute nothing.

In the above *word* is not evaluated unless it is to be used as the substituted string. (So that, for example, echo ${d-`pwd`} will only execute *pwd* if *d* is unset.)

The following *parameters* are automatically set by the shell.

#	The number of positional parameters in decimal.
-	Options supplied to the shell on invocation or by **set**.
?	The value returned by the last executed command in decimal.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following *parameters* are used but not set by the shell.

HOME
The default argument (home directory) for the **cd** command.

PATH The search path for commands (see **execution**).

MAIL If this variable is set to the name of a mail file then the shell informs the user of the arrival of mail in the specified file.

PS1 Primary prompt string, by default '**$** '.

PS2 Secondary prompt string, by default '> '.

IFS Internal field separators, normally **space**, **tab**, and **newline**.

Blank interpretation.
After parameter and command substitution, any results of substitution are scanned for internal field separator characters (those found in **$IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File name generation.
Following substitution, each command word is scanned for the characters *, ? and [. If one of these characters appears then the word is regarded as a pattern. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern then the word is left unchanged. The character . at the start of a file name or immediately following a /, and the character /, must be matched explicitly.

* Matches any string, including the null string.
? Matches any single character.
[...] Matches any one of the characters enclosed. A pair of characters separated by − matches any character lexically between the pair.

Quoting.
The following characters have a special meaning to the shell and cause termination of a word unless quoted.

; & ( ) | < > newline space tab

A character may be *quoted* by preceding it with a ****. **\newline** is ignored. All characters enclosed between a pair of quote marks (´´), except a single quote, are quoted. Inside double quotes ("") parameter and command substitution occurs and **** quotes the characters **\ ´ " $**.

"$*" is equivalent to **"$1 $2 ..."** whereas
"$@" is equivalent to **"$1" "$2" ...** .

Prompting.
When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a newline is typed and further input is needed to complete a command then the secondary prompt (**$PS2**) is issued.

Input output.
Before a command is executed its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are not passed on to the invoked command. Substitution occurs before *word* or *digit* is used.

< *word* Use file *word* as standard input (file descriptor 0).

> *word* Use file *word* as standard output (file descriptor 1). If the file does not exist then it is created; otherwise it is truncated to zero length.

>> *word*
Use file *word* as standard output. If the file exists then output is appended (by seeking to the end); otherwise the file is created.

<< *word*
The shell input is read up to a line the same as *word*, or end of file. The resulting document becomes the standard input. If any character of *word* is quoted then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\newline** is ignored, and **** is used to quote the characters **\ $ ´** and the first character of *word*.

<& *digit*

The standard input is duplicated from file descriptor *digit;* see *dup*(2). Similarly for the standard output using >.

<&- The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example,

... 2>&1

creates file descriptor 2 to be a duplicate of file descriptor 1.

If a command is followed by & then the default standard input for the command is the empty file (/dev/null). Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input output specifications.

Environment.
The environment is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list; see *exec*(2) and *environ*(5). The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a *parameter* for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these *parameters* or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's *parameter* to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to *parameters*. Thus these two lines are equivalent

TERM=450 cmd args
(export TERM; TERM=450; cmd args)

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if the occur after the command name. The following prints 'a=b c' and 'c':
echo a=b c
set -k
echo a=b c

Signals.
The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent. (But see also **trap.**)

Execution.
Each time a command is executed the above substitutions are carried out. Except for the 'special commands' listed below a new process is created and an attempt is made to execute the command via an *exec*(2).

The shell parameters **the the com mand. Each alternative directory name is separated by a colon** :). The default path is **:/bin:/usr/bin.** If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an *a.out* file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a subshell.

Special commands.
The following commands are executed in the shell process and except where specified no input output redirection is permitted for such commands.

: No effect; the command does nothing.

. *file* Read and execute commands from *file* and return. The search path **$PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **$HOME** is the default *arg*.

eval [*arg ...*]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg ...*]
The command specified by the arguments is executed in place of this shell without creating a new process. Input output arguments may appear and if no other arguments are given cause the shell input output to be modified.

exit [*n*]
Causes a non interactive shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed. (An end of file will also exit from the shell.)

export [*name ...*]
The given names are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given then a list of exportable names is printed.

login [*arg ...*]
Equivalent to 'exec login arg ...'.

newgrp [*arg ...*]
Equivalent to 'exec newgrp arg ...'.

read *name ...*
One line is read from the standard input; successive words of the input are assigned to the variables *name* in order, with leftover words to the last variable. The return code is 0 unless the end-of-file is encountered.

readonly [*name ...*]
The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given then a list of all readonly names is printed.

set [**−eknptuvx** [*arg ...*]]

- **−e** If non interactive then exit immediately if a command fails.
- **−k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **−n** Read commands but do not execute them.
- **−t** Exit after reading and executing one command.
- **−u** Treat unset variables as an error when substituting.
- **−v** Print shell input lines as they are read.
- **−x** Print commands and their arguments as they are executed.
- **−** Turn off the **−x** and **−v** options.

These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**.

Remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, etc. If no arguments are given then the values of all names are printed.

shift The positional parameters from **$2**... are renamed **$1**...

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
Arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by invoked commands. If *n* is 0 then the command *arg* is executed on exit from the shell, otherwise upon receipt of signal *n* as numbered in *signal*(2). *Trap* with no arguments prints a list of commands associated with each signal number.

umask [*nnn*]
The user file creation mask is set to the octal value *nnn* (see *umask*(2)). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]
Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for. The return code from this command is that of the process waited for.

Invocation.
If the first character of argument zero is −, commands are read from **$HOME/.profile**, if such a file exists. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked.

−**c** *string* If the −**c** flag is present then commands are read from *string*.

−**s** If the −**s** flag is present or if no arguments remain then commands are read from the standard input. Shell output is written to file descriptor 2.

−**i** If the −**i** flag is present or if the shell input and output are attached to a terminal (as told by *gtty*) then this shell is *interactive*. In this case the terminate signal SIGTERM (see *signal*(2)) is ignored (so that 'kill 0' does not kill an interactive shell) and the interrupt signal SIGINT is caught and ignored (so that **wait** is interruptable). In all cases SIGQUIT is ignored by the shell.

The remaining flags and arguments are described under the **set** command.

FILES

$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

test(1), exec(2),

DIAGNOSTICS

Errors detected by the shell, such as syntax errors cause the shell to return a non zero exit status. If the shell is being used non interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also **exit**).

RESTRICTIONS

If << is used to provide standard input to an asynchronous process invoked by &, the shell gets mixed up about naming the input document. A garbage file /tmp/sh* is created, and the shell complains about not being able to find the file by another name.

NAME

size – size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, and bss portions, and their sum in octal and decimal, of each object-file argument. For overlay text files (0430 or 0431) *size* prints the (decimal) number of bytes contained in the root text segment, each overlay text segment, the data and bss segments, the sum of the root, data, and bss segments, and the total text size. If no file is specified, **a.out** is used.

SEE ALSO

a.out(5)

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3)

RESTRICTIONS

Time must be less than 65536 seconds.

NAME

sort – sort or merge files

SYNOPSIS

sort [**–mubdfinrt***x*] [+*pos1* [–*pos2*]] ... [**–o** name] [**–T** directory] [name] ...

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '–' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option n implies option **b**.

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

The notation +*pos1* –*pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing –*pos2* means the end of the line. Under the –**t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T The next argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples. Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort –u +0f +0 list
```

Print the password file (*passwd*(5)) sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

FILES

/usr/tmp/stm*, /tmp/*: first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **-c**.

RESTRICTIONS

Very long lines are silently truncated.

NAME

spell, spellin, spellout − find spelling errors

SYNOPSIS

spell [option] ... [file] ...

/usr/lib/spellin [list]

/usr/lib/spellout [**−d**] list

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff, tbl* and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell*. Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hash list.

To create the basic hashed spelling list:

```
cd /usr/lib; spellin < /usr/dict/words >hlist
```

To create the American hashed spelling list:

```
cd /usr/lib; (cat american local)|spellin hlist >hlista
```

To create the British hashed spelling list:

```
cd /usr/lib; (cat british local)|spellin hlist >hlistb
```

FILES

SPELL_D=/usr/dict/hlist[ab]: hashed spelling lists, American & British
SPELL_S=/usr/dict/hstop: hashed stop list
SPELL_H=/usr/dict/spellhist: history file
/usr/dict/words: full dictionary
/usr/dict/american: american spelling
/usr/dict/british: british spelling
/usr/dict/local: local spelling
/usr/lib/spell
/usr/lib/spellin
/usr/lib/spellout

deroff(1), sort(1), tee(1), sed(1)

Note: *spell* searches the user's environment for variables of the form 'SHELL_X', if found, it uses those by default.

RESTRICTIONS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions.
British spelling was done by an American.

NAME

spline – interpolate smooth curve

SYNOPSIS

spline [option] ...

DESCRIPTION

Spline takes pairs of numbers from the standard input as abcissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers,* 2nd ed., 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1).

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

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

–k The constant k used in the boundary value computation

$$y_0'' = ky_1'',\ y_n'' = ky_{n-1}''$$

is set by the next argument. By default $k = 0$.

–n Space output points so that approximately n intervals occur between the lower and upper x limits. (Default $n = 100$.)

–p Make output periodic, i.e. match derivatives at ends. First and last input values should normally agree.

–x Next 1 (or 2) arguments are lower (and upper) x limits. Normally these limits are calculated from the data. Automatic abcissas start at lower limit (default 0).

SEE ALSO

graph(1)

DIAGNOSTICS

When data is not strictly monotone in x, *spline* reproduces the input without interpolating extra points.

RESTRICTIONS

A limit of 1000 input points is enforced silently.

NAME

split − split a file into pieces

SYNOPSIS

split [− *n*] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically. If no output name is given, **x** is default.

If no input file is given, or if − is given in its stead, then the standard input file is used.

NAME

strings – find the printable strings in a object, or other binary, file

SYNOPSIS

strings [–] [–**o**] [– *number*] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the – flag is given, *strings* only looks in the initialized data space of object files. If the –**o** flag is given, then each string is preceded by its offset in the file (in octal). If the – *number* flag is given then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

SEE ALSO

od(1)

AUTHOR

Bill Joy

RESTRICTIONS

The algorithm for identifying strings is extremely primitive.

NAME

strip – remove symbols and relocation bits

SYNOPSIS

strip name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and loader. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the **–s** option of *ld*.

Strip automatically accommodates overlay text files (0430 or 0431).

FILES

/tmp/stm? temporary file

SEE ALSO

ld(1)

NAME

struct – structure Fortran programs

SYNOPSIS

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (.e.g. '.GT.' into '>'). The output is appropriately indented.

The following options may occur in any order.

–s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally, a statement whose first nonblank character is not alphanumeric is treated as a continuation.

–i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)

–a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch {
        case pred1: code
        case pred2: code
        case pred3: code
        default: code
    }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

–b Generate goto's instead of multilevel break statements.

–n Generate goto's instead of multilevel next statements.

–e*n* If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit code segments with fewer than *n* statements to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

RESTRICTIONS

Struct knows Fortran 66 syntax, but not full Fortran 77 (alternate returns, IF...THEN...ELSE, etc.)
If an input Fortran program contains identifiers which are reserved words in Ratfor, the structured version of the program will not be a valid Ratfor program.
Extended range DO's generate cryptic errors.
Columns 73-80 are not special even when –s is in effect.
Will not generate Ratfor FOR statements.

NAME

stty – set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

even allow even parity
–even disallow even parity
odd allow odd parity
–odd disallow odd parity
raw raw mode input (no erase, kill, interrupt, quit, EOT; parity bit passed back)
–raw negate raw mode
cooked same as '–raw'
cbreak make each character available to *read*(2) as received; no erase and kill
–cbreak
make characters available to *read* only when newline is received
–nl allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl accept only new-line to end lines
echo echo back every character typed
–echo do not echo characters
lcase map upper case to lower case
–lcase do not map case
–tabs replace tabs by spaces when printing
tabs preserve tabs
ek reset erase and kill characters back to normal # and @
erase *c* set erase character to *c*. *C* can be of the form '*X*' which is interpreted as a 'control X'.
kill *c* set kill character to *c*. '*X*' works here also.
cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))
nl0 nl1 nl2 nl3
select style of delay for linefeed
tab0 tab1 tab2 tab3
select style of delay for tab
ff0 ff1 select style of delay for form feed
bs0 bs1 select style of delay for backspace
tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal
tn300 set all modes suitable for a General Electric TermiNet 300
ti700 set all modes suitable for Texas Instruments 700 series terminal
tek set all modes suitable for Tektronix 4014 terminal
hup hang up dataphone on last close.
–hup do not hang up dataphone on last close.
0 hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

SEE ALSO

ioctl(2), tabs(1)

NAME

stty – set terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. With the argument 'all', all normally used option settings are reported. With the argument 'everything', everything *stty* knows about is printed. The option strings are selected from the following set:

even allow even parity input
– even disallow even parity input
odd allow odd parity input
– odd disallow odd parity input
raw raw mode input (**no** input processing (erase, kill, interrupt, ...); parity bit passed back)
– raw negate raw mode
cooked same as '– raw'
cbreak make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
– cbreak make characters available to *read* only when newline is received
– nl allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl accept only new-line to end lines
echo echo back every character typed
– echo do not echo characters
lcase map upper case to lower case
– lcase do not map case
tandem enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
– tandem disable flow control
– tabs replace tabs by spaces when printing
tabs preserve tabs
ek set erase and kill characters to # and @

For the following commands which take a character argument *c*, you may also specify *c* as the 'u' or 'undef', to set the value to be undefined. A value of '^x', a 2 character sequence, is also interpreted as a control character, with '^?' representing delete.

erase *c* set erase character to *c* (default '#', but often reset to ^H.)
kill *c* set kill character to *c* (default '@', but often reset to ^U.)
intr *c* set interrupt character to *c* (default DEL or ^? (delete), but often reset to ^C.)
quit *c* set quit character to *c* (default control \.)
start *c* set start character to *c* (default control Q.)
stop *c* set stop character to *c* (default control S.)
eof *c* set end of file character to *c* (default control D.)
brk *c* set break character to *c* (default undefined.) This character is an extra wakeup causing character.
cr0 cr1 cr2 cr3
select style of delay for carriage return (see *ioctl*(2))
nl0 nl1 nl2 nl3

select style of delay for linefeed

tab0 tab1 tab2 tab3
select style of delay for tab

ff0 ff1 select style of delay for form feed

bs0 bs1 select style of delay for backspace

tty33 set all modes suitable for the Teletype Corporation Model 33 terminal.

tty37 set all modes suitable for the Teletype Corporation Model 37 terminal.

vt05 set all modes suitable for Digital Equipment Corp. VT05 terminal

dec set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and 'newcrt'.)

tn300 set all modes suitable for a General Electric TermiNet 300

ti700 set all modes suitable for Texas Instruments 700 series terminal

tek set all modes suitable for Tektronix 4014 terminal

0 hang up phone line immediately

50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb
Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

A teletype driver which supports the job control processing of *csh*(1) with more functionality than the basic driver is introduced in *newtty*(4) and fully described in *tty*(4). The following options apply only to it.

new Use new driver (switching flushes typeahead).

old Use old driver (switching flushes typeahead).

crt Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)

crtbs Echo backspaces on erase characters.

prterase For printing terminal echo erased characters backwards within '\' and '/'.

crterase Wipe out erased characters with 'backspace-space-backspace.'

– crterase
Leave erased characters visible; just backspace.

crtkill Wipe out input on like kill ala **crterase.**

– crtkill Just echo line kill character and a newline on line kill.

ctlecho Echo control characters as '^*x*' (and delete as '^?'.) Print two backspaces following the EOT character (control D).

– ctlecho Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.

decctlq After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.

– decctlq After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)

tostop Background jobs stop if they attempt terminal output.

– tostop Output from background jobs to the terminal is allowed.

tilde Convert '~' to '`' on output (for Hazeltine terminals).

– tilde Leave poor '~' alone.

flusho Output is being discarded usually because user hit control O (internal state bit).

– flusho Output is not being discarded.

pendin Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).

– pendin Input is not pending.

intrup Send a signal (SIGTINT) to the terminal control process group whenever an input record (line in cooked mode, character in cbreak or raw mode) is available

for reading.

−intrup Don't send input available interrupts.

mdmbuf Start/stop output on carrier transitions (not implemented).

−mdmbuf
Return error if write attempted after carrier drops.

litout Send output characters without any processing.

−litout Do normal output processing, inserting delays, etc.

etxack Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c* set suspend process character to *c* (default control Z.)

dsusp *c* set delayed suspend process character to *c* (default control Y.)

rprnt *c* set reprint line character to *c* (default control R.)

flush *c* set flush output character to *c* (default control O.)

werase *c* set word erase character to *c* (default control W.)

lnext *c* set literal next character to *c* (default control V.)

SEE ALSO

tabs(1), tset(1), ioctl(2), newtty(4), tty(4)

NAME

style – analyze surface characteristics of a document

SYNOPSIS

style [**–ml**] [**–mm**] [**–a**] [**–e**] [**–l** num] [**–r** num] [**–p**] [**–P**] file ...

DESCRIPTION

Style analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because *style* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **–ms** may be overridden with the flag **–mm**. The flag **–ml**, which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

–a print all sentences with their length and readability index.

–e print all sentences that begin with an expletive.

–p print all sentences that contain a passive verb.

–l*num* print all sentences longer than *num*.

–r*num*
print all sentences whose readability index is greater than *num*.

–P print parts of speech of the words in the document.

SEE ALSO

deroff(1), diction(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

su – substitute user id temporarily

SYNOPSIS

su [userid]

DESCRIPTION

Su demands the password of the specified *userid,* and if it is given, changes to that *userid* and invokes the Shell *sh(1)* without changing the current directory or the user environment (see *environ(5)*). The new user ID stays in force until the Shell exits.

If no *userid* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '*#*' for its usual prompt.

SEE ALSO

sh(1)

NAME

su – substitute user id temporarily

SYNTAX

su [–] [userid]

DESCRIPTION

Su demands the password of the specified *userid,* and if it is given, changes to that *userid* and invokes a shell without changing the current directory or the user environment. The new user ID stays in force until the shell exits. If the 'SHELL' environment variable is set, it is used as the name of the new shell, and, if necessary, the previous user environment is restored for that shell. This allows for the use of multiple shells with *su*. If the optional - flag is given, the shell of the specified user will be the new shell, the environment is set as for that user, and the new shell is started in the new home directory. This can be used to simulate the login environment of a specified user.

If no *userid* is specified, 'root' is assumed. If the file /usr/adm/sulog exists, records are kept there of all *su* attempts to the root id. To remind the superuser of his responsibilities, the shell substitutes '#' for its usual prompt.

FILES

/usr/adm/sulog

SEE ALSO

csh(1), newgrp(1), sh(1)

NAME

sum – sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinguishable from end of file on most devices; check the block count.

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See *sync*(2) for details.

SEE ALSO

sync(2), update(8)

NAME

tabs – set terminal tabs

SYNOPSIS

tabs [**–n**] [terminal]

DESCRIPTION

Tabs sets the tabs on a variety of terminals. Various of the terminal names given in *term*(7) are recognized; the default is, however, suitable for most 300 baud terminals. If the **–n** flag is present then the left margin is not indented as is normal.

SEE ALSO

stty(1), term(7)

NAME

tail − deliver the last part of a file

SYNOPSIS

tail [±number[lbc][fr]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance + *number* from the beginning, or − *number* from the end of the input. *Number* is counted in units of lines, blocks or characters, according to the appended option **l**, **b** or **c**. When no units are specified, counting is by lines.

Specifying **r** causes tail to print lines from the end of the file in reverse order. The default for **r** is to print the entire file this way. Specifying **f** causes *tail* to not quit at end of file, but rather wait and try to read repeatedly in hopes that the file will grow.

SEE ALSO

dd(1)

RESTRICTIONS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length.

Various kinds of anomalous behavior may happen with character special files.

NAME

tapeskip – skip file on /dev/rht0

SYNOPSIS

tapeskip count

DESCRIPTION

Tapeskip skips a specified number of files on /dev/rht0. This may be usefull to acces files in different format on the same tape.

FILES

/etc/tapeskip
/dev/rht0

SEE ALSO

tapeskip is shellscript using dd(1).

NAME

tar – tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped 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. The **c** function implies this.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r.**

The following characters may be used in addition to the letter which selects the function desired.

0,...,7 This modifier selects an alternate drive on which the tape is mounted. The default is **1.**

v Normally *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w causes *tar* to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.

f causes *tar* to use the next argument as the name of the archive instead of /dev/mt?. If the name of the file is '–', tar writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain *Tar* can also be used to move hierarchies with the command

cd fromdir; tar cf - . | (cd todir; tar xf -)

b causes *tar* to use the next argument as the blocking factor for tape records. The default is 20 (the maximum). 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 dumped. If this is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The mod time will be the time of extraction.

FILES

/dev/rmt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

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 magtape driver cannot backspace raw magtape. If the archive is on a disk file the **b** option should not be used at all, as updating an archive stored in this manner can destroy it.
The current limit on file name length is 100 characters.

NAME

tar – tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape or RX50 floppy diskette. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory. *Tar* also saves and restores *special files*.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The **c** function implies this. Writing files on the end of a *tar* tape creates a second archive on that tape. Using the *r* option with a tape where the second archive already exists will overwrite the second archive, not create a third archive. Dealing with the second archive can be clumsy, i.e., the no rewind on close tape special file must be used to position the tape after the first archive. For example:

tar tf /dev/nrht0 >/dev/null

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape. The files are appended to the end of the tape (see *r* above).

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

The following characters may be used in addition to the letter which selects the function desired.

0,...,7 This modifier selects the drive on which the tape is mounted. The default is **0**.

v Normally *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w causes *tar* to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.

f causes *tar* to use the next argument as the name of the archive instead of /dev/rmt? or /dev/rht?, for example, *tar* could archive to an RX50 floppy diskette with the command

tar cf /dev/rrx? files

If the name of the file is '–', tar writes to standard output or reads from

standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain *Tar* can also be used to move hierarchies with the command

cd fromdir; tar cf - . | (cd todir; tar xf -)

b causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 20. This option should only be used with raw magnetic tape archives or with raw disk archives that will never be updated, e.g., RX50 diskettes. The block size is determined automatically when reading tape and disk archives (key letters 'x' and 't'), unless both the 'b' and 'f' keys are specified. In that case the blocking factor specified by the 'b' key is used unconditionally.

l tells *tar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m tells *tar* to not restore the modification times. The mod time will be the time of extraction.

n Select (NRZI) 800 BPI density on the specified tape unit, /dev/rmt0 by default.

p Changes the mode of extracted files to the original mode, as written to tape. By default, the *tar* user's mode is passed to the file.

o Suppress the normal directory information. On output, tar normally places information specifying owner and mode of directories. This allows tar to extract the files even if the top level directory does not exist.

d Selects the RX50 diskettes as the *tar* medium, /dev/rrx0 by default. In addition, the blocking factor is established as 10, unless otherwise specified.

FILES

/dev/rmt?
/dev/rht?
/dev/rrx?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

RESTRICTIONS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **b** option should not be used with archives that are going to be updated. The current magtape driver cannot backspace raw magtape. If the archive is on a disk file the **b** option should not be used at all, as updating an archive stored in this manner can destroy it. The **b** option may be used with raw disk archives if and only if those archives will never be updated.
The current limit on file name length is 100 characters.
Previous versions of *tar* will restore special files as empty regular files, and may return "cannot create" messages when extracting directory files.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

Tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between .TS and .TE command lines, which are assumed to describe tables and reformatted. Details are given in the reference manual.

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

Household Population		
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Branchburg	1644	3.49
Bridgewater	7897	3.81
Far Hills	240	3.19

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

nroff(1), eqn(1)
M. E. Lesk, *TBL*.

NAME

tc – phototypesetter simulator

SYNOPSIS

tc [**–t**] [**–sN**] [**–pL**] [file]

DESCRIPTION

Tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of *tc* is intended for a Tektronix 4015 (a 4014 teminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

troff –t file | tc

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; **s**N will cause the next N pages to be skipped; and **!**line will send line to the shell.

The command line options are:

–t Don't wait between pages; for directing output into a file.

–sN Skip the first N pages.

–pL Set page length to L. L may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

'– l w' Multiply the default aspect ratio, 1.5, of a displayed page by *l/w*.

SEE ALSO

troff (nroff(1)), plot(1)

RESTRICTIONS

Font distinctions are lost.

The aspect ratio option is unbelievable.

NAME

tee – pipe fitting

SYNOPSIS

tee [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the named *files*.

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. Option **–i** ignores interrupts; option **–a** causes the output to be appended to the *files* rather than overwriting them.

NAME

test − condition command

SYNOPSIS

test expr

DESCRIPTION

test evaluates the expression *expr*, and if its value is true then returns zero exit status; otherwise, a non zero exit status is returned. *test* returns a non zero exit if there are no arguments.

The following primitives are used to construct *expr*.

−r file true if the file exists and is readable.

−w file true if the file exists and is writable.

−f file true if the file exists and is not a directory.

−d file true if the file exists and is a directory.

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

−t [fildes]
true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

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

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

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

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

s1 true if *s1* is not the null string.

n1 **−eq** n2
true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, or **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

! unary negation operator

−a binary *and* operator

−o binary *or* operator

(expr)
parentheses for grouping.

−a has higher precedence than **−o**. Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the Shell and must be escaped.

SEE ALSO

sh(1), find(1)

NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The given command is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The execution time can depend on what kind of memory the program happens to land in; the user time in MOS is often half what it is in core.

The times are printed on the diagnostic output stream.

RESTRICTIONS

Elapsed time is accurate to the second, while the CPU times are measured to the 60th second. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

NAME

tip, cu − connect to a remote system

SYNOPSIS

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

DESCRIPTION

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

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

~^D ~. Drop the connection and exit (you may still be logged in on the remote machine).

~c [*name*] Change directory to name (no argument implies change to your home directory).

~! Escape to a shell (exiting the shell will return you to tip).

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

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

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

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the 'to' file defaults to the 'from' file name if it isn't specified. The remote host executes the command string 'cat 'from';echo ^A ' to send the file to *tip*.

~| Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.

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

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

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

~? Get a summary of the tilde escapes

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

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

When *tip* prompts for an argument (e.g. during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

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

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

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

VARIABLES

Tip maintains a set of *variables* which control its operation. Some of these variable are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the 's' escape. The syntax for variables is patterned after *vi*(1) and *Mail*(1). Supplying 'all' as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example 'escape?' displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a '!' to the name. Other variable types are set by concatenating an '=' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the '~s' prefix in a file *.tiprc* in one's home directory). The −v option causes *tip* to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

script
(bool) Session scripting mode; abbreviated *sc*; default is *off*. When *script* is *true*, *tip* will record everything transmitted by the remote machine in the script record file specified in *record*. If the *beautify* switch is on, only printable ASCII characters will be included in the script file (those characters betwee 040 and 0177). The variable *exceptions* is used to indicate characters which are an exception to the normal beautification rules.

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

verbose
(bool) Verbose mode; abbreviated *verb*; default is *true*. When verbose mode is enabled, *tip* prints messages while dialing, shows the current number of lines

transferred during a file transfer operations, and more.

SHELL
(str) The name of the shell to use for the ~! command; default value is '/bin/sh', or taken from the environment.

HOME
(str) The home directory to use for the ~c command; default value is taken from the environment.

FILES

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

DIAGNOSTICS
Diagnostics are, hopefully, self explanatory.

SEE ALSO
remote(5), phones(5)

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

NAME

tk – paginator for the Tektronix 4014

SYNOPSIS

tk [**–t**] [*–N*] [**–p***L*] [file]

DESCRIPTION

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

The command line options are:

–t Don't wait between pages; for directing output into a file.

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

–p*L* Set page length to *L* lines.

SEE ALSO

pr(1)

NAME

touch – update date last modified of a file

SYNOPSIS

touch [**–c**] file ...

DESCRIPTION

Touch attempts to set the modified date of each *file*. This is done by reading a character from the file and writing it back.

If a *file* does not exist, an attempt will be made to create it unless the **–c** option is specified.

NAME

tp – manipulate tape archive

SYNOPSIS

tp [key] [name ...]

DESCRIPTION

Tp saves and restores files on DECtape or magtape. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the tape. If files with the same names already exist, they are replaced. 'Same' is determined by string comparison, so './abc' can never be the same as '/usr/dmr/abc' even if '/usr/dmr' is the current directory. If no file argument is given, '.' is the default.

u updates the tape. **u** is like **r**, but a file is replaced only if its modification date is later than the date stored on the tape; that is to say, if it has changed since it was dumped. **u** is the default command if none is given.

d deletes the named files from the tape. At least one name argument must be given. This function is not permitted on magtapes.

x extracts the named files from the tape to the file system. The owner and mode are restored. If no file argument is given, the entire contents of the tape are extracted.

t lists the names of the specified files. If no file argument is given, the entire contents of the tape is listed.

The following characters may be used in addition to the letter which selects the function desired.

h Specifies 1600 BPI magtape as opposed to DECtape.

m Specifies 800 BPI magtape as opposed to DECtape.

0,...,7 This modifier selects the drive on which the tape is mounted. For DECtape, **x** is default; for magtape '0' is the default.

v Normally *tp* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

c means a fresh dump is being created; the tape directory is cleared before beginning. Usable only with **r** and **u**. This option is assumed with magtape since it is impossible to selectively overwrite magtape.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

f Use the first named file, rather than a tape, as the archive. This option is known to work only with **x**.

w causes *tp* to pause before treating each file, type the indicative letter and the file name (as with **v**) and await the user's response. Response **y** means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response **x** means 'exit'; the *tp* command terminates immediately. In the **x** function, files previously asked about have been extracted

already. With **r, u,** and **d** no change has been made to the tape.

FILES

/dev/tap?
/dev/mt?
/dev/ht?

SEE ALSO

ar(1), tar(1)

DIAGNOSTICS

Several; the non-obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

RESTRICTIONS

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by *tp* difficult to carry to other machines; *tar*(1) avoids the problem.

NAME

tr − translate characters

SYNOPSIS

tr [−**cds**] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options −**cds** may be used: −**c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; −**d** deletes all input characters in *string1;* −**s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a*− *b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z '\012' <file1 >file2
```

SEE ALSO

ed(1), ascii(7)

RESTRICTIONS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

trac – a TRAC-processor

SYNOPSIS

trac

DESCRIPTION

The I/O is implemented by means of channels, numbered from zero to five. First a file is reserved (opened or created) by the FI or FO function. The file is brought after it is selected for reading or writing by the SD function. Upon the start the standard output file is selected on channel "1" and the standard input file on channel "0". If the end-of-file of the standard input file is reached, then the terminal will be the standard input file (channel "0"). The only way to stop the TRAC-processor is by means of the function HL. Two interrupts are implemented. CNTR/C: Stop the active TRAC-program;
load the "idling program". CNTR/Λ: Stop the active TRAC-program;
select standard input and output file;
evaluate (MO,I) function;
load "idling program". Upon the start the "idling program" has the form: :(ps,:(rs))' The form string is implemented in a virtual array. By means of the function (FB,x,z) the first "x" characters of the form string are placed in "core". Here is a list of implemented functions. See for an comprehensive description the graduate report of Kees Langelaan. Input en outputfunctions

:(PS,x)	Print String
:(RS)	Read String
:(RC)	Read one Character
:(RM,x1,x2,....)	Read Meta. Read until one of the strings "x1", "x2", etc. is encountered
:(RN,d)	Read N characters
:(CM,x)	Change Meta. Change activation character in the first character of "x"
:(FI,x,d,z)	reserve File "x" for Input on channel "d"
:(FO,x,d,z)	reserve File "x" for Output on channel "d"
:(SD,d,z)	Select Device (file) with channel "d"
:(FC,d,z)	File with channel "d" is Closed

Trace functions

:(TN)	Trace oN
:(TF)	Trace ofF

Manipulations with forms

:(DS,n,x)	Define String with name "n" and body "x" and delete an old form with name "n", if there is such a form
:(DN,n,x)	Define New string without deleting an old form
:(DI,n,x)	Define as Integer
:(DD,n1,n2,....)	Delete Definitons "n1", "n2", etc.
:(DA)	Delete All definitions (forms)

only the protected forms (see YP)

:(LN,x) give List of Names, separated by "x"

:(PF,n1,n2,....) Print Forms "n1", "n2", etc.

:(SS,n,x1,x2,....) Segment String "n" (the form-body of "n") on all occurrences of x1, x2 etc. and replace these by segment-marks 1, 2 etc.

:(SG,n,x1,x2,....) SinGular segmentation
As SS only "one occurrence" instead of "all occurrences"

:(MS,n,x1,x2,....) Multi-Segmentation
As SS only multi-marks instead of segment-marks

:(CL,n,x1,x2,....) CalL form "n" with arguments "x1", "x2" etc.

:(CF,n,x1,x2,....) Call Form
As CL only the multi-marks are copied and the segment-marks with empty arguments are copied

:(CH,n,x1,x2) CHange form "n"
Substrings "x1" are replaced by "x2"

Research of forms

:(CR,n,d) Cursor Reset. Reset form pointer

:(CS,n,z) Call Segment. Give next segment

:(NS,n,d,z) call N Segments. Call "d" segments

:(CC,n,z) Call Character
Give next character of form "n"

:(CN,n,d,z) Call N characters
Give next "d" characters of form "n"

:(IN,n,x,z) INitial
Search till "x" appears in "n", return with all characters passed, but not including "x", if found, otherwise "z"

:(IL,n,x,z) Initial Left
As IR, but left of form pointer

:(SP,n,x,z) SPan. Search till a character of "n" doesn't appear in "x". Return the passed characters, if found, otherwise "z"

:(BK,n,x,z) BreaK. Search till a character of "n" appears in "x". Return the passed characters, if found, otherwise "z"

:(CG,n,z) Call Gap. Result is the first mark

:(CE,n) Call front. The result is the part of "n" on the left of the form pointer

Arithmetic functions

:(AD,g1,g2) ADd

:(SU,g1,g2) SUbtract
:(ML,g1,g2) MuLtiply
:(DV,g1,g2,z) DiVide. Integer division
:(MN,g,d) Modify Numeric base of integer "g"

Boolean functions

:(BU,b1,b2) Boolean Union
:(BI,b1,b2) Boolean Intersection
:(BC,b) Boolean Complement
:(BS,d,b) Boolean Shift
:(BR,d,b) Boolean Rotate

Test functions

:(EQ,x1,x2,t,f) EQual. If "x1"="x2" then "t" else "f"
:(GR,g1,g2,t,f) GReater. If "d1">"d2" then "t" else "f"
:(LG,x1,x2,t,f) Lexicografic Greater. As GR but now for strings instead of numbers
:(YT,n,x,t,f) Ys there. If the string "x" appears in form "n" then "t" else "f"
:(NB,n,t,f) Name branch. If name "n" presents in form string then "t" else "f"
:(EM,t,f) End of Medium branch
If last input function find an end-of-file then "t" else "f"
:(NI,t,f) Neutral Implied. If last implicit function was neutral then "t" else "f"

Other functions

:(HL) Halt. Stop the TRAC-processor
:(RT,d) ReTurn "d" levels
:(RI,x) Return to idling level and do "x"
:(YD,n1,n2,....) Dump form "n1", "n2" etc.
:(YF) Fetch one dumped form
:(YN,d) Fetch "d" dumped forms
:(CO,x1,x2) Character to Octal. Result is the octal value of the characters in string "x2" separated by "x1"
:(OC,o1,o2,....) Octal to Character
Result is the string of characters with octal value "o1", "o2" etc.
:(SC,d) Super Character
Result is a mark with number "d"
If neutral function then multi-mark else segment-mark
:(LP,n) Left of Pointer. Number of characters left of form pointer. If neutral function exclusive marks else

inclusive

:(RP,n) Right of Pointer. See LP

:(SW,x1,x2,....) SWitch. If "x1"="x3" then "x4" else if "x1"="x5" then "x6" else etc. else "x2"

:(RV,x) ReVerse. Result is the reversed value of string "x"

:(SQ) SQueeze. Remove all "open places" in form string

:(YP,n) Protect forms
If "(yp)" then make all forms unprotected (unattainable)
If "(yp,n)" then make all forms above form "n" and "n" protected

:(FB,d,z) Fetch Block. Set first "d" characters of form string permanent in "core"

:(XR,o) eXperimental Read

:(XW,o1,o2) eXperimental Write

Mode functions

:(MO,I) Mode Idle
Syntactic character = (:)
Meta character = (')
Numeric base = 10
Boolean base = 8
The TRAC-processor is set in:
print half-duplex mode
read half-duplex mode
normal search order
trace off mode

:(MO,MN,d) Modify Numeric base. From now on all arithmetic functions will be done in base "d"

:(MO,MB,d) Modify Boolean base. From now on all boolean functions will be done in base "d"

:(MO,MS,x) Modify Syntactic character. The function indication character will be changed in the first charater of "x" (default is :)

:(MO,ML,d) Modify Linelength

:(MO,TN) Trace Normal

:(MO,TA) Trace All

:(MO,RE) Read Echoplex. Everything input with RS, RC, RN and RM is echoed on the output file

:(MO,RH) Read Half-duplex. The input is not echoed

:(MO,PE) Print Echoplex. Everything output with PS and PF is echoed on the

standard output file (channel 0)

:(MO,PH) Print Half-duplex.The output is not echoed

:(MO,ON) Overflow Normal

:(MO,OT,x) Overflow Trap. If overflow then return to idle and do "x"

:(MO,SM,x) Scan algorithm Modification
If "x"="on" then the input string will fill up at a shortage of ')'
If "x"="of" then the input string won't fill up at a shortage of ')'

:(MO,SO,x) Search Order. If "x"="on" then the TRAC-processor is set in "special search order" if "x"="of" then "normal search order"

:(MO,QN) Query Numeric base

:(MO,QB) Query Boolean base

:(MO,QL) Query Linelength

:(MO,QF) Query length of Form string

:(MO,QE) Query length "open places" in form string

:(MO,QA) Query maximum length brought by the active and neutral string

:(MO,ST) Set Time to zero

:(MO,TI) query TIme

FILES

/usr/man/doc/trac/*

NAME

tree - generate a nice listing of a UNIX file tree

SYNOPSIS

tree [**-d**] [**-nnn**] [**-lnnn**] [**-t**] [top_of_tree]

DESCRIPTION

Tree generates a nice listing of a UNIX tree. It uses the *termcap*(5U) features to underline the executables and high light (reverse or bold) the directories. The lowest line of the screen is used for showing the current possition in the tree. For this a capability to set the scroll region on the terminal is needed (the **sc** capability).

OPTIONS

-nnn The depth of the tree is limited to **nnn** , default is 99.

-d Only list the directories, not the contents.

-lnnn Make output for the lineprinter or *pr* (1). The line length is **nnn** (default is 132). The screen fascilities are replaced by underlines and backspaces for the directories and a star (*) is inserted before executables. On every page (ended with a formfeed) a header with the current possition in the tree is added.

-t Force screen output to be made even if the standard output is not connected to a terminal.

FILES

/etc/termcap terminal info

SEE ALSO

find(1), termlib(3U), termcap(5U)

BUGS

A new capability (sc) had to be added to the termcap file.

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
        command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tset, reset – set terminal modes

SYNOPSIS

tset [options] [**–m** [ident][test baudrate]:type] ... [type]

reset ...

DESCRIPTION

Tset sets up your terminal when you first log in to a UNIX system. It does terminal dependent processing such as setting erase and kill characters, setting or resetting delays, and the like. It first determines the *type* of terminal involved, and then does necessary initializations and mode settings. The type of terminal attached to each UNIX port is specified in the */etc/ttytype* database. Type names for terminals may be found in the */etc/termcap* database. If a port is not wired permanently to a specific terminal (not hardwired) it will be given an appropriate generic identifier such as *dialup*.

In the case where no arguments are specified, *tset* simply reads the terminal type out of the environment variable TERM and reinitializes the terminal. The rest of this manual concerns itself with mode and environment initialization, typically done once at login, and options used at initialization time to determine the terminal type and set up terminal modes.

When used in a startup script (*.profile* for *sh*(1) users or *.login* for *csh*(1) users) it is desirable to give information about the type of terminal you will usually use on ports which are not hardwired. These ports are identified in */etc/ttytype* as *dialup* or *plugboard* or *arpanet*, etc. To specify what terminal type you usually use on these ports, the **–m** (map) option flag is followed by the appropriate port type identifier, an optional baud rate specification, and the terminal type. (The effect is to 'map' from some conditions to a terminal type, that is, to tell *tset* 'If I'm on this kind of port, guess that I'm on that kind of terminal'.) If more than one mapping is specified, the first applicable mapping prevails. A missing port type identifier matches all identifiers. Any of the alternate generic names given in *termcap* may be used for the identifier.

A *baudrate* is specified as with *stty*(1), and is compared with the speed of the diagnostic output (which should be the control terminal). The baud rate *test* may be any combination of: >, @, <, and !; @ means 'at' and ! inverts the sense of the test. To avoid problems with metacharacters, it is best to place the entire argument to **–m** within '' characters; users of *csh*(1) must also put a '\' before any '!' used here.

Thus

tset –m 'dialup>300:adm3a' –m dialup:dw2 –m 'plugboard:?adm3a'

causes the terminal type to be set to an *adm3a* if the port in use is a dialup at a speed greater than 300 baud; to a *dw2* if the port is (otherwise) a dialup (i.e. at 300 baud or less). If the *type* finally determined by *tset* begins with a question mark, the user is asked if s/he really wants that type. A null response means to use that type; otherwise, another type can be entered which will be used instead. Thus, in the above case, the user will be queried on a plugboard port as to whether they are actually using an *adm3a*.

If no mapping applies and a final *type* option, not preceded by a **–m**, is given on the command line then that type is used; otherwise the identifier found in the */etc/ttytype* database will be taken to be the terminal type. This should always be the case for hardwired ports.

It is usually desirable to return the terminal type, as finally determined by *tset*, and information about the terminal's capabilities to a shell's environment. This can be done using the **–s** option; using the Bourne shell, *sh*(1):

eval `tset -s *options...*`

or using the C shell, *csh*(1):

set noglob; eval `tset -s *options...*`

If you have an old csh with no eval command, do the following:

tset -s options > /tmp/tset$$; source /tmp/tset$$; rm /tmp/tset$$

In either *csh* case, you should probably make an alias in your .cshrc:

alias tset 'set noglob; eval `tset -s *`'

or

alias tset 'tset -s * > /tmp/tset$$; source /tmp/tset$$; rm /tmp/tset$$'

Either of these aliases allow the command

tset 2621

to be invoked at any time from your login csh. **Note to Bourne Shell users:** It is *not* possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. (If a process could set its parent's environment, none of this nonsense would be necessary in the first place.)

These commands cause *tset* to generate as output a sequence of shell commands which place the variables TERM and TERMCAP in the environment; see *environ*(5).

Once the terminal type is known, *tset* engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('#' on standard systems), the erase character is changed to DELETE.

The options are:

-e*c* set the erase character to be the named character *c* on all terminals, the default being the backspace character on the terminal, in this case DELETE. The character *c* can either be typed directly, or entered using the hat notation used here.

-k*c* is similar to **-e** but for the line kill character rather than the erase character; *c* defaults to ^X (for purely historical reasons). The kill character is left alone if **-k** is not specified. The hat notation can also be used for this option.

-n On systems with the new tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. See newtty(4) for more detail.

-v On systems with CB/UNIX virtual terminals, normally *tset* sets up the tty driver modes and the environment TERMCAP variable to use the virtual terminal. The **-v** option turns off this processing, causing tset to not use a virtual terminal.

-I suppresses transmitting terminal initialization strings.

-Q suppresses printing the 'Erase set to' and 'Kill set to' messages.

−S If your shell is *csh*, *tset* will output the strings to be assigned to TERM and TERMCAP in the environment rather than commands for a shell. This is mainly useful to avoid the (slow) source command with an old *csh*.

If *tset* is invoked as *reset*, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or '−1' is reset to it's default value.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type '<LF>reset<LF>' to get it to work since <CR> may not work in this state. Often none of this will echo.

EXAMPLES

These examples all assume the Bourne shell and use the −s option. If you use *csh*, use one of the variations described above. Note that a typical use of *tset* in a .profile or .login will also use the **−e** and **−k** options, and often the **−n** or **−Q** options as well. These options have not been included here to keep the examples small. (**NOTE:** some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real *tset* commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are *always* on a 2621.

```
eval `tset −s 2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttytype.

```
eval `tset −s −m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone elses terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
eval `tset −s −m 'switch>1200:?vt100' −m 'switch<=1200:2621'`
```

All of the above entries will fall back on the terminal type specified in /etc/ttytype if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
eval `tset −s ?adm3a`
```

If the file /etc/ttytype is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
eval `tset −s −m '>1200:vt100' 2621`
```

Here is a fancy example to illustrate the power of *tset* and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds

higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in /etc/ttytype. You want your erase character set to control H, your kill character set to control U, and don't want *tset* to print the "Erase set to Backspace, Kill set to Control U" message.

```
eval `tset -e -k^U -Q -s -m 'switch<=1200:concept100' -m 'switch:?vt100'
        -m dialup:concept100 -m arpanet:dm2500`
```

FILES

/etc/ttytype	port name to terminal type mapping database
/etc/termcap	terminal capability database

SEE ALSO

csh(1), sh(1), stty(1), environ(5), termcap(5), ttytype(5)

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

RESTRICTIONS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

tty – get terminal name

SYNOPSIS

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

NAME

ul – do underlining

SYNOPSIS

ul [**–i**] [**–t** *terminal*] [*name ...*]

DESCRIPTION

Ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The **–t** option overrides the terminal kind specified in the environment. The file */etc/termcap* is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, *ul* degenerates to *cat*(1). If the terminal cannot underline, underlining is ignored.

The **–i** option causes *ul* to indicate underlining onto by a separate line containing appropriate dashes '–'; this is useful when you want to look at the underlining which is present in an *nroff* output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

AUTHOR

Mark Horton wrote *ul*. The **–i** option was originally an option of the editor *ex*(1), then an *iul* command.

RESTRICTIONS

Nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [–rSID] [–s] [–n] files

DESCRIPTION

Unget undoes the effect of a **get –e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. Keyletter arguments apply independently to each named file.

–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.

SEE ALSO

delta(1), get(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq − report repeated lines in a file

SYNOPSIS

uniq [−**udc** [+n] [−n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the −**u** flag is used, just the lines that are not repeated in the original file are output. The −**d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the −**u** and −**d** mode outputs.

The −**c** option supersedes −**u** and −**d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.54000e+ 00
        / 3.93701e- 01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 pounds force/in2
You want: atm
        * 1.02069e+ 00
        / 9.79730e- 01
```

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

RESTRICTIONS

Don't base your financial plans on the currency conversions.

NAME

users – compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format.

FILES

/etc/utmp

SEE ALSO

finger(1), who(1)

NAME

uucp, uulog – unix to unix copy

SYNOPSIS

uucp [option] ... source-file ... destination-file

uulog [option] ...

DESCRIPTION

Uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form

system-name!pathname

where 'system-name' is taken from a list of system names which *uucp* knows about. Shell metacharacters ?*[] appearing in the pathname part will be expanded on the appropriate system.

Pathnames may be one of

(1) a full pathname;

(2) a pathname preceded by ~*user*; where *user* is a userid on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*.

–d Make all necessary directories for the file copy.

–c Use the source file when copying out rather than copying the file to the spool directory.

–m Send mail to the requester when the copy is complete.

Uulog maintains a summary log of *uucp* and *uux*(1) transactions in the file '/usr/spool/uucp/LOGFILE' by gathering information from partial log files named '/usr/spool/uucp/LOG.*.?'. It removes the partial log files.

The options cause *uulog* to print logging information:

–s*sys* Print information about work involving system *sys*.

–u*user*
Print information about work done for the specified *user*.

FILES

/usr/spool/uucp - spool directory
/usr/lib/uucp/* - other data and program files

SEE ALSO

uux(1), mail(1)
D. A. Nowitz, *Uucp Implementation Description*

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames.

RESTRICTIONS

All files received by *uucp* will be owned by uucp.

The −m option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters ?*[] will not activate the −m option.)

NAME

uuq – display queue for uucp and uux

SYNOPSIS

uuq

DESCRIPTION

Uuq displays the current queue for files to be sent over with uucp. In case of uux it also prints the issued command. Furthermore it gives information on the status of the systems known to the system you are working on, and on the file that is currently being sent or received.

SEE ALSO

uucp(1), uux(1)

AUTHOR

N. Plat

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [option] ...

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:

–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.

–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 he is the super-user.

–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.

–j*all* Report the status of all the *uucp* requests.

–v Report the *uucp* status verbosely. If this option is not specified, a status code is printed with each *uucp* request.

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**, or the rest of other options may be specified.

For example, the command

uustat –uhdc –smhtsa –y72 –v

will print the verbose 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
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad *uucp* command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file
00400	cannot execute *uucp*
01000	copy succeeded
02000	copy finished, job deleted
04000	job is queued

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).
Uustat – A UUCP Status Inquiry Program, by H. Che.

NAME

uuto, uupick – public UNIX-to-UNIX file copy

SYNOPSIS

uuto [options] source-files destination
uupick [**–s** system]

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*(1C)). *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.*

FILES

PUBDIR	/usr/spool/uucppublic	public directory

SEE ALSO

mail(1), uucp(1C), uulog(1C), uuname(1C), uustat(1C), uux(1C).

NAME

uux − unix to unix command execution

SYNOPSIS

uux [−] 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.

The − option will cause the standard input to the *uux* command to be the standard input to the *command-string*. For example, the command

```
uux "!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !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.

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. For example, the command

```
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".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES

/usr/lib/uucp/spool	spool directory
/usr/lib/uucp/*	other data and programs

SEE ALSO

uuclean(1M) (only on-line, not in this manual), uucp(1C).
Uucp Implementation Description by D. A. Nowitz

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.

NAME

val – validate SCCS file

SYNOPSIS

val –
val [**–s**] [**–r**SID] [**–m**name] [**–y**type] 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., r1.0 or r1.1.0 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 = can't open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **–y** mismatch;
bit 7 = %M%, **–m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc – version control

SYNOPSIS

vc [**–a**] [**–t**] [**–c***char*] [**–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.

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 – normal
1 – any error

NAME

vi − screen oriented (visual) display editor based on ex

SYNOPSIS

vi [**−r**] [**+** *command*] [**−l**] [**−w***n*] name ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on *ex*(1). *Ex* and *vi* run the same code; it is possible to get to the command mode of *ex* from within *vi* and vice-versa.

The *ULTRIX-11 Programmer's Guide* and the *Introduction to Display Editing with Vi* provide full details on using *vi*.

FILES

See *ex*(1).

SEE ALSO

ex (1), edit (1), "ULTRIX-11 Programmer's Guide", "An Introduction to Display Editing with Vi".

AUTHOR

William Joy
Mark Horton added macros to *visual* mode and is maintaining version 3

RESTRICTIONS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as **:source**; there is no way to use the **:append**, **:change**, and **:insert** commands, since it is not possible to give more than one line of input to a **:** escape. To use these on a **:global** you must **Q** to *ex* command mode, execute them, and then reenter the screen editor with *vi* or *open*.

NAME

wait – await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1)

RESTRICTIONS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for.

NAME

wall – write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be superuser to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

wc – word count

SYNOPSIS

wc [−l] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines. All other characters are simply ignored.

The *−l* flag suppresses all output exept the line count.

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters **l**, **w**, or **c**.

NAME

what – identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, newline, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

What is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

RESTRICTIONS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

who – who is on the system

SYNOPSIS

who [who-file] [**am I**]

DESCRIPTION

Who, without an argument, lists the login name, terminal name, and login time for each current UNIX user.

Without an argument, *who* examines the /etc/utmp file to obtain its information. If a file is given, that file is examined. Typically the given file will be /usr/adm/wtmp, which contains a record of all the logins since it was created. Then *who* lists logins, logouts, and crashes since the creation of the wtmp file. Each login is listed with user name, terminal name (with '/dev/' suppressed), and date and time. When an argument is given, logouts produce a similar line without a user name. Reboots produce a line with 'x' in the place of the device name, and a fossil time indicative of when the system went down.

With two arguments, as in 'who am I' (and also 'who are you'), *who* tells who you are logged in as.

FILES

/etc/utmp

SEE ALSO

getuid(2), utmp(5)

NAME

whoami – print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd

SEE ALSO

who (1)

AUTHOR

Bill Joy

NAME

write – write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname yourttyname...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal—(**o**) for 'over' is conventional—that the other may reply. (**oo**) for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1)

NAME

xstr − extract strings from C programs to implement shared strings

SYNOPSIS

xstr [**−c**] [−] [file]

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr −c name

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

xstr

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

xstr name

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '−' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc −E name.c | xstr −c −
cc −c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

strings	Data base of strings
x.c	massaged C source
xs.c	C source for definition of array 'xstr'
/tmp/xs*	temporary file for 'xstr name' *strings*

SEE ALSO

mkstr(1)

AUTHOR

Bill Joy

RESTRICTIONS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc − yet another compiler-compiler

SYNOPSIS

yacc [−**vd**] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *Lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the −**v** flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the −**d** flag is used, the file *y.tab.h* is generated with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names'. This allows source files other than *y.tab.c* to access the token codes.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs
/lib/liby.a	library with default 'main' and 'yyerror'

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC − Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

RESTRICTIONS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

NAME

yacc_prep – pre-process some attributes for YACC

SYNOPSIS

yacc_prep input_file output_file

DESCRIPTION

Yacc_prep is a simple preprocessor for use with *yacc* , a description of its functioning and use is to be found in SIGPLAN NOTICES October 1983.

SEE ALSO

yacc(1)

BUGS

Who knows

AUTHOR

J van Katwijk

NAME

zaptty – run a program without a controlling terminal.

SYNOPSIS

zaptty command

DESCRIPTION

Zaptty will disassociate itself from it's controlling terminal, and then exec the specified command. This is useful for re-starting daemons that have died for some reason, and don't want to be associated with any terminal when they are re-started.

Zaptty is restricted to the superuser.

SEE ALSO

zaptty(2), nohup(1)