RISC/os (UMIPS) User's Reference Manual Volume I (System V)

Order Number 3204DOC

mips

The power of RISC is in the system.

RISC/os (UMIPS) User's Reference Manual Volume I (System V)

Order Number 3204DOC

March 1989

Your comments on our products and publications are welcome. A postage–paid form is provided for this purpose on the last page of this manual.

Mfg. Part Number 02–00130–002/84–00133–002

MIPS Computer Systems, Inc.
930 Arques Ave.
Sunnyvale, CA 94086

Customer Service Telephone Numbers:

California:	(800)	992–MIPS
All other states:	(800)	443–MIPS
International:	(415)	330–7966

Mfg. Part Number 02–00130–002/84–00133–002

TABLE OF CONTENTS

1. Commands and Application Programs

PERMUTED INDEX

on the target board cross a.run	: download and execute a program	a.run(1)
on the target board cross a.run	: download and execute a program	a.run(1)
object file odump	: dumps selected parts of an	odump(1)
echo	: echo arguments	echo(1)
enable, disable	: enable disable LP printers	enable(1)
crypt	: encode decode	crypt(1)
transmission uuencode,uudecode	: encode decode a binary file for	uuencode(1)
and history mechanism for a.view	: establish command abbreviations	a.view(1)
and history mechanism for a.view	: establish command abbreviations	a.view(1)
expression expr	: evaluate arguments as an	expr(1)
time at, batch	: execute commands at a later	at(1)
to implement shared strings xstr	: extract strings from C programs	xstr(1)
less	: file browser	less(1)
pg	: file perusal filter for CRTs	pg(1)
viewing more, page	: file perusal filter for crt	more(1)
col	: filter reverse line feeds	col(1)
find	: find files	find(1)
object library lorder	: find ordering relation for an	lorder(1)
hashmake, spellin, hashcheck	: find spelling errors spell,	spell(1)
an object or other strings	: find the printable strings in	strings(1)
width output device fold	: fold long lines for finite	fold(1)
a.pr	: format source code	a.pr(1)
a.pr	: format source code	a.pr(1)
cflow	: generate C flowgraph	cflow(1)
cross-reference cxref	: generate C program	cxref(1)
makekey	: generate encryption key	makekey(1)
lexical tasks lex	: generate programs for simple	lex(1)
file rls_id	: generate release identification	rls_id(1)
get	: get a version of an SCCS file	get(1)
logname	: get login name	logname(1)
mips, pdp11, u3b, u3b2, u3b5, vax	: get processor type truth value	machid(1)
tty	: get the name of the terminal	tty(1)
multivol	: handle multivolume files	multivol(1)
what	: identify SCCS files	what(1)
using keywords locate	: identify a UNIX system command	locate(1)
ident	: identify files	ident(1)
system for beginning starter	: information about the UNIX	starter(1)
terminfo database tput	: initialize a terminal or query	tput(1)
user setup	: initialize system for first	setup(1)
a.help	: interactive help utility	a.help(1)
a.help	: interactive help utility	a.help(1)
system mailx	: interactive message processing	mailx(1)
ftoc	: interface between prof and cord	ftoc(1)
rcsintro	: introduction to RCS commands	rcsintro(1)
application programs intro	: introduction to commands and	intro(1)
nl	: line numbering filter	nl(1)
a.ls	: list compiled programs	a.ls(1)
a.ls	: list compiled programs	a.ls(1)
ls	: list contents of directory	ls(1)
options a.info	: list or change VADS library	a.info(1)
options a.info	: list or change VADS library	a.info(1)
m4	: macro processor	m4(1)
program mt	: magnetic tape manipulating	mt(1)
regenerate groups of make	: maintain, update, and	make(1)
SCCS file delta	: make a delta (change) to an	delta(1)
mkdir	: make directories	mkdir(1)
a.mklib	: make library directory	a.mklib(1)
a.mklib	: make library directory	a.mklib(1)
banner	: make posters	banner(1)
session script	: make typescript of terminal	script(1)
administration sysadm	: menu interface to do system	sysadm(1)
rcsmerge	: merge RCS revisions	rcsmerge(1)
files or subsequent lines paste	: merge same lines of several	paste(1)
files nm	: name list dump of MIPS object	nm(1)
information uname	: obtain current system	uname(1)
number factor	: obtain the prime factors of a	factor(1)
od	: octal dump	od(1)
getopt	: parse command options	getopt(1)
getopts, getoptcvt	: parse command options	getopts(1)
language awk	: pattern scanning and processing	awk(1)
mesg	: permit or deny messages	mesg(1)
tee	: pipe fitting	tee(1)
a.ld	: prelinker	a.ld(1)
a.ld	: prelinker	a.ld(1)
lpstat	: print LP status information	lpstat(1)
prs	: print an SCCS file	prs(1)

interval sleep	: suspend execution for an	sleep(1)
sar	: system activity reporter	sar(1)
tar	: tape archiver	tar(1)
kill	: terminate a process	kill(1)
havetcp	: test system for TCP support	havetcp(1)
ed, red	: text editor	ed(1)
ex	: text editor	ex(1)
casual users) edit	: text editor (variant of ex for	edit(1)
cpp	: the C language preprocessor	cpp(1)
merge	: three-way file merge	merge(1)
time	: time a command	time(1)
data and system activity timex	: time a command; report process	timex(1)
tsort	: topological sort	tsort(1)
tr	: translate characters	tr(1)
tftp	: trivial file transfer program	tftp(1)
file unget	: undo a previous get of an SCCS	unget(1)
times of a file touch	: update access and modification	touch(1)
crontab	: user crontab file	crontab(1)
protocol telnet	: user interface to the TELNET	telnet(1c)
control uustat	: uucp status inquiry and job	uustat(1)
val	: validate SCCS file	val(1)
vc	: version control	vc(1)
man	: view online manual pages	man(1)
vsar	: visual system activity reporter	vsar(1)
who	: who is on the system	who(1)
machines rwho	: who's logged in on local	rwho(1c)
wc	: word count	wc(1)
pwd	: working directory name	pwd(1)
wall	: write to all users	wall(1)
network rwall	: write to all users over a	rwall(1)
write	: write to another user	write(1)
yacc	: yet another compiler-compiler	yacc(1)
comparison diff3 :	3-way differential file	diff3(1)
: MIPS Fortran	77 compiler	f77
ftp :	ARPANT file transfer program	ftp(1c)
cc : MIPS	C compiler	cc(1)
cflow : generate	C flowgraph	cflow(1)
cpp : the	C language preprocessor	cpp(1)
cb :	C program beautifier	cb(1)
lint : a	C program checker	lint(1)
cxref : generate	C program cross-reference	cxref(1)
ctrace :	C program debugger	ctrace(1)
xstr : extract strings from	C programs to implement shared	xstr(1)
and history mechanism for	C shell command abbreviations	a.view(1)
and history mechanism for	C shell command abbreviations	a.view(1)
error message file by massaging	C source mkstr : create an	mkstr(1)
object file list : produce	C source listing from a common	list(1)
: MIPS	COBOL compiler	cobol
pg : file perusal filter for	CRTs	pg(1)
shell (command interpreter) with	C-like syntax csh : a	csh(1)
help : UNIX system Help	Facility	help(1)
: MIPS	Fortran 77 compiler	f77
help : UNIX system	Help Facility	help(1)
: send cancel requests to an	LP line printer lp, cancel	lp(1)
enable, disable : enable disable	LP printers	enable(1)
lpstat : print	LP status information	lpstat(1)
cc :	MIPS C compiler	cc(1)
:	MIPS COBOL compiler	cobol
:	MIPS Fortran 77 compiler	f77
pl1 :	MIPS PL I compiler	pl(1)
pc :	MIPS Pascal compiler	pc(1)
as :	MIPS assembler	as(1)
ld :	MIPS link editor	ld(1)
nm : name list dump of	MIPS object files	nm(1)
pl1 : MIPS	PL I compiler	pl(1)
pc : MIPS	Pascal compiler	pc(1)
rcsintro : introduction to	RCS commands	rcsintro(1)
rcs : change	RCS file attributes	rcs(1)
and other information about	RCS files : print log messages	rlog(1)
ci : check in	RCS revisions	ci(1)
co : check out	RCS revisions	co(1)
rcsdiff : compare	RCS revisions	rcsdiff(1)
rcsmerge : merge	RCS revisions	rcsmerge(1)
change the delta commentary of an	SCCS delta cdc:	cdc(1)
comb : combine	SCCS deltas	comb(1)
: make a delta (change) to an	SCCS file delta	delta(1)

ci :	check in RCS revisions	ci(1)
co :	check out RCS revisions	co(1)
lint : a C program	checker	lint(1)
file sum : print	checksum and block count of a	sum(1)
chown,	chgrp : change owner or group	chown(1)
	chmod : change mode	chmod(1)
group	chown, chgrp : change owner or	chown(1)
	ci : check in RCS revisions	ci(1)
	cmp : compare two files	cmp(1)
	co : check out RCS revisions	co(1)
a.pr : format source	code	a.pr(1)
a.pr : format source	code	a.pr(1)
pixie : add profiling	code to a program	pixie(1)
	col : filter reverse line feeds	col(1)
	comb : combine SCCS deltas	comb(1)
comb :	combine SCCS deltas	comb(1)
common to two sorted files	comm : select or reject lines	comm(1)
test : condition evaluation	command	test(1)
time : time a	command	time(1)
argument list(s) and execute	command xargs : construct	xargs(1)
mechanism for a.view : establish	command abbreviations and history	a.view(1)
mechanism for a.view : establish	command abbreviations and history	a.view(1)
nice : run a	command at low priority	nice(1)
examples usage : retrieve a	command description and usage	usage(1)
env : set environment for	command execution	env(1)
uux : UNIX-to-UNIX system	command execution	uux(1)
quits nohup : run a	command immune to hangups and	nohup(1)
syntax csh : a shell	(command interpreter) with C-like	csh(1)
getopt : parse	command options	getopt(1)
getopts, getoptcvt : parse	command options	getopts(1)
: shell, the standard restricted	command programming language rsh	sh(1)
system activity timex : time a	command; report process data and	timex(1)
locate : identify a UNIX system	command using keywords	locate(1)
rcsintro : introduction to RCS	commands	rcsintro(1)
intro : introduction to	commands and application programs	intro(1)
at, batch : execute	commands at a later time	at(1)
cdc: change the delta	commentary of an SCCS delta	cdc(1)
glossary : definitions of	common UNIX system terms and	glossary(1)
: produce C source listing from a	common object file list	list(1)
comm : select or reject lines	common to two sorted files	comm(1)
ipcs : report inter-process	communication facilities status	ipcs(1)
diff : differential file	comparator	diff(1)
rcsdiff :	compare RCS revisions	rcsdiff(1)
cmp :	compare two files	cmp(1)
file sccsdiff :	compare two versions of an SCCS	sccsdiff(1)
diff3 : 3-way differential file	comparison	diff3(1)
dircmp : directory	comparison	dircmp(1)
a.rmlib : remove	compilation library	a.rmlib(1)
a.rmlib : remove	compilation library	a.rmlib(1)
regcmp : regular expression	compile	regcmp(1)
a.ls : list	compiled programs	a.ls(1)
a.ls : list	compiled programs	a.ls(1)
cc : MIPS C	compiler	cc(1)
: MIPS COBOL	compiler	cobol
: MIPS Fortran 77	compiler	f77
pc : MIPS Pascal	compiler	pc(1)
pl1 : MIPS PL I	compiler	pl(1)
yacc : yet another	compiler-compiler	yacc(1)
program on the target board cross	compilers download and execute a	a.run(1)
program on the target board cross	compilers download and execute a	a.run(1)
wait : await	completion of process	wait(1)
pack, pcat, unpack :	compress and expand files	pack(1)
cat :	concatenate and print files	cat(1)
test :	condition evaluation command	test(1)
execute command xargs :	construct argument list(s) and	xargs(1)
remove nroff troff, tbl, and eqn	constructs deroff :	deroff(1)
: determine which project library	contains a unit a.which	a.which(1)
: determine which project library	contains a unit a.which	a.which(1)
ls : list	contents of directory	ls(1)
csplit :	context split	csplit(1)
: uucp status inquiry and job	control uustat	uustat(1)
vc : version	control	vc(1)
units :	conversion program	units(1)
libraries ranlib :	convert archives to random	ranlib(1)
rcp : remote file	copy	rcp(1c)
uuname : UNIX-to-UNIX system	copy uucp, uulog,	uucp(1)

: public UNIX-to-UNIX system file	copy uuto, uupick	uuto(1)
cpio :	copy file archives in and out	cpio(1)
cp, ln, mv :	copy, link or move files	cp(1)
ftoc : interface between prof and	cord	ftoc(1)
an executable file to facilitate	cord : rearranges procedures in	cord(1)
in an executable file to	cord2 : rearranges basic blocks	cord2(1)
wc : word	count	wc(1)
sum : print checksum and block	count of a file	sum(1)
files	cp, ln, mv : copy, link or move	cp(1)
out	cpio : copy file archives in and	cpio(1)
	cpp : the C language preprocessor	cpp(1)
available VADS versions and	create a default library display	a.vadsrc(1)
available VADS versions and	create a default library display	a.vadsrc(1)
mkshlib :	create a shared library	mkshlib(1)
a.tags :	create a tags file	a.tags(1)
a.tags :	create a tags file	a.tags(1)
ctags :	create a tags file	ctags(1)
massaging C source mkstr :	create an error message file by	mkstr(1)
admin :	create and administer SCCS files	admin(1)
	crontab : user crontab file	crontab(1)
crontab : user	crontab file	crontab(1)
a program on the target board	cross compilers and execute	a.run(1)
a program on the target board	cross compilers and execute	a.run(1)
cxref : generate C program	cross-reference	cxref(1)
page : file perusal filter for	crt viewing more,	more(1)
	crypt : encode decode	crypt(1)
interpreter) with C-like syntax	csh : a shell (command	csh(1)
	csplit : context split	csplit(1)
terminal	ct : spawn getty to a remote	ct(1)
	ctags : create a tags file	ctags(1)
	ctrace : C program debugger	ctrace(1)
or UNIX system	cu : call another UMIPS-V system	cu(1)
activity sact : print	current SCCS file editing	sact(1)
: set or display name of	current domain system domainname	domainname(1)
. : set or print identifier of	current host system hostid	hostid(1)
hostname : set or print name of	current host system	hostname(1)
uname : obtain	current system information	uname(1)
each line of a file	cut : cut out selected fields of	cut(1)
line of a file cut :	cut out selected fields of each	cut(1)
cross-reference	cxref : generate C program	cxref(1)
: display call graph profile	data gprof	gprof(1)
prof : analyze profile	data	prof(1)
: time a command; report process	data and system activity timex	timex(1)
file newaliases : rebuild the	data base for the mail aliases	newaliases(1)
a terminal or query terminfo	database tput : initialize	tput(1)
join : relational	database operator	join(1)
date : print and set the	date	date(1)
	date : print and set the date	date(1)
	dbx : source level debugger	dbx(1)
	dc : desk calculator	dc(1)
a.db : source level	debugger	a.db(1)
a.db : source level	debugger	a.db(1)
ctrace : C program	debugger	ctrace(1)
dbx : source level	debugger	dbx(1)
crypt : encode	decode	crypt(1)
uuencode,uudecode : encode	decode a binary file for	uuencode(1)
VADS versions and create a	default library available	a.vadsrc(1)
VADS versions and create a	default library available	a.vadsrc(1)
terms and symbols glossary :	definitions of common UNIX system	glossary(1)
tail :	deliver the last part of a file	tail(1)
the delta commentary of an SCCS	delta cdc: change	cdc(1)
an SCCS file	delta : make a delta (change) to	delta(1)
delta : make a	delta (change) to an SCCS file	delta(1)
cdc: change the	delta commentary of an SCCS delta	cdc(1)
rmdel : remove a	delta from an SCCS file	rmdel(1)
comb : combine SCCS	deltas	comb(1)
mesg : permit or	deny messages	mesg(1)
: recompile source files in	dependency order a.make	a.make(1)
: recompile source files in	dependency order a.make	a.make(1)
and eqn constructs	deroff : remove nroff troff, tbl,	deroff(1)
usage : retrieve a command	description and usage examples	usage(1)
dc :	desk calculator	dc(1)
file :	determine file type	file(1)
contains a unit a.which :	determine which project library	a.which(1)
contains a unit a.which :	determine which project library	a.which(1)
lines for finite width output	device fold : fold long	fold(1)

reject lines common to two sorted	files comm : select or	comm(1)
cp, ln, mv : copy, link or move	files	cp(1)
find : find	files	find(1)
ident : identify	files	ident(1)
multivol : handle multivolume	files	multivol(1)
: name list dump of MIPS object	files nm	nm(1)
unpack : compress and expand	files pack, pcat,	pack(1)
pr : print	files	pr(1)
and other information about RCS	files rlog : print log messages	rlog(1)
sort : sort and or merge	files	sort(1)
what : identify SCCS	files	what(1)
a.make : recompile source	files in dependency order	a.make(1)
a.make : recompile source	files in dependency order	a.make(1)
rm, rmdir : remove	files or directories	rm(1)
: merge same lines of several	files or subsequent lines of one	paste(1)
nl : line numbering	filter	nl(1)
pg : file perusal	filter for CRTs	pg(1)
more, page : file perusal	filter for crt viewing	more(1)
col :	filter reverse line feeds	col(1)
	find : find files	find(1)
find :	find files	find(1)
object library lorder :	find ordering relation for an	lorder(1)
hashmake, spellin, hashcheck :	find spelling errors spell,	spell(1)
object or other binary strings :	find the printable strings in an	strings(1)
fold : fold long lines for	finite width output device	fold(1)
tee : pipe	fitting	tee(1)
cflow : generate C	flowgraph	cflow(1)
width output device	fold : fold long lines for finite	fold(1)
output device fold :	fold long lines for finite width	fold(1)
: archive (library) file	format	ar
: archive (library) file	format	xar
newform : change the	format of a text file	newform(1)
a.pr :	format source code	a.pr(1)
a.pr :	format source code	a.pr(1)
cord	ftoc : interface between prof and	ftoc(1)
program	ftp : ARPANT file transfer	ftp(1c)
search a file for a pattern using	full regular expressions egrep :	egrep(1)
cflow :	generate C flowgraph	cflow(1)
cross-reference cxref :	generate C program	cxref(1)
makekey :	generate encryption key	makekey(1)
lexical tasks lex :	generate programs for simple	lex(1)
file rls_id :	generate release identification	rls_id(1)
	getopt : parse command options	getopt(1)
getopts,	getoptcvt : parse command options	getopts(1)
command options	getopts, getoptcvt : parse	getopts(1)
ct : spawn	getty to a remote terminal	ct(1)
UNIX system terms and symbols	glossary : definitions of common	glossary(1)
profile data	gprof : display call graph	gprof(1)
gprof : display call	graph profile data	gprof(1)
pattern	grep : search a file for a	grep(1)
chown, chgrp : change owner or	group	chown(1)
maintain, update, and regenerate	groups of programs make :	make(1)
multivol :	handle multivolume files	multivol(1)
nohup : run a command immune to	hangups and quits	nohup(1)
spell, hashmake, spellin,	hashcheck : find spelling errors	spell(1)
find spelling errors spell,	hashmake, spellin, hashcheck :	spell(1)
support	havetcp : test system for TCP	havetcp(1)
	help : UNIX system Help Facility	help(1)
a.help : interactive	help utility	a.help(1)
a.help : interactive	help utility	a.help(1)
command abbreviations and	history mechanism for C shell	a.view(1)
command abbreviations and	history mechanism for C shell	a.view(1)
ruptime : show	host status of local machines	ruptime(1c)
or print identifier of current	host system hostid : set	hostid(1)
: set or print name of current	host system hostname	hostname(1)
of current host system	hostid : set or print identifier	hostid(1)
current host system	hostname : set or print name of	hostname(1)
semaphore set or shared memory	id : remove a message queue,	ipcrm(1)
su, ssu : substitute user	id temporarily	su(1)
	ident : identify files	ident(1)
rls_id : generate release	identification file	rls_id(1)
hostid : set or print	identifier of current host system	hostid(1)
what :	identify SCCS files	what(1)
using keywords locate :	identify a UNIX system command	locate(1)
ident :	identify files	ident(1)
nohup : run a command	immune to hangups and quits	nohup(1)

strings from C programs to	implement shared strings extract	xstr(1)
: remove source unit and library	information a.rm	a.rm(1)
: remove source unit and library	information a.rm	a.rm(1)
lpstat : print LP status	information	lpstat(1)
uname : obtain current system	information	uname(1)
: print log messages and other	information about RCS files rlog	rlog(1)
for beginning users starter :	information about the UNIX system	starter(1)
terminfo database tput :	initialize a terminal or query	tput(1)
setup :	initialize system for first user	setup(1)
uustat : uucp status	inquiry and job control	uustat(1)
a.help :	interactive help utility	a.help(1)
a.help :	interactive help utility	a.help(1)
system mailx :	interactive message processing	mailx(1)
ftoc :	interface between prof and cord	ftoc(1)
administration sysadm : menu	interface to do system	sysadm(1)
telnet : user	interface to the TELNET protocol	telnet(1c)
csh : a shell (command	interpreter) with C-like syntax	csh(1)
facilities status ipcs : report	inter-process communication	ipcs(1)
sleep : suspend execution for an	interval	sleep(1)
and application programs	intro : introduction to commands	intro(1)
rcsintro :	introduction to RCS commands	rcsintro(1)
application programs intro :	introduction to commands and	intro(1)
semaphore set or shared memory	ipcrm : remove a message queue,	ipcrm(1)
communication facilities status	ipcs : report inter-process	ipcs(1)
news : print news	items	news(1)
uustat : uucp status inquiry and	job control	uustat(1)
operator	join : relational database	join(1)
makekey : generate encryption	key	makekey(1)
a UNIX system command using	keywords locate : identify	locate(1)
	kill : terminate a process	kill(1)
: pattern scanning and processing	language awk	awk(1)
: arbitrary-precision arithmetic	language bc	bc(1)
restricted command programming	language : shell, the standard	sh(1)
cpp : the C	language preprocessor	cpp(1)
at, batch : execute commands at a	later time	at(1)
	ld : MIPS link editor	ld(1)
	less : file browser	less(1)
a.db : source	level debugger	a.db(1)
a.db : source	level debugger	a.db(1)
dbx : source	level debugger	dbx(1)
simple lexical tasks	lex : generate programs for	lex(1)
: generate programs for simple	lexical tasks lex	lex(1)
: convert archives to random	libraries ranlib	ranlib(1)
a.rmlib : remove compilation	library	a.rmlib(1)
a.rmlib : remove compilation	library	a.rmlib(1)
versions and create a default	library : display available VADS	a.vadsrc(1)
versions and create a default	library : display available VADS	a.vadsrc(1)
ordering relation for an object	library lorder : find	lorder(1)
mkshlib : create a shared	library	mkshlib(1)
a.which : determine which project	library contains a unit	a.which(1)
a.which : determine which project	library contains a unit	a.which(1)
a.cleanlib : reinitialize	library directory	a.cleanlib(1)
a.cleanlib : reinitialize	library directory	a.cleanlib(1)
a.mklib : make	library directory	a.mklib(1)
a.mklib : make	library directory	a.mklib(1)
a.du : summarize	library disk usage	a.du(1)
a.du : summarize	library disk usage	a.du(1)
: archive	(library) file format	ar
: archive	(library) file format	xar
a.rm : remove source unit and	library information	a.rm(1)
a.rm : remove source unit and	library information	a.rm(1)
ar : archive and	library maintainer	ar(1)
a.info : list or change VADS	library options	a.info(1)
a.info : list or change VADS	library options	a.info(1)
a.pathxs : report or change VADS	library search list	a.path(1)
a.pathxs : report or change VADS	library search list	a.path(1)
line : read one	line	line(1)
	line : read one line	line(1)
col : filter reverse	line feeds	col(1)
nl :	line numbering filter	nl(1)
: produce program listing with	line numbers a.list	a.list(1)
: produce program listing with	line numbers a.list	a.list(1)
: cut out selected fields of each	line of a file cut	cut(1)
: send cancel requests to an LP	line printer lp, cancel	lp(1)
comm : select or reject	lines common to two sorted files	comm(1)
device fold : fold long	lines for finite width output	fold(1)

	mkshlib : create a shared library	mkshlib(1)
file by massaging C source	mkstr : create an error message	mkstr(1)
chmod : change	mode	chmod(1)
umask : set file-creation	mode mask	umask(1)
touch : update access and	modification times of a file	touch(1)
for crt viewing	more, page : file perusal filter	more(1)
cp, ln, mv : copy, link or	move files	cp(1)
program	mt : magnetic tape manipulating	mt(1)
files	multivol : handle multivolume	multivol(1)
multivol : handle	multivolume files	multivol(1)
cp, ln,	mv : copy, link or move files	cp(1)
logname : get login	name	logname(1)
pwd : working directory	name	pwd(1)
files nm :	name list dump of MIPS object	nm(1)
domainname : set or display	name of current domain system	domainname(1)
hostname : set or print	name of current host system	hostname(1)
tty : get the	name of the terminal	tty(1)
	netstat : show network status	netstat(1)
rwall : write to all users over a	network	rwall(1)
netstat : show	network status	netstat(1)
base for the mail aliases file	newaliases : rebuild the data	newaliases(1)
text file	newform : change the format of a	newform(1)
	news : print news items	news(1)
news : print	news items	news(1)
priority	nice : run a command at low	nice(1)
	nl : line numbering filter	nl(1)
object files	nm : name list dump of MIPS	nm(1)
hangups and quits	nohup : run a command immune to	nohup(1)
constructs deroff : remove	nroff troff, tbl, and eqn	deroff(1)
: obtain the prime factors of a	number factor	factor(1)
nl : line	numbering filter	nl(1)
produce program listing with line	numbers a.list :	a.list(1)
produce program listing with line	numbers a.list :	a.list(1)
dis : disassemble an	object file	dis(1)
C source listing from a common	object file list : produce	list(1)
: dumps selected parts of an	object file odump	odump(1)
: prints the section size of an	object file size	size(1)
nm : name list dump of MIPS	object files	nm(1)
: find ordering relation for an	object library lorder	lorder(1)
find the printable strings in an	object or other binary file :	strings(1)
uname :	obtain current system information	uname(1)
number factor :	obtain the prime factors of a	factor(1)
od :	octal dump	od(1)
	od : octal dump	od(1)
an object file	odump : dumps selected parts of	odump(1)
man : view	online manual pages	man(1)
join : relational database	operator	join(1)
: list or change VADS library	options a.info	a.info(1)
: list or change VADS library	options a.info	a.info(1)
getopt : parse command	options	getopt(1)
getoptcvt : parse command	options getopts,	getopts(1)
stty : set the	options for a terminal	stty(1)
source files in dependency	order a.make : recompile	a.make(1)
source files in dependency	order a.make : recompile	a.make(1)
library lorder : find	ordering relation for an object	lorder(1)
for applying a diff file to an	original patch : a program	patch(1)
fold long lines for finite width	output device fold :	fold(1)
chown, chgrp : change	owner or group	chown(1)
expand files	pack, pcat, unpack : compress and	pack(1)
crt viewing more,	page : file perusal filter for	more(1)
man : view online manual	pages	man(1)
getopt :	parse command options	getopt(1)
getopts, getoptcvt :	parse command options	getopts(1)
tail : deliver the last	part of a file	tail(1)
odump : dumps selected	parts of an object file	odump(1)
	passwd : change login password	passwd(1)
passwd : change login	password	passwd(1)
several files or subsequent	paste : merge same lines of	paste(1)
diff file to an original	patch : a program for applying a	patch(1)
grep : search a file for a	pattern	grep(1)
language awk :	pattern scanning and processing	awk(1)
egrep : search a file for a	pattern using full regular	egrep(1)
	pc : MIPS Pascal compiler	pc(1)
expand files pack,	pcat, unpack : compress and	pack(1)
processor type machid: mips,	pdp11, u3b, u3b2, u3b5, vax : get	machid(1)
mesg :	permit or deny messages	mesg(1)

pg : file	perusal filter for CRTs	pg(1)
more, page : file	perusal filter for crt viewing	more(1)
	pg : file perusal filter for CRTs	pg(1)
split : split a file into	pieces	split(1)
tee :	pipe fitting	tee(1)
program	pixie : add profiling code to a	pixie(1)
execution	pixstats : analyze program	pixstats(1)
	pl1 : MIPS PL I compiler	pl(1)
banner : make	posters	banner(1)
	pr : print files	pr(1)
a.ld :	prelinker	a.ld(1)
a.ld :	prelinker	a.ld(1)
cpp : the C language	preprocessor	cpp(1)
unget : undo a	previous get of an SCCS file	unget(1)
factor : obtain the	prime factors of a number	factor(1)
lpstat :	print LP status information	lpstat(1)
prs :	print an SCCS file	prs(1)
date :	print and set the date	date(1)
cal :	print calendar	cal(1)
a file sum :	print checksum and block count of	sum(1)
activity sact :	print current SCCS file editing	sact(1)
cat : concatenate and	print files	cat(1)
pr :	print files	pr(1)
system hostid : set or	print identifier of current host	hostid(1)
information about RCS rlog :	print log messages and other	rlog(1)
hostname : set or	print name of current host system	hostname(1)
news :	print news items	news(1)
printenv :	print out the environment	printenv(1)
mailq :	print sendmail mail queue	mailq(1)
winsize : set	print window size	winsize(1)
other binary strings : find the	printable strings in an object or	strings(1)
environment	printenv : print out the	printenv(1)
cancel requests to an LP line	printer lp, cancel : send	lp(1)
disable : enable disable LP	printers enable,	enable(1)
object file size :	prints the section size of an	size(1)
nice : run a command at low	priority	nice(1)
to facilitate cord : rearranges	procedures in an executable file	cord(1)
kill : terminate a	process	kill(1)
wait : await completion of	process	wait(1)
timex : time a command; report	process data and system activity	timex(1)
ps : report	process status	ps(1)
awk : pattern scanning and	processing language	awk(1)
mailx : interactive message	processing system	mailx(1)
m4 : macro	processor	m4(1)
pdp11, u3b, u3b2, u3b5, vax : get	processor type truth value mips,	machid(1)
common object file list :	produce C source listing from a	list(1)
numbers a.list :	produce program listing with line	a.list(1)
numbers a.list :	produce program listing with line	a.list(1)
	prof : analyze profile data	prof(1)
ftoc : interface between	prof and cord	ftoc(1)
gprof : display call graph	profile data	gprof(1)
prof : analyze	profile data	prof(1)
pixie : add	profiling code to a program	pixie(1)
ftp : ARPANT file transfer	program	ftp(1c)
mt : magnetic tape manipulating	program	mt(1)
pixie : add profiling code to a	program	pixie(1)
sdiff : side-by-side difference	program	sdiff(1)
tftp : trivial file transfer	program	tftp(1)
units : conversion	program	units(1)
cb : C	program beautifier	cb(1)
lint : a C	program checker	lint(1)
cxref : generate C	program cross-reference	cxref(1)
ctrace : C	program debugger	ctrace(1)
pixstats : analyze	program execution	pixstats(1)
to an original patch : a	program for applying a diff file	patch(1)
a.list : produce	program listing with line numbers	a.list(1)
a.list : produce	program listing with line numbers	a.list(1)
a.run : download and execute a	program on the target board cross	a.run(1)
a.run : download and execute a	program on the target board cross	a.run(1)
the standard restricted command	programming language : shell,	sh(1)
a.ls : list compiled	programs	a.ls(1)
a.ls : list compiled	programs	a.ls(1)
to commands and application	programs intro : introduction	intro(1)
update, and regenerate groups of	programs make : maintain,	make(1)
lex : generate	programs for simple lexical tasks	lex(1)
xstr : extract strings from C	programs to implement shared	xstr(1)

a.which : determine which	project library contains a unit	a.which(1)
a.which : determine which	project library contains a unit	a.which(1)
: user interface to the TELNET	protocol telnet	telnet(1c)
true, false :	provide truth values	true(1)
	prs : print an SCCS file	prs(1)
	ps : report process status	ps(1)
copy uuto, uupick :	public UNIX-to-UNIX system file	uuto(1)
	pwd : working directory name	pwd(1)
tput : initialize a terminal or	query terminfo database	tput(1)
mailq : print sendmail mail	queue	mailq(1)
memory ipcrm : remove a message	queue, semaphore set or shared	ipcrm(1)
a command immune to hangups and	quits nohup : run	nohup(1)
ranlib : convert archives to	random libraries	ranlib(1)
random libraries	ranlib : convert archives to	ranlib(1)
	rcp : remote file copy	rcp(1c)
	rcs : change RCS file attributes	rcs(1)
	rcsdiff : compare RCS revisions	rcsdiff(1)
commands	rcsintro : introduction to RCS	rcsintro(1)
	rcsmerge : merge RCS revisions	rcsmerge(1)
vmsbackup :	read a VMS backup tape	vmsbackup(1)
rmail : send mail to users or	read mail mail,	mail(1)
line :	read one line	line(1)
executable file to cord2 :	rearranges basic blocks in an	cord2(1)
executable file to cord :	rearranges procedures in an	cord(1)
mail aliases file newaliases :	rebuild the data base for the	newaliases(1)
dependency order a.make :	recompile source files in	a.make(1)
dependency order a.make :	recompile source files in	a.make(1)
ed,	red : text editor	ed(1)
compile	regcmp : regular expression	regcmp(1)
make : maintain, update, and	regenerate groups of programs	make(1)
regcmp :	regular expression compile	regcmp(1)
a file for a pattern using full	regular expressions : search	egrep(1)
a.cleanlib :	reinitialize library directory	a.cleanlib(1)
a.cleanlib :	reinitialize library directory	a.cleanlib(1)
files comm : select or	reject lines common to two sorted	comm(1)
lorder : find ordering	relation for an object library	lorder(1)
join :	relational database operator	join(1)
rls_id : generate	release identification file	rls_id(1)
strip : remove symbols and	relocation bits	strip(1)
calendar :	reminder service	calendar(1)
rcp :	remote file copy	rcp(1c)
rlogin :	remote login	rlogin(1)
rsh, remsh :	remote shell	rsh(1c)
ct : spawn getty to a	remote terminal	ct(1)
rmdel :	remove a delta from an SCCS file	rmdel(1)
set or shared memory id ipcrm :	remove a message queue, semaphore	ipcrm(1)
a.rmlib :	remove compilation library	a.rmlib(1)
a.rmlib :	remove compilation library	a.rmlib(1)
rm, rmdir :	remove files or directories	rm(1)
constructs deroff :	remove nroff troff, tbl, and eqn	deroff(1)
information a.rm :	remove source unit and library	a.rm(1)
information a.rm :	remove source unit and library	a.rm(1)
bits strip :	remove symbols and relocation	strip(1)
rsh,	remsh : remote shell	rsh(1c)
uniq : report	repeated lines in a file	uniq(1)
communication facilities ipcs :	report inter-process	ipcs(1)
search list a.pathxs :	report or change VADS library	a.path(1)
search list a.pathxs :	report or change VADS library	a.path(1)
activity timex : time a command;	report process data and system	timex(1)
ps :	report process status	ps(1)
uniq :	report repeated lines in a file	uniq(1)
sar : system activity	reporter	sar(1)
vsar : visual system activity	reporter	vsar(1)
lp, cancel : send cancel	requests to an LP line printer	lp(1)
sh, rsh : shell, the standard	restricted command programming	sh(1)
and usage examples usage :	retrieve a command description	usage(1)
col : filter	reverse line feeds	col(1)
ci : check in RCS	revisions	ci(1)
co : check out RCS	revisions	co(1)
rcsdiff : compare RCS	revisions	rcsdiff(1)
rcsmerge : merge RCS	revisions	rcsmerge(1)
other information about RCS	rlog : print log messages and	rlog(1)
	rlogin : remote login	rlogin(1)
identification file	rls_id : generate release	rls_id(1)
directories	rm, rmdir : remove files or	rm(1)
read mail mail,	rmail : send mail to users or	mail(1)

stty : set the options for a	terminal	stty(1)
tty : get the name of the	terminal	tty(1)
database tput : initialize a	terminal or query terminfo	tput(1)
script : make typescript of	terminal session	script(1)
kill :	terminate a process	kill(1)
: initialize a terminal or query	terminfo database tput	tput(1)
definitions of common UNIX system	terms and symbols glossary :	glossary(1)
command	test : condition evaluation	test(1)
havetcp :	test system for TCP support	havetcp(1)
ed, red :	text editor	ed(1)
ex :	text editor	ex(1)
casual users) edit :	text editor (variant of ex for	edit(1)
newform : change the format of a	text file	newform(1)
program	tftp : trivial file transfer	tftp(1)
merge :	three-way file merge	merge(1)
: update access and modification	times of a file touch	touch(1)
process data and system activity	timex : time a command; report	timex(1)
tsort :	topological sort	tsort(1)
modification times of a file	touch : update access and	touch(1)
query terminfo database	tput : initialize a terminal or	tput(1)
	tr : translate characters	tr(1)
ftp : ARPANT file	transfer program	ftp(1c)
tftp : trivial file	transfer program	tftp(1)
tr :	translate characters	tr(1)
: encode decode a binary file for	transmission via mail	uuencode(1)
tftp :	trivial file transfer program	tftp(1)
deroff : remove nroff	troff, tbl, and eqn constructs	deroff(1)
values	true, false : provide truth	true(1)
u3b5, vax : get processor type	truth value pdp11, u3b, u3b2,	machid(1)
true, false : provide	truth values	true(1)
	tsort : topological sort	tsort(1)
terminal	tty : get the name of the	tty(1)
file : determine file	type	file(1)
u3b2, u3b5, vax : get processor	type truth value pdp11, u3b,	machid(1)
script : make	typescript of terminal session	script(1)
processor machid: mips, pdp11,	u3b, u3b2, u3b5, vax : get	machid(1)
type machid: mips, pdp11, u3b,	u3b2, u3b5, vax : get processor	machid(1)
machid: mips, pdp11, u3b, u3b2,	u3b5, vax : get processor type	machid(1)
	ul : do underlining	ul(1)
mask	umask : set file-creation mode	umask(1)
information	uname : obtain current system	uname(1)
ul : do	underlining	ul(1)
file unget :	undo a previous get of an SCCS	unget(1)
SCCS file	unget : undo a previous get of an	unget(1)
file	uniq : report repeated lines in a	uniq(1)
which project library contains a	unit a.which : determine	a.which(1)
which project library contains a	unit a.which : determine	a.which(1)
a.rm : remove source	unit and library information	a.rm(1)
a.rm : remove source	unit and library information	a.rm(1)
	units : conversion program	units(1)
files pack, pcat,	unpack : compress and expand	pack(1)
times of a file touch :	update access and modification	touch(1)
programs make : maintain,	update, and regenerate groups of	make(1)
and load statistics	uptime : display system up time	uptime(1)
a.du : summarize library disk	usage	a.du(1)
a.du : summarize library disk	usage	a.du(1)
description and usage examples	usage : retrieve a command	usage(1)
a command description and	usage examples usage : retrieve	usage(1)
: initialize system for first	user setup	setup(1)
write : write to another	user	write(1)
crontab :	user crontab file	crontab(1)
su, ssu : substitute	user id temporarily	su(1)
protocol telnet :	user interface to the TELNET	telnet(1c)
editor (variant of ex for casual	users) edit : text	edit(1)
the UNIX system for beginning	users : information about	starter(1)
wall : write to all	users	wall(1)
mail, rmail : send mail to	users or read mail	mail(1)
rwall : write to all	users over a network	rwall(1)
: search a file for a pattern	using full regular expressions	egrep(1)
: identify a UNIX system command	using keywords locate	locate(1)
a.help : interactive help	utility	a.help(1)
a.help : interactive help	utility	a.help(1)
control uustat :	uucp status inquiry and job	uustat(1)
UNIX-to-UNIX system copy	uucp, uulog, uuname :	uucp(1)
a binary file for transmission	uuencode,uudecode : encode decode	uuencode(1)
system copy uucp,	uulog, uuname : UNIX-to-UNIX	uucp(1)

uucp, uulog,	uuname : UNIX-to-UNIX system copy	uucp(1)
system file copy uuto,	uupick : public UNIX-to-UNIX	uuto(1)
job control	uustat : uucp status inquiry and	uustat(1)
UNIX-to-UNIX system file copy	uuto, uupick : public	uuto(1)
execution	uux : UNIX-to-UNIX system command	uux(1)
	val : validate SCCS file	val(1)
val :	validate SCCS file	val(1)
vax : get processor type truth	value pdp11, u3b, u3b2, u3b5,	machid(1)
true, false : provide truth	values	true(1)
edit : text editor	(variant of ex for casual users)	edit(1)
mips, pdp11, u3b, u3b2, u3b5,	vax : get processor type truth	machid(1)
	vc : version control	vc(1)
vc :	version control	vc(1)
get : get a	version of an SCCS file	get(1)
a.vadsrc : display available VADS	versions and create a default	a.vadsrc(1)
a.vadsrc : display available VADS	versions and create a default	a.vadsrc(1)
sccsdiff : compare two	versions of an SCCS file	sccsdiff(1)
display editor based on ex	vi : screen-oriented (visual)	vi(1)
a binary file for transmission	via mail : encode decode	uuencode(1)
man :	view online manual pages	man(1)
: file perusal filter for crt	viewing more, page	more(1)
ex vi : screen-oriented	(visual) display editor based on	vi(1)
vsar :	visual system activity reporter	vsar(1)
tape	vmsbackup : read a VMS backup	vmsbackup(1)
reporter	vsar : visual system activity	vsar(1)
process	wait : await completion of	wait(1)
	wall : write to all users	wall(1)
	wc : word count	wc(1)
rwho :	who's logged in on local machines	rwho(1c)
fold : fold long lines for finite	width output device	fold(1)
winsize : set print	window size	winsize(1)
	winsize : set print window size	winsize(1)
wc :	word count	wc(1)
pwd :	working directory name	pwd(1)
	write : write to another user	write(1)
wall :	write to all users	wall(1)
rwall :	write to all users over a network	rwall(1)
write :	write to another user	write(1)
list(s) and execute command	xargs : construct argument	xargs(1)
programs to implement shared	xstr : extract strings from C	xstr(1)
compiler-compiler	yacc : yet another	yacc(1)
yacc :	yet another compiler-compiler	yacc(1)

NAME

a.cleanlib – reinitialize library directory

SYNOPSIS

a.cleanlib [options] [VADS_library]

DESCRIPTION

a.cleanlib preserves all non-compilation information contained in *ada.lib*, including any additional libraries contained in the library search list and any other directives found in *ada.lib*.

The command will empty the files *GVAS_table*, *ada.lib*, and *gnrx.lib* of all separate compilation information and remove the contents of the directories *.lines*, *.imports*, *.nets*, and *.objects* from the named library, or, if no library is specified, from the current library directory. It will also remove the file *name_lib* if present.

If **a.cleanlib** cannot find every library component, it will abort without removing any information unless the **–f** (force) option is given.

The **-F** option is provided to allow **a.cleanlib** to clean a library having a reserved name (*standard, verdixlib, publiclib*).

OPTIONS

–F	(force name) allow the cleaning of a VADS library having a reserved name
–f	(force) clean the VADS library structure even if components are missing or if lock files are found.

FILES

GVAS_table	address assignment file
gnrx.lib	generic instantiation reference file
ada.lib	library reference file
.lines	line number reference files directory
.imports	imported Ada units directory
.nets	Ada network control files directory
.objects	Ada object files directory

DIAGNOSTICS

An error is reported if any VADS component is missing, and no action is taken unless the **-f** option is used.

SEE ALSO

[VADS Reference], **a.mklib**, **a.rmlib**.

NAME

a.db – source level debugger

SYNOPSIS

a.db [options] [executable_file]

DESCRIPTION

a.db is a symbolic debugger for Ada programs and for C programs compiled with the **-go** option for those using *4.2 BSD UNIX* or the **-g** option on *System V UNIX*. Detailed descriptions of interactive **a.db** commands and runtime configuration file options are provided in the *VADS Debugger Reference*, which is also available on-line using **a.help** or the debugger's internal **help** command.

VADS_location/bin/a.db is a wrapper program that executes the correct executable based upon directives visible in the **ada.lib** file. This permits multiple VADS compilers to exist on the same host. The **-sh** option prints the name of the actual executable file.

OPTIONS

–i *file_name*	(input) read input from the specified file
–p *VADS_library*	(program) read program compilation information from the specified VADS library directory (rather than the current directory)
–sh	(show) display the name of the tool executable but do not execute it.
–v	(visual) invoke the screen-mode debugger directly.

See also

VADS Debugger Reference for a list of all debugger commands.

NAME

a.du – summarize library disk usage

SYNOPSIS

a.du [options] [VADS_library]

DESCRIPTION

a.du lists size in bytes for all compiler-generated files in the specified VADS library. If no library is specified, the current directory is assumed.

VADS_location/bin/a.du is a wrapper program that executes the correct executable based upon directives visible in the *ada.lib* file. This permits multiple VADS compilers to exist on the same host. The **-sh** option prints the name of the actual executable file.

OPTIONS

– **a**	(address) sort the output by the GVAS address of each unit.
– **e**	(erroneous) include information for units with damaged or out-of-date net files.
– **f**	(file) sort output by the name of the file containing the unit.
– **g**	(GVAS) provide the base address of each unit in the GVAS.
– **i**	(imports) include information for imported units.
– **sh**	(show) display the name of the tool executable but do not execute it.

FILES

GVAS_table	address assignment file
.imports	imported Ada units directory
.lines	line number reference files directory
.nets	Ada network control files directory
.objects	Ada object files directory

NAME

a.error – analyze and disperse error messages

SYNOPSIS

a.error [options] [error_file]

DESCRIPTION

a.error is generally called from the **ada** command, but it can also be used separately. **a.error** analyzes and optionally disperses diagnostic error messages produced by the VADS compiler. It looks at the specified error file or the standard input, determines the source file and line number to which the error refers, determines whether the error is to be ignored or not, and outputs the associated source line followed by the error line(s).

a.error will also insert the error lines into the source file and invoke the **vi**(1) editor if the **–v** option is given. Error lines placed into files this way are of two types. The first gives the position of the error and the second identifies it. Multiple errors on a single line are referenced by sequential alphabetic characters.

```
    subtype T is range 1..1f;
-----------^A                  ###
-----------------^B            ###
-### A: syntax error: "identifier" inserted
-### B: lexical error: deleted
```

Because all error lines are flagged with ###, the **vi** editor command **:g/###/d** can be used to delete them. However, any source lines containing ### will also be deleted; consequently, do not use ### in any source with which **a.error -v** may be used.

In the case of source files with multiple links, **a.error** creates a new copy of the file with only one link to it.

OPTIONS

– **e** *editor* (editor) Insert the error messages in the source file and invoke the specified editor.

– **l** (listing) Produce a listing on the standard output.

– **N** (no) do not display line numbers.

–t *number* (tabs) Change tab default setting (8). (No space between **-t** and the following digit.)

– **v** (vi) Embed error messages in the source file and call the environment editor ERROR_EDITOR. (If ERROR_EDITOR is defined, the environment variable ERROR_PATTERN should also be defined. ERROR_PATTERN is an editor search command that locates the first occurrence of '###' in the error file.) If no editor is specified, call **vi**.

– **W** (warnings) Ignore warnings.

DIAGNOSTICS

a.error produces diagnostics indicating 'no errors' if **-v** is used and no errors were detected and 'no such file or directory' if invoked with an invalid file name.

SEE ALSO

[VADS Reference] **ada.**

NAME

a.help – interactive help utility

SYNOPSIS

a.help [-options] [subject]

DESCRIPTION

On-line help is available for each of the VADS utilities and for debugger commands and concepts. Without a specified subject, **a.help** provides information on use of the help utility and prompts for additional subject names. Use **q** to exit from **a.help**.

Without the **-p** option, **a.help** will use the paging program defined by the environment variable HELPER, requiring the full pathname with surrounding quotes for additional options. If HELPER is not defined, **more** is used.

Reference manual entries for the compiler and tools only are available on-line by using the **man** command if the local system administrator has elected to install them. A list of topics can be obtained with

```
man ada
```

and typing

```
man VADS_command)
```

will show the entry for a specific command.

VADS_location\bin\a.help is a wrapper program that executes the correct executable based upon directives visible in the **ada.lib**. This permits multiple VADS compilers to exist on the same host. The **-sh** option prints the name of the actual executable file.

OPTION

– **p** *pager*	(pager) Use **pager** as the paging program. The complete pathname must be given with surrounding quotes if additional options to the paging program are desired.
– **sh**	(show) display the name of the tool executable but do not execute it.

ON-LINE HELP FROM THE DEBUGGER

Access on-line help for the debugger as well as the compiler and tools during a debugging session by typing

help [subject)]

or

:help[**subject**] while in screen mode.

If the subject is omitted, a list of debugger commands is displayed. This overview can also be obtained by typing **intro** after a help prompt. Help with the **help** command can be obtained by typing **help** at a help prompt.

FILES

*BVADS_location/sup/help_files/**

NAME

a.info – list or change VADS library options

SYNOPSIS

a.info [options]

DESCRIPTION

a.info is used to examine the INFO and LINK directives of the **ada.lib**. It can also be used to add or delete those directives from the **ada.lib** or to display or change the library search list.

All directives have the format

name:type:value:

where **name** is the name of the directive, **type** can be the word LINK or INFO, and **value** is usually a file name. (More information on directives can be found in discussions of **a.ld**.)

Without options, **a.info** displays all directives in the current library.

The **-i** option executes **a.info** in interactive mode. In this mode, all command line actions may be performed interactively. **a.info** prompts for and checks that the desired directive names and values are supported.

For a complete list of directive names, BSee also
Implementation Reference, Supported INFO and LINK Directive Names.

For a discussion of WITH*n* directives used with the prelinker, See also *Users Guide, Program Generation Tools*, **a.ld** and *[VADS Reference]*, **a.ld**.

Regular expressions, shown in the options below, are formed by following the operating system documentation.

SEE ALSO

Operating system documentation, **ed** (1).

OPTIONS

– a (all) Display all directives in each library on the library search list.

– F (suffix) Display LINK directives with the suffix (L) and INFO directives with the suffix (I).

– i (interactive) Operate in interactive mode.

– p (path) Print the library search list.

– s (short) Show just the INFO and LINK names.

– v (verbose) Display maximum information.

[+-]**info** *name value*
Add, delete an INFO directive.

[+-]**link** *name value*
Add, delete an LINK directive.

[+-]**link WITHn** *value*
Add, delete an LINK directive.

+**link WITH** *value*
Add a LINK directive having the next number: WITH*n*

[+-]*number VADS_library*
add, delete *VADS_library* in the *number* position to the library search list.

–number removes VADS library in position *number* from the library search list

"regular_expression"
Print directives whose first *name* field matches *regular_expression*.

–value "*regular_expression*"
Print directives whose last *value* field matches *regular_expression*.

FILES

VADS_location\sup\LEGAL.INFO A list of legal directives for this implementation.

NAME

a.ld – prelinker

SYNOPSIS

a.ld [options] unit_name [ld_options]

DESCRIPTION

a.ld collects the object files needed to make **unit_name** a main program and calls the UNIX linker **ld**(1) to link together all Ada and other language objects required to produce an executable image in **a.out**. **unit_name** is the main program and must be a non-generic subprogram. If **unit_name** is a function, it must return a value of the **type** STANDARD.INTEGER. This integer result will be passed back to the UNIX shell as the status code of the execution. The utility uses the net files produced by the Ada compiler to check dependency information. **a.ld** produces an exception mapping table and a unit elaboration table and passes this information to the linker.

a.ld reads instructions for generating executables from the **ada.lib** file in the VADS libraries on the search list. Besides information generated by the compiler, these directives also include WITH*n* directives that allow the automatic linking of object modules compiled from other languages or Ada object modules not named in context clauses in the Ada source. Any number of WITH directives may be placed into a library, but they must be numbered contiguously beginning at WITH1. The directives are recorded in the library's **ada.lib** file and have the following form.

WITH1:LINK:*object_file*:
WITH2:LINK:*archive_file*:

WITH directives may placed in the local Ada libraries or in any VADS library on the search list.

A WITH directive in a local VADS library or earlier on the library search list will hide the same numbered WITH directive in a library later in the library search list.

Use the tool **a.info** to change or report library directives in the current library.

All arguments after *unit_name* are passed on to the linker. These may be options for it, archive libraries, library abbreviations, or object files.

VADS_location/bin/a.ld is a wrapper program that executes the correct executable based upon directives visible in the **ada.lib** file. This permits multiple VADS compilers to exist on the same host. The **-sh** option prints the name of the actual executable file.

OPTIONS

–E *unit_name* (elaborate) Elaborate *unit_name* as early in the elaboration order as possible.

–F (files) Print a list of dependent files in order and suppress linking.

–o *executable_file*
(output) Use the specified file name the name of the output rather than the default, **a.out**.

–sh (show) Display the name of the tool executable but do not execute it. **–U** (units) Print a list of dependent units in order and suppress linking.

–v (verbose) Print the linker command before executing it.

–V (verify) Print the linker command but suppress execution.

FILES

VADS_location/standard/*
startup and standard library routines

*.objects/**	Ada object files
a.out	default output file

SEE ALSO

Operating system documentation, **ld**(1)

DIAGNOSTICS

Self-explanatory diagnostics are produced for missing files, etc. Occasional additional messages are produced by the linker.

NAME

a.list – produce program listing with line numbers

SYNOPSIS

a.list [-N] **ada_source.a**

DESCRIPTION

a.list provides a convenient way of producing a listing for programs containing no errors that closely resembles the output of **a.error**. The listing is written to the standard output and may be piped or redirected to a file.

OPTION

–N	(no) Suppress line numbers.

SEE ALSO

[*VADS Reference*], **a.error, a.pr.**

NAME

a.ls – list compiled programs

SYNOPSIS

a.ls [options] [unit_name] ... [-f ada_source.a ...]

DESCRIPTION

a.ls provides a list of the units compiled in the current VADS directory. Options are provided to give more or less extensive information, to change the format of the list, or to provide a list of compiled units occurring in specified source files. Additionally, **unit_name** can be specified as a regular expression to match groups of units. (If the regular expression contains any of the shell's meta-characters, the expression must be quoted.)

Without the **−l** or **−v** options, **a.ls** prints output in multiple columns. This can be overridden with the **−1** (single) option.

The options **−F**, **−l**, and **−v** (in increasing order of listing detail) are mutually exclusive. If more than one of these three is given, the listing will be that with the most detail.

OPTIONS

−a	(all) List all units visible in libraries in the library search list.
−b	(body) Limit output to unit bodies.
−f *filename*	(file) List only units found in *filename*.
−F	(suffix) List unit bodies with a trailing #.
−l	(long) List source file date, net file date, unit, and unit type.
−s	(specification) Limit output to unit specifications.
−v	(verbose) List source file name, source file date, net file date, and unit.
−1	(single) Print output in a single column.

SEE ALSO

Operating system documentation for regular expressions in *ed*(1).

NAME

a.make – recompile source files in dependency order

SYNOPSIS

a.make [options] [unit_name]... [ld_options] [-f ada_source.a ...]
a.make [options] [path/unit_name]... [ld_options] [-f ada_source.a ...]

DESCRIPTION

This utility determines which files must be recompiled in order to produce a current executable file with **unit_name** as the main unit. It also calls **a.ld** to create the appropriate executable, if and only if **unit_name** is a procedure or an integer function; otherwise, it just ensures that the named unit is up-to-date, recompiling any dependencies if necessary.

The utility uses DIANA net files to determine the correct order of compilation and elaboration.

a.make will have no knowledge of any source file (*foo.a*) until that file has been compiled in a way that changes the program library. Unless the **–f** option is used, this requires that *foo.a* be compiled 'by hand' at least once. Unless the **–U** or **–D** option is given, the file must compile successfully or else the program library will remain unchanged. A single compilation is sufficient (unless syntax errors are present) if the **–U** option is used to force changes to the program library. In any case, syntax errors must be corrected before the file will be 'seen' by **a.make**.

VADS_location/bin/a.make is a wrapper program that executes the correct executable based upon directives visible in the **ada.lib** file. This permits multiple VADS compilers to exist on the same host. The **–sh** option prints the name of the actual executable file.

Supplied names and unknown options are passed to **a.ld**.

OPTIONS

–A *VADS_library* [-A *VADS_library*] ...
: (add) Bring the listed libraries up to date if necessary.

–All
: (all) Bring all libraries on the library search path up to date.

–C "*compiler*"
: (compiler) Use the string *compiler* in recompiling the required units. This option is normally used to provide specific options to the compiler. For example, to call the compiler with the optimizing option and invoke the **vi**(1) editor on compilation errors, use a command of the following type.

```
a.make -C "ada -ev" [other commands]
```

–D
: (dependencies) List the file-to-file dependencies.

–f *ada_source.a* ...
: (files) Treat remaining non-option arguments as file names in the current VADS library to compile. All units in these files will be brought up to date; **-f** may be used with one of the other options to print actions or dependencies without executing them, but must be the last option given.

–I *ada_source.a*
: (if) List actions that would be taken if *ada_source.a* were changed.

–L "*linker*"
: (linker) Use the string *linker* in linking the required units. This option can be used to provide unusual options to **a.ld** when using **a.make**.

–O[0-9]
: (optimize) Invoke the code optimizer (no space before the digit). An optional digit limits the number of optimization passes; without the **-O** option, one pass is made; **-O0** prevents optimization; **O** with no digit optimizes as far as possible.

–g1
: Have the compiler produce additional symbol tabel informatin for accurate

but limited symbolic debugging of partially optimixes code.

−g "or" **−g**2	Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging. **−g**2 is the defult.
−g3	Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.
−S	(suppress) Apply **pragma** SUPPRESS to the entire compilation.
−sh	(show) Display the name of the tool executable but do not execute it.
−U	(units) List the list of dependent units in order, but do not link.
−v	(verbose) List the recompilation commands as they are executed.
−V	(verify) List the recompilation commands that would be executed, but do not execute them.

NAME

a.mklib – make library directory

SYNTAX

a.mklib [-F -f -i -v] [-t target] [new_VADS_library [parent_VADS_library]]

DESCRIPTION

a.mklib creates and initializes a new VADS library directory, creating three files (*GVAS_table, ada.lib*, and *gnrx.lib*) and four directories (**.lines**, **.imports**, **.nets**, and **.objects**). It constructs library pointers in *ada.lib* to all libraries available from the parent library and to the parent library itself. As a result, Ada units in the new library can reference all Ada units defined by the parent library and all units that were accessible from the parent library.

If *parent_VADS_library* is unspecified, the default libraries are *verdixlib* and *standard*.

The tool **a.vadsrc** may also be used to create a local configuration file called *.vadsrc* either in the current directory, or in the user's $HOME directory, so that future libraries created in a directory below the current directory or $HOME directory will reference a particular VADS version.

If *new_VADS_library* is unspecified, the current working directory is initialized.

The **−f** option will force initialization of the VADS library structure, overwriting any existing components and deleting any existing lock files.

Without the **−F** option, **a.mklib** cannot create libraries named *standard*, *verdixlib*, or *publiclib*.

A list of available targets can be obtained with the **−i** option or with the tool **a.vadsrc**.

OPTIONS

−f	(force) Create VADS library structure even if some components are already present.
−F	(force name) Allow creation of VADS library with a restricted name.
−i	(interactive) Display all versions of VADS installed on the system and prompt for selection of VADS version unless modified with the **−t** option.
−t *target*	(target) Create a library for a specific target machine.
−v	(verbose) Display the library search list and target directives.

EXAMPLE

If the user is positioned at the directory */usr/babbage/code* and the VADS library *olddir* exists below it in the UNIX hierarchy, the command

a.mklib newdir olddir

creates the library directory */usr/babbage/code/newdir* and provides access to the Ada compilation units previously compiled in the *olddir* library directory. Any units available to *olddir* from other libraries are now available from *newdir* as well.

FILES

.vadsrc	local default configuration file
/usr/lib/VADS	VADS version reference file

DIAGNOSTICS

An error is reported and no action is taken (without the -f option) if **new_VADS_library** contains any VADS components or lock files or if the name specified exists but is not a directory.

SEE ALSO

[*VADS Reference*], **a.cleanlib, a.rmlib, a.vadsrc.**

NAME

a.pathxs – report or change VADS library search list

SYNTAX

a.path [options] [VADS_library1 [VADS_library2]]

DESCRIPTION

A list of libraries to be searched during compilation is maintained in the current VADS library directory in the file *ada.lib*. **a.path** changes or reports the list of library names contained there. During compilation, any program units not found in the current library will be searched for in the VADS libraries listed on the search list. If the unit is not found in the first VADS library, it is searched for in the second, and so on in listed order. When **a.path** is used with no options, it reports the contents of the current library search list, one library to a line.

OPTIONS

–a *VADS_library1* [*VADS_library2*]
(append) Append *VADS_library1* after *VADS_library2*. With a single argument, append *VADS_library1* to the end of the library search list.

–i *VADS_library* [*VADS_library2*]
(insert) Insert *VADS_library1* before *VADS_library2*. With a single argument, insert *VADS_library1* at the beginning of the list.

–r *VADS_library1*
(remove) Remove *VADS_library1* from the library search list.

–v (verbose) Display path as it is changed.

–t (target) Display library search list and target information.

–x *VADS_library1*
(except) Remove all except *VADS_library1* from the list.

BUGS

Removing a library name from the library search list does not remove compilation information from the referenced libraries.

Maximum length of the library search list is 2048 characters.

NAME

a.pr – format source code

SYNTAX

a.pr [options] [ada_source.a]

DESCRIPTION

a.pr reformats Ada source code according to the options specified in a runtime configuration file with the name **.prrc**. This allows users to tailor **a.pr** for individual Ada coding standards. The configuration file may be located either in the user's current working directory or the home directory.

Additionally, options can be specified on the command line that override those in the configuration file. The options are listed below.

Invoked without a filename, **a.pr** reads its input from standard input.

Error and warning messages are written to standard error.

.prrc CONFIGURATION FILE OPTIONS

(Defaults shown in brackets.)

align_cmts where	align comments to the right of the longest line (*line*) or the longest line containing a comment (*comment*) [*comment*]
chars *number*	Specify maximum number of characters of code per line including comment and indentation; any line extending over this limit will be continued on the next line; valid range is from 20 .. 500 [132].
comment case	print all comments in the specified case: *upper*, *lower*, *same* [*same*]
ident case	print all identifiers in the specified case: *upper*, *lower*, *same* [*upper*]
indent *number*	Specify amount of indentation between levels; valid range is 1 .. 8 [8].
lines *number*	Specify maximum number of lines allowed on a page; valid range is from 1 .. 1000 [55].
margin *number*	Specify starting margin for top-most level; valid range is from 0 .. 15 [0].
no_page	Paginate only when pragma PAGE is encountered.[*page*]
no_warning	suppress warning messages regarding line length greater than desired [provide warnings]
page *number*	Set page size; perform pagination with blank lines; valid range is from 1 .. 1000 [paginate using form feeds].
page_lu	Start each library unit (indicated by a WITH clause) on a new page [do not start on new page].
record where	print *record* on either the same line (*same*) or on the next one (*next*) [*same*]
reserved case	print all reserved words in the specified case: *upper*, *lower*, *same* [*lower*]
tabs *number*	Print tabs for indentation whenever the number of spaces needed for indentation is greater than or equal to the specified number; valid range is from 0 .. 8; if *tabs 0* is specified, indentation will be performed with blanks [8].

a.pr COMMAND LINE OPTIONS

(Defaults shown in brackets.)

–ac	(align comment) Align comments to the right of the longest line that contains

	a comment [default].
–al	(align line) Align comments to the right of the longest line, regardless of whether it contains a comment [-ac].
–c *number*	(characters) Specify maximum number of characters of source code allowed on a line. Valid range is from 20 .. 500 [132].
–cl	(comments lower) Print comments in lower case [-cs].
–cs	(comments same) Print comments as in source code [default].
–cu	(comments upper) Print comments in upper case [-cs].
–i *number*	(indent) Specify indentation between levels. Valid range is from 1 .. 8 [8].
–il	(identifiers lower) Print identifiers in lower case [-iu].
–is	(identifiers same) Print identifiers as in source code [-iu].
–iu	(identifiers upper) Print identifiers in upper case [default].
–l *number*	(lines) Specify maximum number of lines allowed on a page. Valid range is from 1 .. 1000 [55].
–m *number*	(margin) Specify starting margin for top-most level. Valid range is from 0 .. 15 [0].
–nl	(no page library unit) Do not start a new page for each library unit [default].
–np	(no pagination) Specify no pagination. Pagination will occur only when *pragma* PAGE is encountered [-pg].
–nw	(no warnings) Suppress warning messages regarding line length [-w].
–p *number*	(page) Specify page size. Valid range is from 1 .. 1000 [-pg].
–pg	(pagination) Paginate using form feeds [default].
–pl	(page library) Start a new page whenever a library unit is encountered [-nl].
–rl	(reserved lower) Print reserved words in lower case [default].
–RN	(record next) Print RECORD on the line following *type* or *for* [-RS].
–rs	(reserved same) Print reserved words as in source code [-rl].
–RS	(record same) Print RECORD on the same line as *type* or *for* [default].
–ru	(reserved upper) Print reserved words in upper case [-rl].
–t *number*	(tabs) Specify tabs for indentation whenever the number of spaces needed is greater than or equal to the specified number. If *-t 0* is specified, indentation will be performed with spaces. Valid range is from 0 .. 8 [8].
–w	(warning) Provide warning messages regarding line lengths greater than desired [default].

NAME

a.rm – remove source unit and library information

SYNTAX

a.rm [options] unit_name
a.rm [options] [ada_source.a]

DESCRIPTION

The **a.rm** command is executed while positioned in a VADS directory. It removes all information associated with the named unit(s) or file(s). When **unit_name** is specified, the corresponding files in *.nets*, *.objects*, and *.lines* are removed and the *ada.lib* entries are deleted.

When a file name *ada_source.a* is given, all net, object, and line number files are removed for each unit defined in the file, and the appropriate entries are deleted from *ada.lib*. A name ending in *.a* is taken to be an Ada source file name unless the **–u** option is given.

Unit names with dotted notation such as *aaa.bbb* or *aaa.bbb.ccc* are taken to be the names of Ada subunits.

OPTIONS

–b	(body) Delete the bodies of the specified units named files.
–f	(file) Remove the Ada source file in addition to the compiler-generated files whenever all units in a file are deleted.
–i	(interactive) Prompt for confirmation before deleting information for any units.
–s	(specification) Remove the compilation information for the specifications of the specified units.
–u	(unit) Force the next name to be treated as a unit even though it ends in **.a**.
–v	(verbose) List the units as they are removed.
–V	(verify) List the units that would be removed, but do not remove them.

NAME

a.rmlib – remove compilation library

SYNTAX

a.rmlib [-f -F] [VADS_library]

DESCRIPTION

a.rmlib removes all VADS library components from *VADS_library* or from the current library if no argument is given. It removes three files (*GVAS_table*, *ada.lib*, and *gnrx.lib*), four directories (*.lines*, *.imports*, *.nets*, and *.objects*), and lock files, if the –**f** option is used. The directory itself, any other files it contains, and any other subordinate directories are untouched.

If *VADS_library* is unspecified, the current VADS library is used.

If **a.rmlib** cannot find every library component or lock files exist, it will abort without removing any files unless the –**f** (force) option is given.

Without the –**F** option **a.rmlib** cannot operate in a library bearing the name *standard*, *verdixlib*, or *publiclib*.

OPTION

–**f**	(force) Clean VADS library structure even if some components are missing or lock files exist.
–**F**	(force name) Allow the cleaning of the VADS library structure of a library having a restricted name.

DIAGNOSTICS

An error is reported and no action is taken (without the –**f** option) if *VADS_library* contains an incomplete set of components or a lock file.

An error message will be issued if any files or directories are not accessible for deletion.

SEE ALSO

[*VADS Reference*], **a.mklib, a.cleanlib**

BUGS

The directory name for the removed library is left in dependent library paths. This blocks compilation in any dependent libraries until **a.path** is used to remove the path entry that specifies this directory. Compilation could also proceed if a VADS library is re-created in the named directory from which the library information was removed.

NAME

a.run – download and execute a program on the target board [*cross compilers only*]

SYNTAX

a.run [options] [executable_file]

DESCRIPTION

a.run downloads and runs a VOX format file on a target board. The interface (TDM or emulator) must be set up as described, and the target board correctly connected as required by **a.db**.

If the Ada program fails with a runtime error on the board, **a.run** reports the error and the PC at the time of the failure.

If *executable_file* is not given, the name *a.vox* is used.

VADS_location/bin/a.run is a wrapper program that executes the correct executable based upon the names in the *ada.lib* file or indicated by the **-t** option. This permits multiple VADS compilers to exist on the same host. The **-sh** option prints the name of the actual executable file.

OPTIONS

–b	(benchmark) print the elapsed time for running the program.
–c	(checksum) do not checksum the executable load sections.
–l	(load) load **executable_file** only, do not execute.
–s address	(start) set the starting program counter to *address* (must be a hexadecimal number without delimiting characters)
–sh	(show) display the name of the tool executable, but do not execute it.
–**t** target_name	(target) specifies which target to use. Can be used to run in a directory in which no ada library present or to override the target named by the ada.lib file. The –**t** option requires the –**l** or –**s** option, as it does not get the start address from the *ada.lib file*.
–**T** number	(timeout) stop if the program doesn't return after **number** seconds. (0 means no timeout). Default is 240.
–v	(verbose) show downloading progress

NAME

a.tags – create a tags file

SYNTAX

a.tags [options] ada_source.a ...

DESCRIPTION

a.tags makes a *tags* file from the specified Ada source(s). The operation is similar to the UNIX **ctags**(1) command with modifications for Ada-specific features.

Each line of the *tags* file lists the object name, the file in which it is defined, and search patterns for locating each object's definition. UNIX editors such as **vi**(1) or **ex**(1) can use the *tags* file to locate units and, if the **–t** option was used to create the *tags* file, to locate types as well. Create the *tags* file with the command

```
a.tags *.a
```

For example, to edit unit END_PROG without specifying the file that contains it, type the following command.

```
vi -t END_PROG
```

Ada allows unit name overloading, and **a.tags** requires special conventions to access different units having the same name. Ada specifications are named by prefacing the Ada simple me with **s#**. Bodies are named with the unmodified Ada name. Stubs for separates are named by prefacing the Ada simple name with *stub#*.

Nested packages, subprograms, types, generics, and task definitions are always listed with their full name (Ada expanded name) with any tag prefaces added to the simple name. Simple names for nested units are listed only if the simple name is unique across all other tags. Thus the user may use the simple name if it is unique and may always use the full name.

Fully qualified overloaded names within a file are not differentiated. However, the tag identifies the correct file, and repeated application of the search pattern will find the desired subprogram. The search pattern is generalized to match all versions of the overloaded subprogram; this generalization may cause the pattern to recognize things other than the desired unit. Identical fully qualified names across files are not handled.

The **-x** and **-v** options provide listings on the standard output; all other options refer to the file **tags** generated for use by **ex** or **vi**.

OPTIONS

–a	(append) Append to the *tags* file.
–B	(backward) Record backward searching patterns (?).
–F	(forward) Record forward searching patterns (/). Default.
–t	(types) Create tags for types also.
–v	(vgrind) Generate an index with line numbers for **vgrind**(1) on the standard output.
–w	(warnings) Suppress warning messages.
–x	(cross) Generate an indexed list of all tags on the standard output.

SEE ALSO

Operating system documentation, **ctags**(1).

BUGS

When using **ex** or **vi** with the **-t** option, the command line must contain the desired unit or type in the same case (upper or lower) as its occurrence in the source file.

NAME

a.vadsrc – display available VADS versions and create a default library configuration file

SYNTAX

a.vadsrc [-i]

DESCRIPTION

When multiple VADS targets or versions are present on the same system, **a.vadsrc** is useful to control the default version or target processor for which libraries are created.

With no option, **a.vadsrc** simply reports the installed VADS version.

If the –**i** (interactive) option is used, the tool prompts for selection of a VADS version and creates a .vadsrc file in the current directory.

OPTIONS

–**i**	(interactive) Show all versions of VADS installed on the system and prompt for a selection.

Files

/usr/lib/VADS	VADS version reference file

SEE ALSO

[*VADS Reference*], **a.mklib**.

NAME

a.view – establish command abbreviations and history mechanism for C shell

SYNTAX

source **a.view**

DESCRIPTION

a.view defines a number of aliases that simplify and enhance the use of the basic VADS commands for users of the C shell. The alias definitions allow a file name to be set once and thereafter alias commands use it until it is changed. Similarly, a main unit name need be entered only once. (It need not be entered at all if it is the same as the last specified file name prefix.) Compilation and linking aliases enter history and timing information into the *ada.history* file.

For a full description, see the *VADS Users Guide* , *Additional Tools,* **a.view**.

To use the aliases without any alteration, put the following single line in the *.login* file.

```
  source VADS_location/bin/a.view
```

This defines the aliases for interactive use. This line must appear at the beginning of scripts using these aliases.

Aliases defined in **a.view** are summarized below. The term 'tracking' is used to indicate whether or not the main unit name is set to the same as the file name prefix.

ALIASES

a	Compile established file name, put errors in *ada.errors/file_name,* and history entry in *ada.history*.
ad	Compile and run the debugger.
ah	List last entry in **ada.history**.
al	List established file name using **more**.
ald	Link the established main unit.
am	Execute **a.make** using file name specified in **sm** and put errors in *ada.errors/unit_name.m*.
ao	compile and optimize code.
av	Edit the established file name with **vi**.
ax	Execute the established main unit.
axtime	Execute a main unit and put timing entry into **ada.history** .
e	List erroneous lines and diagnostics from last compilation of established file name.
el	List established file name with diagnostics from last compilation interspersed.
ev	Edit the established file name with **vi** with diagnostics from last compilation interspersed.
s *name*	Set file name prefix. If new working directory, then set tracking on. If tracking is on, then set main unit.
sb *name*	Set file name prefix and main unit; set tracking on.
sm *name*	Set main unit and set tracking off, so that the main unit name does not change with **s** command.
sp	Print settings of file name prefix and main unit.
vs	List status for the last executed VADS command.

In the commands that take *name*, additional arguments are ignored, and any trailing *.a* is stripped. (The prefix is desired for the file name.) In addition, only the tail component of name (the part following the last /) is used to set the main unit. (Main unit is an Ada unit name, which does not allow '/'). The intention of this convention is to allow the use of file name substitution for easy specification of a full file name and main unit.

For example, if the current directory contains the files *tasking_limit_test.a* (Ada source) and *tasking_limit_test.out* (executable object) and if there were no other files beginning with tas, the command **s tas*** would set the file name prefix to *tasking_limit_test* and the main unit to the same string. When the main unit name differs from the file name, the **sm** command may be used.

In all other commands, additional arguments are passed to the underlying VADS command. Thus

```
ald -ltermcap
```

will cause the linker to search the *termcap* library in addition to standard libraries.

FILES

ada.history	history of compilations and results
ada.errors	directory containing error files from compilations

DIAGNOSTICS

Warnings are produced if any **set** command is used in a non-VADS library directory or if the specified source file does not exist in the library.

NAME

a.which – determine which project library contains a unit

SYNTAX

a.which [options] [unit_name]
a.which [options] [path/unit_name]

DESCRIPTION

a.which lists the name of the source file that defines the version of *unit_name* visible in the current VADS library. The program library search sequence may also be printed. The **-b** (body) option lists the source file location of the unit body. Without this option, the unit's specification is located.

OPTIONS

–b	(body) Give the location of the body.
–sh	(show) Display the name of the tool executable but do not execute it.
–v	(verbose) Give the library search list.

BUGS

An option is needed so that hidden units can be printed as well to allow programmers to identify unit naming conflicts.

NAME

ada – Ada compiler

SYNTAX

ada [options] [ada_source.a]... [linker_options] [object_file.o]...

DESCRIPTION

The command **ada** executes the Ada compiler and compiles the named Ada source file, ending with the **Ia** suffix. The file must reside in a VADS library directory. The *ada.lib* file in this directory is modified after each Ada unit is compiled.

The object for each compiled Ada unit is left in a file with with the same name as that of the source with **.01**, **.02**, etc. substituted for *.a*. The **-o** option can be used to produce an executable with a name other than *a.out*, the default. For cross compilers, the default name is *a.vox*.

By default, **ada** produces only object and net files. If the **-M** option is used, the compiler automatically invokes **a.ld** and builds a complete program wih the named library unit as the main program.

Non-Ada object files (**.o** files produced by a compiler for another language) may be given as arguments to **ada**. These files will be passed on to the linker and will be linked with the specified Ada object files.

Command line options may be specified in any order, but the order of compilation and the order of the files to be passed to the linker can be significant.

Several VADS compilers may be simultaneously available on a single system. Because the **ada** command in any VADS_location/*bin* on a system will execute the correct compiler components based upon visible library directives, the option **-sh** is provided to print the name of the components actually executed.

Program listings with a disassembly of machine instructions are generated by **a.db** or **a.das**.

OPTIONS

–a *file_name* (archive) treat *file_name* as an *ar* file. Since archive files end with *.a*, *-a* is used to distinguish archive files from Ada source files.

–d (dependencies) analyze for dependencies only. Do not do semantic analysis or code generation. Update the library, marking any defined units as uncompiled. The **-d** option is used by *a.make* to establish dependencies among new files.

–e (error) process compilation error messages using *a.error* and direct it to *stdout*.- only the source lines containing errors are listed. Only one **-e** or **-E** option should be used.

–E

–E *file*

–E *directory* (error output) without a file or directory argument, **ada** processes error messages using **a.error** and directs the output to **stdout**; the raw error messages are left in **ada_source.err**. If a file pathname is given, the raw error messages are placed in that file. If a directory argument is supplied, the raw error output is placed in **dir/source.err**. Only one **-e** or **-E** option should be used.

–el (error listing) intersperse error messages among source lines and direct to *stdout*.

–El

−El *file*	
−El *directory*	(error listing) same as the **-E** option, except that source listing with errors is produced.
−ev	(error vi) process syntax error messages using **a.error**, embed them in the source file, and call the environment editor ERROR_EDITOR. (If ERROR_EDITOR is defined, the environment variable ERROR_PATTERN should also be defined. ERROR_PATTERN is an editor search command that locates the first occurrence of '###' in the error file.) If no editor is specifed, call **vi**.
−g0	Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.
−-g "or" **−g**2	Have the compiler produce additinal symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging. **−g**2 is the default.
−g3	Have the compiler produce additional sybol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.
−l *file_abbreviation*	(link) Link this library file. (Do not space between the **-l**and the file abbreviation.) See also *Operating system documentation,* **ld**(1).
−M *unit_name*	(main) produce an executable program using the named unit as the main program. The unit must be either a parameterless procedure or a parameterless function returning an integer. The executable program will be left in the file *a.out* unless overridden with the **-o** option.
-M *ada_source.a*	(main) like **-M** *unit_name*, except that the unit name is assumed to be the root name of the *.a* file (for *foo.a* the unit is *foo*). Only one *.a* file may be preceded by **-M.**
−o *executable_file*	(output) this option is to be used in conjunction with the **-M** option. *executable_file* is the name of the executable rather than the default *a.out*.
−O0	Turn off all optimizations.
−O1	Turn on all MIPS optimizations that can be done quickly and do one pass using the Verdix optimizer. This is the default.
−O2	Invoke the MIPS global ucode optimizer and optimize as far as possible using the Verdix optimizer. (MIPS global ucode optimizer not supported in this release.) **−O** is the same as **−O**2.
−R *VADS_library*	(recompile instantiation) force analysis of all generic instantiations, causing reinstantiation of any that are out of date.
−S	(suppress) apply **pragma** SUPPRESS to the entire compilation for all suppressible checks.
−T	(timing) print timing information for the compilation.
−v	(verbose) print compiler version number, date and time of compilation, name of file compiled, command input line, total compilation time, and error summary line.
−w	(warnings) suppress warning diagnostics.

–W c arg1,[arg2...] Pass the argument[s] *argi* to a compiler pass, where *c* is one of the characters in the next table that designates the pass.

Pass	Character
include	h
backend driver	D
ucgen	G
ujoin	j
uld	u
usplit	s
umesrge	m
uopt	o
ugen	c
as1	b

SEE ALSO

[VADS Reference] **a.db**, **a.error**, **a.ld**, **a.mklib**, **a.das** and Operating system documentation, **ld(1)**

DIAGNOSTICS

The diagnostics produced by the VADS compiler are intended to be self-explanatory. Most refer to the RM. Each RM reference includes a section number and optionally, a paragraph number enclosed in parentheses.

NAME

admin – create and administer SCCS files

SYNOPSIS

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

DESCRIPTION

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

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

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

−n	This keyletter indicates that a new SCCS file is to be created.
−i*[name]*	The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **−r** keyletter for delta numbering scheme). If the **i** keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the **i** keyletter is supplied. Using a single *admin* to create two or more SCCS files requires that they be created empty (no **−i** keyletter). Note that the **−i** keyletter implies the **−n** keyletter.
−r*rel*	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 greater than 0 but less

than or equal to 9999, which may be retrieved by a *get(1)* command for editing. The default value for an unspecified **c** flag is 9999.

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

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

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

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

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

<list> ::= <range> | <list> , <range>
<range>~::= | **a**

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

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

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

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

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

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

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

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

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

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

−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.

−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).

−h Causes *admin* to check the structure of the SCCS file [see *sccsfile(4)*], 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.

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.

FILES

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*.

SEE ALSO

delta(1), ed(1), get(1), help(1), prs(1), what(1).
sccsfile(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

ar – archive and library maintainer

SYNOPSIS

ar option [*posname*] file1 ... fileN

DESCRIPTION

The archiver (**ar**) maintains groups of files as a single archive file. Generally, you use this utility to create and update library files that the link editor uses; however, you can use the archiver for any similar purpose. **NOTE:** This version uses a portable ACII- format archive that you can use on various machines that run UNIX.

In the text, option refers to a character (from the set **drqtpmx**) that you can concatenate with one or more of **svuaibclo.** A suboption refers to options (from the set **abiou**) that you can only use with other options.

The options do these things:

d Deletes the specified files from the archive file.

r Replaces the specified files in the archive file. If you use the suboption **u** with **r,** the archiver only replaces those files that have 'last-modified' dates later than the archive files. If you use a positioning character (from the set **abi**) you must specify the *posname* argument to tell the archiver to put the new files after (**a**) or before (**b** or **i**). Otherwise, the archiver puts new files at the end of the archive.

q Appends the specified files to the end of the archive file. The archiver does not accept suboption positioning characters with the **q** option. It also does not check whether the files you want to add already exist in the archive. Use the **q** option only to avoid quadratic behavior when you create a large archive piece by piece.

t Prints a table of contents for the files in the archive file. If you don't specify any file names, the archiver builds a table of contents for all files. If you specify file names, the archiver builds a table of contents only for those files.

p Prints the specified files from the archive.

m Moves the specified files to the end of the archive. If you specify a positioning character, you must also specify the *posname* (as in option **r**) to tell the archiver where to move the files.

x Extracts the specified files from the archive. If you don't specify any file names, the archiver extracts all files. When it extracts files, the archiver does not change any file. Normally, the 'last-modified' date for each extracted file shows the date when someone extracted it; however, when you use **o,** the archiver resets the 'last-modified' date to the date recorded in the archive.

s Makes a symbol definition (symdef file) as the first file of an archive. This file contains a hash table of *ranlib* structures and a corresponding string table. The symdef file's name is based on the byte ordering of the hash table and the byte ordering of the file's target machine. Files must be consistent in their target byte ordering before the archiver can create a symdef file. If you change the archive contents, the symdef file becomes obsolete because the archive file's name changes. If you specify 's', the archiver creates the symdef file as its last action before finishing execution. You must specify at least one other archive option (m, p, q, r, or t) when you use the **s** option. For UMIPS-V, archives include member objects based on the definition of a common object only. For UMIPS-BSD, they define the common object, but do not include the object.

v Gives a verbose file-by-file description as the archiver makes a new archive file from an old archive and its constituent files. When you use this option with **t,** the archiver lists all information about the files in the archive. When you use this option with **p**, the archiver precedes each file with a name.

c Suppresses the normal message that the archiver prints when it creates the specified archive file. Normally, the archiver creates the specified archiver file when it needs to.

l Puts temporary files in the local directory. Normally, the archiver puts its temporary files in the directory /tmp.

The suboptions do these things:

a Specifies that the file goes after the existing file (*posname*). Use this suboption with the **m** or **r** options.

b Specifies that the file goes before the existing file (*posname*). Use this suboption with the **m** or **r** options.

i Specifies that the file goes before the existing file (*posname*). Use this suboption with the **m** or **r** options.

o Forces a newly created file to have the 'last modified' data that it had before it was extracted from the archive. Use this suboption with the **x** option.

u Prevents the archiver from replacing an existing file unless the replacement is newer than the existing file. This option uses the UNIX system 'last modified' data for this comparison. Use this suboption with the **r** option.

FILES

/tmp/v*temporaries

SEE ALSO

lorder(1), ld(1), odump(1), ar(4), ranhash(3x).

BUGS

If you specify the same file twice in an argument list, it can appear twice in the archive file.

The **o** option does not change the 'last-modified' date of a file unless you own the extracted file or you are the super-user.

NAME

as – MIPS assembler

SYNOPSIS

as [option] ... file

DESCRIPTION

as, the MIPS assembler, produces files in the following formats: MIPS object code in MIPS extended *coff* format (the normal result) and binary assembly language. *As* never runs the loader. *As* accepts one type of argument:

The argument *file* is assumed to be symbolic assembly language source program. It is assembled, producing an object file.

Mas always defines the C preprocessor macros **mips**, **host_mips**, **unix** and **LANGUAGE_ASSEMBLY** to the C macro preprocessor. It also defines **SYSTYPE_SYSV** by default but this changes if the **−systype** *name* option is specified (see the description below).

The following options are interpreted by *as* and have the same meaning in *cc*(1).

−g0 Have the assembler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the assembler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the assembler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the assembler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages.

−P Run only the C macro preprocessor and put the result in a file with the suffix of the source file changed to '.i' or if the file has no suffix then a '.i' is added to the source file name. The '.i' file has no '#' lines in it. This sets the **−cpp** option.

−E Run only the C macro preprocessor on the file and send the result to the standard output. This sets the **−cpp** option.

−o *output*
Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−D*name*=*def*
−D*name*
Define the *name* to the C macro 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 specified in **−I** options, and finally in the standard directory (**/usr/include**).

−I This option will cause '#include' files never to be searched for in the standard directory (**/usr/include**).

−G *num*
Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 8 bytes.

−v Print the passes as they execute with their arguments and their input and output files.

−V Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−cpp Run the C macro preprocessor on assembly source files before compiling. This is the default for *as*(1).

−nocpp
Do not run the C macro preprocessor on assembly source files before compiling.

Either object file target byte ordering can be produced by *as*. The default target byte ordering matches the machine where the assembler is running. The options **−EB** and **−EL** specify the target byte ordering (big-endian and little-endian, respectively). The assembler also defines a C preprocessor macro for the target byte ordering. These C preprocessor macros are **MIPSEB** and **MIPSEL** for big-endian and little-endian byte ordering respectively.

−EB Produce object files targeted for big-endian byte ordering. The C preprocessor macro **MIPSEB** is defined by the assembler.

−EL Produce object files targeted for little-endian byte ordering. The C preprocessor macro **MIPSEL** is defined by the assembler.

The following option is specific for *as*:

−m Apply the M4 preprocessor to the source file before assembling it.

The option described below is primarily used to provide UNIX compilation environments other than the native compilation environment.

−systype *name*
Use the named compilation environment *name*. See *compilation*(7) for the compilation environments that are supported and their *name*s. This has the effect of changing the standard directory for '#include' files. The new items are located in their usual paths but with */name* prepended to their paths. Also a preprocessor macro of the form **SYSTYPE_***NAME* (with *name* capitalized) is defined in place of the default **SYSTYPE_SYSV.**

The options described below primarily aid compiler development and are not generally used:

−H*c* Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be [**a**]. It selects the assembler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T', or a '.T' is added if the source file has no suffix. This file is not removed.

−K Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.G' file for binary assembly language). If the source file has no suffix the conventional suffix is added to the source file name. These intermediate files are never removed even when a pass encounters a fatal error.

−W*c[c...],arg1[,arg2...]*
Pass the argument[s] *argi* to the compiler pass[es] *c[c..]*. The *c's* are one of [**pab**]. The c's selects the compiler pass in the same way as the **−t** option.

The options **−t[hpab]**, **−h***path,* and **−B***string* select a name to use for a particular pass. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

−t[**hpab**]

Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
include	h (see note below)
cpp	p
as0	a
as1	b

If the character 'h' is in the **−t** argument then a directory is added to the list of directories to be used in searching for '#include' files. This directory name has the form COMP_TARGET_ROOT/usr/include*string* . This directory is to contain the include files for the *string* release of the compiler. The standard directory is still searched.

−h*path*

Use *path* rather than the directory where the name is normally found.

−B*string*

Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B,** the **−t** option is assumed to be "hpab". This list designates all names.

Invoking the assembler with a name of the form **as***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/.** If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for the includes rather than the default **/.**

If the environment variable ROOTDIR is set, the value is used as the root directory for all names rather than the default **/usr/.** This also affects the standard directory for '#include' files, /usr/include .

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/ .**

Other arguments are ignored.

FILES

file.o	object file
a.out	assembler output
/tmp/ctm?	temporary
/usr/lib/cpp	C macro preprocessor
/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/include	standard directory for '#include' files

SEE ALSO

Assembly Language Programmer's Guide
cc(1), as0(1), what(1)

DIAGNOSTICS

The diagnostics produced by the assembler are intended to be self-explanatory.

NAME

at, batch – execute commands at a later time

SYNOPSIS

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

DESCRIPTION

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

−r Removes jobs previously scheduled with *at*.

−l Reports all jobs scheduled for the invoking user. Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost. Users are permitted to use *at* if their name appears in the file */usr/lib/cron/at.allow*. If that file does not exist, the file */usr/lib/cron/at.deny* is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If *at.deny* is empty, global usage is permitted. The allow/deny files consist of one user name per line. These files can only be modified by the superuser. The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix *am* or *pm* may be appended; otherwise a 24-hour clock time is understood. The suffix *zulu* may be used to indicate GMT. The special names *noon*, *midnight*, *now*, and *next* are also recognized. An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special "days", *today* and *tomorrow* are recognized. If no *date* is given, *today* is assumed if the given hour is greater than the current hour and *tomorrow* is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed. The optional *increment* is simply a number suffixed by one of the following: *minutes*, *hours*, *days*, *weeks*, *months*, or *years*. (The singular form is also accepted.)
Thus legitimate commands include:

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

at and *batch* write the job number and schedule time to standard error. *batch* submits a batch job. It is almost equivalent to "at now", but not quite. For one, it goes into a different queue. For another, "at now" will respond with the error message *too late*. *at* **-r** removes jobs previously scheduled by *at* or **batch**. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can only remove your own jobs unless you are the super-user.

EXAMPLES

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

This sequence can be used at a terminal:

```
            batch
```

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

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

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

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

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

FILES

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

SEE ALSO

kill(1), mail(1), nice(1), ps(1), sh(1), sort(1).
cron(1M) in the *System Administrator's Reference Manual.*

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [**−d**] [**−F***c*] *commands* [*parameters*] [*file*...]

awk [**−d**] [**−Fc**] [**−f** *script*] [*parameters*] [*file*...]

DESCRIPTION

awk scans each input *file* for lines that match any of a set of patterns specified in the *commands*argument. With each pattern in the *commands*, 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 in the *commands*, or in a file specified as **−f** *script*. The *commands* must be enclosed in single quotes (′) to protect it from the shell.

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

A debug option, **−d** , allows the user to print the entire state of *awk* as the program runs. Be forewarned that volumes of data, mostly meaningless unless you are familiar with the internals of **awk**, will be printed on **stderr**.

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

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

A pattern-action statement has the form:

```
pattern { action }
```

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

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

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

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

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

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

A relational expression is one of the following:

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

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

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

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

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

or by using the **−F***c* option.

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

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 }
```

Print file, filling in page numbers starting at 5:

```
/Page/  { $2 = n++; }
        { print }
```

command line: awk −f program n=5 input

SEE ALSO

grep(1), lex(1), sed(1).
printf(3S) in the *Programmer's Reference Manual.*

ERRORS

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

NAME

banner − make posters

SYNOPSIS

banner *strings*

DESCRIPTION

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

SEE ALSO

echo(1).

NAME

basename, dirname deliver portions of path names

SYNOPSIS

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

DESCRIPTION

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

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

EXAMPLES

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

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

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

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

SEE ALSO

sh(1)

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [**−c**] [**−l**] [*file* ...]

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The *bc(1)* utility is actually a preprocessor for *dc(1),* which it invokes automatically unless the **−c** option is present. In this case the *dc* input is sent to the standard output instead. The options are as follows:

−c Compile only. The output is send to the standard output.

−l Argument stands for the name of an arbitrary precision math library.

The syntax for *bc* programs is as follows; L means letter a–z, E means expression, S means statement.

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

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

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

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

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

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

Functions in **−l** math library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

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

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

EXAMPLE

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

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

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

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

FILES

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

SEE ALSO

dc(1).

BUGS

The **bc** command does not yet recognize the logical operators, **&&** and **||**.
for statement must have all three expressions (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 in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for *diff*.

The parameters to *bdiff* are:

file1 (file2)
The name of a file to be used. If *file1 (file2)* is –, the standard input is read.

n
The number of line 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.

–s
Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff(1)*, which *bdiff* calls.

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. 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), help(1).

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

bfs − big file scanner

SYNOPSIS

bfs [**−**] *name*

DESCRIPTION

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

Normally, the size of the file being scanned is printed, as is the size of any file written with the w command. The optional **−** suppresses printing of sizes. Input is prompted with * if **P** and a carriage return are typed, as in *ed(1)*. Prompting can be turned off again by inputting another **P** and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed(1)* are supported. In addition, regular expressions may be surrounded with two symbols besides **/** and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters **a** through **z** may be used, and all 26 marks are remembered.

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

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

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

xo [*file*]
: Further output from the **p** and null commands is diverted to the named *file*, which, if necessary, is created mode 666 (readable and writable by everyone), unless your *umask* setting (see *umask(1)*) dictates otherwise. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

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

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

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

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

```
xb/^/ label
```

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

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

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

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

will all print the first 100 lines.

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

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

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

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

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

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

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

```
xv7\!date
```

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

xbz *label*

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

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

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

SEE ALSO

csplit(1), ed(1), umask(1).

DIAGNOSTICS

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

NAME

cal – print calendar

SYNOPSIS

cal [[*month*] *year*]

DESCRIPTION

cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and the United States.

EXAMPLES

An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type:

```
cal 9 1752
```

ERRORS

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

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

calendar consults the file *calendar* in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". 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 or her login directory and sends them any positive results by *mail(1).* Normally this is done daily by facilities in the UNIX operating system.

FILES

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

SEE ALSO

mail(1).

ERRORS

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

NAME

cat – concatenate and print files

SYNOPSIS

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

DESCRIPTION

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

```
cat file
```

prints the file, and:

```
cat file1 file2 >file3
```

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

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

The following options apply to **cat**.

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

−s　**Cat** is silent about non-existent files.

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

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

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

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

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

WARNING

Command formats such as

```
cat file1 file2 >file1
```

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

SEE ALSO

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

NAME

cb - C program beautifier

SYNOPSIS

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

DESCRIPTION

The *cb* comand reads C programs either from its arguments or from the standard input, and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, *cb* preserves all user new-lines.

cb accepts the following options.

−s	Canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*.
−j	Causes split lines to be put back together.
−l *leng*	Causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).
The C Programming Language. Prentice-Hall, 1978.

ERRORS

Punctuation that is hidden in preprocessor statements will cause indentation errors. Structure assignments are not handled properly. The braces ({}) are treated like other braces in that they cause indentation.

NAME

cc – MIPS C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

cc, the MIPS *ucode* C compiler, produces files in the following formats; MIPS object code in MIPS extended *coff* format (the normal result), binary or symbolic *ucode*, *ucode* object files and binary or symbolic assembly language. *cc* accepts several types of arguments:

Arguments whose names end with '.c' are assumed to be C source programs. They are compiled, and each object program is left in the file whose name consists of the last component of the source with '.o' substituted for '.c'. The '.o' file is only deleted when a single source program is compiled and loaded all at once.

Arguments whose names end with '.s' are assumed to be symbolic assembly language source programs. They are assembled, producing a '.o' file. Arguments whose names end with '.i' are assumed to be C source after being processed by the C preprocessor. They are compiled without being processed by the C preprocessor.

If the highest level of optimization is specified (with the **−O3** flag) or only ucode object files are to be produced (with the **−j** flag) each C source file is compiled into a *ucode* object file. The *ucode* object file is left in a file whose name consists of the last component of the source with '.u' substituted for '.c'.

The suffixes described below primarily aid compiler development and are not generally used. Arguments whose names end with '.B', '.O', '.S', and '.M' are assumed to be binary *ucode*, produced by the front end, optimizer, ucode object file splitter and ucode merger respectively. Arguments whose names end with '.U' are assumed to be symbolic *ucode*. Arguments whose names end with '.G' are assumed to be binary assembly language, which is produced by the code generator and the symbolic to binary assembler.

Files that are assumed to be binary *ucode*, symbolic *ucode*, or binary assembly language by the suffix conventions are also assumed to have their corresponding symbol table in a file with a '.T' suffix.

cc always defines the C preprocessor macros **mips**, **host_mips** and **unix** to the C macro preprocessor and defines the C preprocessor macro **LANGUAGE_C** when a '.c' file is being compiled. *cc* will define the C preprocessor macro **LANGUAGE_ASSEMBLY** when a '.s' file is being compiled. It also defines **SYSTYPE_SYSV** by default but this changes if the **−systype** *name* option is specified (see the description below).

The following options are interpreted by *cc*(1). 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.

−g0 Have the compiler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages.

−p0 Do not permit any profiling. This is the default. If loading happens, the standard runtime startup routine (**crt1.o**) is used, no profiling library is searched.

−p1 or **−p**
Set up for profiling by periodically sampling the value of the program counter. This option only affects the loading. When loading happens, this option replaces the standard runtime startup routine with the profiling runtime startup routine (**mcrt1.o**) and searches the level 1 profiling library (**libprof1.a**). When profiling happens, the startup routine calls *monstartup*(3) and produces a file *mon.out* that contains execution-profiling data for use with the postprocessor *prof*(1).

−O0 Turn off all optimizations.

−O1 Turn on all optimizations that can be done quickly. This is the default.

−O or **−O2**
Invoke the global *ucode* optimizer.

−O3 Do all optimizations, including global register allocation. This option must precede all source file arguments. With this option, a *ucode* object file is created for each C source file and left in a '.u' file. The newly created ucode object files, the ucode object files specified on the command line and the runtime startup routine and all the runtime libraries are ucode linked. Optimization is done on the resulting ucode linked file and then it is linked as normal producing an "a.out" file. No resulting '.o' file is left from the ucode linked result as in previous releases. In fact **−c** can no longer be specified with **−O3**.

−feedback *file*
Used with the **−cord** option to specify *file* to be used as a feedback file. This *file* is produced by *prof*(1) with its **−feedback** option from an execution of the program produced by *pixie*(1).

−cord Run the procedure-rearranger, *cord*(1), on the resulting file after linking. The rearrangement is done to reduce the cache conflicts of the program's text. The output of *cord*(1) is left in the file specified by the **−o** *output* option or 'a.out' by default. At least one **−feedback** *file* must be specified.

−j Compile the specified source programs, and leave the *ucode* object file output in corresponding files suffixed with '.u'.

−ko *output*
Name the output file created by the ucode loader as *output*. This file is not removed. If this file is compiled, the object file is left in a file whose name consists of *output* with the suffix changed to a '.o'. If *output* has no suffix, a '.o' suffix is appended to *output*.

−k Pass options that start with a **−k** to the ucode loader. This option is used to specify ucode libraries (with **−kl***x*) and other ucode loader options.

−S Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with '.s'.

−P Run only the C macro preprocessor and put the result for each source file (by suffix convention, i.e. '.c' and '.s') in a corresponding '.i' file. The '.i' file has no '#' lines in it. This sets the **−cpp** option.

−E Run only the C macro preprocessor on the files (regardless of any suffix or not), and send the result to the standard output. This sets the **−cpp** option.

−o *output*

Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−D*name*=*def*
−D*name*
Define the *name* to the C macro 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 specified in **−I** options, and finally in the standard directory (**/usr/include**).

−I This option will cause '#include' files never to be searched for in the standard directory (**/usr/include**).

−G *num*
Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 8 bytes.

−v Print the passes as they execute with their arguments and their input and output files.

−V Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−std Have the compiler produce warnings for things that are not standard in the language.

−cpp Run the C macro preprocessor on C and assembly source files before compiling. This is the default for *cc*(1).

−nocpp
Do not run the C macro preprocessor on C and assembly source files before compiling.

−Olimit *num*
Specify the maximum size, in basic blocks, of a routine that will be optimized by the global optimizer. If a routine has more than this number of basic blocks it will not be optimized and a message will be printed. An option specifying that the global optimizer is to be run (**−O**, **−O2**, or **−O3**) must also be specified. *Num* is assumed to be a decimal number. The default value for *num* is 500 basic blocks.

Either object file target byte ordering can be produced by *cc*. The default target byte ordering matches the machine where the compiler is running. The options **−EB** and **−EL** specify the target byte ordering (big-endian and little-endian, respectively). The compiler also defines a C preprocessor macro for the target byte ordering. These C preprocessor macros are **MIPSEB** and **MIPSEL** for big-endian and little-endian byte ordering respectively.

If the specified target byte ordering does not match the machine where the compiler is running, then the runtime startups and libraries come from **/usr/libeb** for big-endian runtimes on a little-endian machine and from **/usr/libel** for little-endian runtimes on a big-endian machine.

−EB Produce object files targeted for big-endian byte ordering. The C preprocessor macro **MIPSEB** is defined by the compiler.

−EL Produce object files targeted for little-endian byte ordering. The C preprocessor macro **MIPSEL** is defined by the compiler.

The following options are specific to *cc*:

−signed

Cause all *char* declarations to be *signed char* declarations, the default is to treat them as *unsigned char* declarations.

−volatile
Causes all variables to be treated as *volatile.*

−varargs
Prints warnings for lines that may require the *varargs.h* macros.

−float Cause the compiler to never promote expressions of type *float* to type *double.*

The option described below is primarily used to provide UNIX compilation environments other than the native compilation environment.

−systype *name*
Use the named compilation environment *name.* See *compilation*(7) for the compilation environments that are supported and their *name*s. This has the effect of changing the standard directory for '#include' files, the runtime libraries and where runtime libraries are searched for. The new items are located in their usual paths but with */name* prepended to their paths. Also a preprocessor macro of the form **SYSTYPE_*NAME*** (with *name* capitalized) is defined in place of the default **SYSTYPE_SYSV.**

The options described below primarily aid compiler development and are not generally used:

−H*c* Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be **[fjusmoca].** It selects the compiler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T' and is not removed.

−K Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.B' file for binary *ucode,* produced by the front end). These intermediate files are never removed even when a pass encounters a fatal error. When ucode linking is performed and the **−K** option is specified the base name of the files created after the ucode link is 'u.out' by default. If **−ko** *output* is specified, the base name of the object file is *output* without the suffix if it exists or suffixes are appended to *output* if it has no suffix.

−# Converts binary *ucode* files ('.B') or optimized binary ucode files ('.O') to symbolic *ucode* (a '.U' file) using *btou*(1). If a symbolic ucode file is to be produced by converting the binary *ucode* from the C compiler front end then the front end option **−Xu** is used instead of *btou*(1).

−W*c[c...],arg1[,arg2...]*
Pass the argument[s] *argi* to the compiler pass[es] *c[c..].* The *c's* are one of **[pfjusmocablyz].** The c's selects the compiler pass in the same way as the **−t** option.

The options **−t[hpfjusmocablyzrnt]**, **−h***path,* and **−B***string* select a name to use for a particular pass, startup routine, or standard library. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

The **−EB** or **−EL** options, the **−p[01]** options and the **−systype** option must precede all **−B** options because they can affect the location of runtimes and what runtimes are used.

−t[hpfjusmocablyzrnt]
Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
include	h (see note below)
cpp	p
ccom	f
ujoin	j
uld	u
usplit	s
umerge	m
uopt	o
ugen	c
as0	a
as1	b
ld	l
ftoc	y
cord	z
[m]crt[1n].o	r
libprof1.a	n
btou, utob	t

If the character 'h' is in the **−t** argument then a directory is added to the list of directories to be used in searching for '#include' files. This directory name has the form COMP_TARGET_ROOT/usr/include*string* . This directory is to contain the include files for the *string* release of the compiler. The standard directory is still searched.

−h*path*
Use *path* rather than the directory where the name is normally found.

−B*string*
Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B**, the **−t** option is assumed to be "hpfjusmocablyzrnt". This list designates all names. If no **−t** argument has been processed before the **−B** then a **−B***string* is passed to the loader to use with its **−l***x* arguments.

Invoking the compiler with a name of the form **cc***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/**. If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for all include and library names rather than the default **/**. This affects the standard directory for '#include' files, /usr/include, and the standard library, /usr/lib/libc.a. If this is set, the first directory that is searched for libraries, using the **−l***x* option, is COMP_TARGET_ROOT/usr/lib/cmplrs/cc. The standard directories for libraries are then searched, see *ld*(1).

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/** .

If the environment variable RLS_ID_OBJECT is set, the value is used as the name of an object to link in if a link takes place. This is used to add release identification information to objects. It is always the last object specified to the loader. See *rls_id*(1) for the tools to create this information.

Other arguments are assumed to be either loader options or *C*-compatible object files, typically produced by an earlier *cc* run, or perhaps libraries of *C*-compatible routines. These files, together with the results of any compilations specified, are loaded in the order given, producing an executable program with the default name **a.out.**

FILES

file.c	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/usr/lib/cpp	C macro preprocessor
/usr/lib/ccom	C front end
/usr/lib/ujoin	binary ucode and symbol table joiner
/usr/bin/uld	ucode loader
/usr/lib/usplit	binary ucode and symbol table splitter
/usr/lib/umerge	procedure intergrator
/usr/lib/uopt	optional global ucode optimizer
/usr/lib/ugen	code generator
/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/lib/crt1.o	runtime startup
/usr/lib/crtn.o	runtime startup
/usr/lib/mcrt1.o	startup for profiling
/usr/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libprof1.a	level 1 profiling library
/usr/include	standard directory for '#include' files
/usr/bin/ld	MIPS loader
/usr/lib/ftoc	interface between *prof*(1) and *cord*(1)
/usr/lib/cord	procedure-rearranger
/usr/bin/btou	binary to symbolic ucode translator
/usr/bin/utob	symbolic to binary ucode translator
mon.out	file produced for analysis by *prof*(1)

Runtime startups and libraries for the opposite byte sex of machine the compiler is running on have the same names but are located in different directories. For big-endian runtimes on a little-endian machine the directory is /usr/libeb and for little-endian runtimes on a big-endian machine the directory is /usr/libel.

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
B. W. Kernighan, *Programming in C–a tutorial*
D. M. Ritchie, *C Reference Manual*
Languages Programmer's Guide
monstartup(3), prof(1), ld(1), dbx(1), what(1), cord(1), pixie(1), ftoc(1)

DIAGNOSTICS

The diagnostics produced by *cc* are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

NOTES

The standard library, /usr/lib/libc.a, is loaded by using the -lc loader option and not a full path name. The wrong one could be loaded if there are files with the name libc.a*string* in the directories specified with the **−L** loader option or in the default directories searched by the loader.

The handling of include directories and libc.a is confusing.

NAME

cd - change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

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

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

SEE ALSO

pwd(1), sh(1).
chdir(2) in the *Programmer's Reference Manual.*

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

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

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

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

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

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

Simply stated, the keyletter arguments 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

```
n 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.

```
n 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), help(1), get(1), prs(1).
sccsfile(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

cflow – generate C flowgraph

SYNOPSIS

cflow [**−r**] [**−ix**] [**−i_**] [**−d** *num*] [**−g** *file*] [**−f** *file*] *file...*

DESCRIPTION

The *cflow* command analyzes a collection of C, *yacc, lex,* assembler, and object files and attempts to build a graph charting the external references. Files suffixed with **.y**, **.l**, and **.c** are *yacc, lex,* and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with **.i**, are then run through the first pass of *lint(1).* Files suffixed with **.s** are assembled. Assembled files, and files suffixed with **.o**, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is displayed upon the standard output.

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

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

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}

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

the command

```
cflow −ix file.c
```

produces the output

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

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

In addition to the **−D**, **−I**, and **−U** options [which are interpreted just as they are by *cc(1)* and *cpp(1)*], the following options are interpreted by **cflow**:

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

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

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

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

−g *file* Use the named file as the temporary file instead of the default. This is useful when temporary space is low.

−f *file* Use the named language source file as input, but do not preprocess the file.

DIAGNOSTICS

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

SEE ALSO

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

BUGS

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

NAME

chmod – change mode

SYNOPSIS

chmod *mode file ...*

chmod *mode directory ...*

DESCRIPTION

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

chmod nnn file(s)

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

chmod a operator b file(s)

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

An absolute mode is given as an octal number constructed from the OR of the following modes:

4000	set user ID on execution
20#0	set group ID on execution if # is **7**, **5**, **3**, or **1**
	enable mandatory locking if # is **6**, **4**, **2**, or **0**
1000	sticky bit is turned on ((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

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that **u**ser, **g**roup, and **o**thers all have **r**eading, **w**riting, and e**x**ecution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

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

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

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

chmod g+rw file

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

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

u	**u**ser's permissions
g	**g**roup's permissions
o	**o**thers permissions

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

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

Permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit is turned on
l	mandatory locking will occur during access

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

Mandatory file and record locking (**l**̂) refers to a file's ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on the set-group-ID and enable a file to be locked on execution at the same time. The following examples,

```
chmod g+x,+l le

chmod g+s,+l le
```

are, therefore, illegal usages and will elicit error messages.

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

EXAMPLES

```
chmod a−x file
chmod 444 file
```

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

```
chmod go+rw file
chmod 606 file
```

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

```
chmod +l file
```

This causes a file to be locked during access.

```
chmod =rwx,g+s file
chmod 2777 file
```

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

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the *ls −l* command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

SEE ALSO

ls(1).
chmod(2) in the *Programmer's Reference Manual*.

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown *owner file* ...

chown *owner directory* ...

chgrp *group file* ...

chgrp *group directory* ...

DESCRIPTION

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

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

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

Only the owner of a file (or the super-user) may change the owner or group of that file.

FILES

/etc/passwd
/etc/group

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the *ls −l* command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

SEE ALSO

chmod(1).
chown(2), group(4), passwd(4) in the *Programmer's Reference Manual*.

NAME

ci - check in RCS revisions

SYNOPSIS

ci [*options*] *file* ...

DESCRIPTION

ci stores new revisions into RCS files. Each file name ending in ',v' is taken to be an RCS file, all others are assumed to be working files containing new revisions. **Ci** deposits the contents of each working file into the corresponding RCS file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section of *co(1)*).

1) Both the RCS file and the working file are given. The RCS file name is of the form *path1/workfile*,v and the working file name is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then the working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ',v'.

3) Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix ',v'.

If the RCS file is omitted or specified without a path, then *ci* looks for the RCS file first in the directory ./RCS and then in the current directory.

For *ci* to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to *strict* (see *rcs(1)*). A lock held by someone else may be broken with the *rcs* command.

Normally, *ci* checks whether the revision to be deposited is different from the preceding one. If it is not different, *ci* either aborts the deposit (if **-q** is given) or asks whether to abort (if **-q** is omitted). A deposit can be forced with the **-f** option.

For each revision deposited, *ci* prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single '.' or a control-D. If several files are checked in, *ci* asks whether to reuse the previous log message. If the std. input is not a terminal, *ci* suppresses the prompt and uses the same log message for all files. See also **-m**.

The number of the deposited revision can be given by any of the options **-r**, **-f**, **-k**, **-l**, **-u**, or **-q** (see **-r**).

If the RCS file does not exist, *ci* creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, *ci* requests descriptive text (see **-t** below).

−r[*rev*] assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is also the default.

If *rev* is omitted, *ci* derives the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1. If the caller holds no lock, but he is the owner of the file and locking is not set to *strict*, then the revision is appended to the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev.1*.

Exception: On the trunk, revisions can be appended to the end, but not inserted.

−f[*rev*] forces a deposit; the new revision is deposited even it is not different from the preceding one.

−k[*rev*] searches the working file for keyword values to determine its revision number, creation date, author, and state (see *co(1)*), and assigns these values to the deposited revision, rather than computing them locally. A revision number given by a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **-k** option at these sites to preserve its original number, date, author, and state.

−l[*rev*] works like **-r**, except it performs an additional *co -l* for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the checkin.

−u[*rev*] works like **-l**, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after checkin.

−q[*rev*] quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless **-f** is given.

−m*msg* uses the string *msg* as the log message for all revisions checked in.

−n*name* assigns the symbolic name *name* to the number of the checked-in revision. *Ci* prints an error message if *name* is already assigned to another number.

−N*name* same as **-n**, except that it overrides a previous assignment of *name*.

−s*state* sets the state of the checked-in revision to the identifier *state*. The default is *Exp*.

−t[*txtfile*] writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, *ci* prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if **-t** is not given. The prompt is suppressed if std. input is not a terminal.

DIAGNOSTICS

For each revision, *ci* prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in, and is 0 if the operation was successful, 1 otherwise.

FILE MODES

An RCS file created by *ci* inherits the read and execute permissions from the working file. If the RCS file exists already, *ci* preserves its read and execute permissions. **Ci** always turns off all write permissions of RCS files.

FILES

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. **Ci** always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.9 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1M).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

cmp – compare two files

SYNOPSIS

cmp [**−l**] [**−s**] *file1 file2* [*offset1* [*offset2*]

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.

The *offset* arguments are positive integers that tell how many bytes to skip in each file before starting the comparison.

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.

ERRORS

Many versions of **cmp** require **−l** and **−s** to be given in this order. While this version does not require this, it is a good idea not to depend on this behavior.

Most BSD-based versions of **cmp** can handle the *offset* arguments, but they are not documented. AT&T-based versions generally forbid this (though the code is actually there to handle it).

NAME

co – check out RCS revisions

SYNOPSIS

co [*options*] *file* ...

DESCRIPTION

co retrieves revisions from RCS files. Each file name ending in ',v' is taken to be an RCS file. All other files are assumed to be working files. *co* retrieves a revision from each RCS file and stores it into the corresponding working file.

Pairs of RCS files and working files may be specified in 3 ways (see also the example section).

1) Both the RCS file and the working file are given. The RCS file name is of the form *path1/workfile*,v and the working file name is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a file name.

2) Only the RCS file is given. Then the working file is created in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix ',v'.

3) Only the working file is given. Then the name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix ',v'.

If the RCS file is omitted or specified without a path, then *co* looks for the RCS file first in the directory ./RCS and then in the current directory.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later checkin must normally be locked. Locking a revision currently locked by another user fails. (A lock may be broken with the *rcs(1)* command.) *co* with locking requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. *co* without locking is not subject to access list restrictions.

A revision is selected by number, checkin date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the *selected branch*. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number may be attached to one of the options **-l**, **-p**, **-q**, **-b**, or **-r**.

A *co* command applied to an RCS file with no revisions creates a zero-length file. *co* always performs keyword substitution (see below).

–l[*rev*] locks the checked out revision for the caller. If omitted, the checked out revision is not locked. See option **-r** for handling of the revision number *rev*.

–b[*rev*] Causes all first branches found to be followed to the end. See option **-r** for handling of the revision number *rev*.

–p[*rev*] prints the retrieved revision on the std. output rather than storing it in the working file. This option is useful when *co* is part of a pipe.

–q[*rev*] quiet mode; diagnostics are not printed.

–d*date* retrieves the latest revision on the selected branch whose checkin date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Examples of formats for *date*:

22-April-1982, 17:20-CDT,

2:25 AM, Dec. 29, 1983,
Tue-PDT, 1981, 4pm Jul 21 (free format),
Fri, April 16 15:52:25 EST 1982 (output of ctime).

Most fields in the date and time may be defaulted. *co* determines the defaults in the order year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date "20, 10:30" defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

−r[*rev*] retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *Rev* is composed of one or more numeric or symbolic fields separated by '.'. The numeric equivalent of a symbolic field is specified with the **-n** option of the commands *ci* and *rcs*.

−s*state* retrieves the latest revision on the selected branch whose state is set to *state*.

−w[*login*] retrieves the latest revision on the selected branch which was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

−j*joinlist* generates a new revision which is the join of the revisions on *joinlist*. *Joinlist* is a comma-separated list of pairs of the form *rev2:rev3,* where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial such pair, *rev1* denotes the revision selected by the options **-l**, ..., **-w**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, *co* joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* on the same branch, joining generates a new revision which is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, *co* prints a warning and includes the overlapping sections, delimited by the lines <<<<<<< *rev1,* =======, and >>>>>>> *rev3.*

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **-l** is present, the initial *rev1* is locked.

KEYWORD SUBSTITUTION

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $,* where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form *$keyword$*. On checkout, *co* replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Keywords and their corresponding values:

$Author$	The login name of the user who checked in the revision.

$Date$	The date and time the revision was checked in.
$Header$	A standard header containing the RCS file name, the revision number, the date, the author, and the state.
$Locker$	The login name of the user who locked the revision (empty if not locked).
Log	The log message supplied during checkin, preceded by a header containing the RCS file name, the revision number, the author, and the date. Existing log messages are NOT replaced. Instead, the new log message is inserted after *$Log:...$*. This is useful for accumulating a complete change log in a source file.
$Revision$	The revision number assigned to the revision.
$Source$	The full pathname of the RCS file.
$State$	The state assigned to the revision with *rcs -s* or *ci -s*.

DIAGNOSTICS

The RCS file name, the working file name, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out, and is 0 if the operation was successful, 1 otherwise.

EXAMPLES

Suppose the current directory contains a subdirectory 'RCS' with an RCS file 'io.c,v'. Then all of the following commands retrieve the latest revision from 'RCS/io.c,v' and store it into 'io.c'.

```
co io.c;   co RCS/io.c,v;   co io.c,v;
co io.c RCS/io.c,v;   co io.c io.c,v;
co RCS/io.c,v io.c;   co io.c,v io.c;
```

FILE MODES

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on, unless the file is checked out unlocked and locking is set to *strict* (see *rcs(1)*).

If a file with the name of the working file exists already and has write permission, *co* aborts the checkout if **-q** is given, or asks whether to abort if **-q** is not given. If the existing working file is not writable, it is deleted before the checkout.

FILES

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory which contains the RCS file.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.8 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1M).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

LIMITATIONS

The option **-d** gets confused in some circumstances, and accepts no date before 1970. There is no way to suppress the expansion of keywords, except by writing them differently. In nroff and troff, this is done by embedding the null-character '\&' into the keyword.

ERRORS

The option **-j** does not work for files that contain lines with a single '.'.

NAME

cobol − MIPS COBOL compiler

SYNOPSIS

cobol [option] ... file ...

DESCRIPTION

Cobol, the MIPS *ucode* cobol compiler, produces files in the following formats: MIPS object code in MIPS extended *coff* format (the normal result), binary or symbolic *ucode*, *ucode* object files and binary or symbolic assembly language. *Cobol* accepts several types of arguments:

Arguments whose names end with '.cob' are assumed to be Cobol source programs. They are compiled, and each object program is left in the file whose name consists of the last component of the source with '.o' substituted for '.cob'. The '.o' file is only deleted when a single source program is compiled and loaded all at once.

When this command results in a call to the linker the first object the linker encounters on the command line will be where execution begins when the final load module is executed.

Arguments whose names end with '.s' are assumed to be symbolic assembly language source programs. They are assembled, producing a '.o' file.

The suffixes described below primarily aid compiler development and are not generally used. Arguments that end with '.il' are assumed to be a file containing LPI intermediate code operators and its corresponding file containing the LPI intermediate code symbols is assumed to be in a file with a '.st' suffix.

Arguments whose names end with '.B', '.O', '.S', and '.M' are assumed to be binary *ucode*, produced by the front end, optimizer, ucode object file splitter and ucode merger respectively. Arguments whose names end with '.U' are assumed to be symbolic *ucode*. Arguments whose names end with '.G' are assumed to be binary assembly language, which is produced by the code generator and the symbolic to binary assembler.

Files that are assumed to be binary *ucode*, symbolic *ucode*, or binary assembly language by the suffix conventions are also assumed to have their corresponding symbol table in a file with a '.T' suffix.

The following options are interpreted by *cobol*(1). 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.

−g0 Have the compiler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages (level 1 (INFORMATIONAL) error messages).

−p0 Do not permit any profiling. This is the default. If loading happens, the standard runtime startup routine (**crt1.o**) is used, no profiling library is searched.

−p1 or **−p**
Set up for profiling by periodically sampling the value of the program counter. This option only affects the loading. When loading happens, this option replaces the standard runtime startup routine with the profiling runtime startup routine (**mcrt1.o**) and searches the level 1 profiling library (**libprof1.a**). When profiling happens, the startup routine calls *monstartup*(3) and produces a file *mon.out* that contains execution-profiling data for use with the postprocessor *prof*(l).

−O0 Turn off all optimizations.

−O1 Turn on all optimizations that can be done quickly. This is the default.

−O or **−O2**
Invoke the global *ucode* optimizer.

−feedback *file*
Used with the **−cord** option to specify *file* to be used as a feedback file. This *file* is produced by *prof*(1) with its **−feedback** option from an execution of the program produced by *pixie*(1).

−cord Run the procedure-rearranger, *cord*(1), on the resulting file after linking. The rearrangement is done to reduce the cache conflicts of the program's text. The output of *cord*(1) is left in the file specified by the **−o** *output* option or 'a.out' by default. At least one **−feedback** *file* must be specified.

−j Compile the specified source programs, and leave the *ucode* object file output in corresponding files suffixed with '.u'.

−ko *output*
Name the output file created by the ucode loader as *output*. This file is not removed. If this file is compiled, the object file is left in a file whose name consists of *output* with the suffix changed to a '.o'. If *output* has no suffix, a '.o' suffix is appended to *output.*

−k Pass options that start with a **−k** to the ucode loader. This option is used to specify ucode libraries (with **−kl***x*) and other ucode loader options.

−S Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with '.s'.

−o *output*
Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−G *num*
Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 8 bytes.

−v Print the passes as they execute with their arguments and their input and output files.

−V Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−Olimit *num*
Specify the maximum size, in basic blocks, of a routine that will be optimized by the global optimizer. If a routine has more than this number of basic blocks it will not be optimized and a message will be printed. An option specifying that the global optimizer is to be run (**−O**, **−O2**, or **−O3**) must also be specified. *Num* is assumed to be a decimal number. The default value for *num* is 500 basic blocks.

Either object file target byte ordering can be produced by *cobol*. The default target byte ordering matches the machine where the compiler is running. The options **−EB** and **−EL** specify the target byte ordering (big-endian and little-endian, respectively).

If the specified target byte ordering does not match the machine where the compiler is running, then the runtime startups and libraries come from **/usr/libeb** for big-endian runtimes on a little-endian machine and from **/usr/libel** for little-endian runtimes on a big-endian machine.

−EB Produce object files targeted for big-endian byte ordering.

−EL Produce object files targeted for little-endian byte ordering.

The following options are specific to *cobol*:

−defext
Allows the use of external data. This is required in programs where external data are defined.

−dline Compiles all source lines having a 'D' in the indicator area (column 7). If this option is not specified, all source lines with a 'D' in the indicator area are treated as comment lines.

−f *n* Flags all items in the source program that exceed the Federal Information Processing Standard (FIPS) Level specified by *n*, where *n* stands for one of the following:

1	FIPS Low Level
2	FIPS Low-Intermediate Level
3	FIPS High-Intermediate Level
4	FIPS High Level

−fsc74 Turns off the default ANSI-85 status codes and generates ANSI-74 status codes.

−l *[listing]*
Produces a compiler listing file with a suffix '.l'. If *listing* is specified, the listing file is named by it. This option is only recognized by the cobol front-end; it must be used in conjunction with the -Wf option.

−supp_cob85
Removes the additional ANSI-85 reserved words from the compiler's reserved word table, freeing them for use as user names.

−supp_cod
Removes the supplemental CODASYL reserved words from the compiler's reserved word table, freeing them for use as user names.

−comp_trunc
Truncates values in COMPUTATIONAL data items.

−ansi Turns off the extensions to the ACCEPT and DISPLAY statements.

−lpilock
Specifies LPI record locking.

−nolock
Suppresses record locking.

The option described below is primarily used to provide UNIX compilation environments other than the native compilation environment.

−systype *name*
Use the named compilation environment *name*. See *compilation*(7) for the compilation environments that are supported and their *name*s. This has the effect of changing the standard directory for '#include' files, the runtime libraries and where runtime libraries are searched for. The new items are located in their usual paths but with */name* prepended to their paths.

The options described below primarily aid compiler development and are not generally used:

−H*c* Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be [**fkjusmoca**]. It selects the compiler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T' and is not removed.

−K Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.B' file for binary *ucode,* produced by the front end). These intermediate files are never removed, even when a pass encounters a fatal error. When ucode linking is performed and the **−K** option is specified the base name of the files created after the ucode link is 'u.out' by default. If **−ko** *output* is specified, the base name of the object file is *output* without the suffix if it exists or suffixes are appended to *output* if it has no suffix.

−# Converts binary *ucode* files ('.B') or optimized binary ucode files ('.O') to symbolic *ucode* (a '.U' file) using *btou*(1).

−W*c[c...],arg1[,arg2...]*
Pass the argument[s] *argi* to the compiler pass[es] *c[c..]*. The *c's* are one of [**fkjusmocablyz**]. The c's selects the compiler pass in the same way as the **−t** option.

The options **−t[fkjusmocablyzrCSO1EMnt]**, **−h***path,* and **−B***string* select a name to use for a particular pass, startup routine, or standard library. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

The **−EB** or **−EL** options, the **−p[01]** options and the **−systype** option must precede all **−B** options because they can affect the location of runtimes and what runtimes are used.

−t[fkjusmocablyzrCSO1EMnt]
Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
cobfe	f
ulpi	k
ujoin	j
uld	u
usplit	s
umerge	m
uopt	o
ugen	c
as0	a
as1	b
ld	l
ftoc	y

cord	z
[m]crt[1n].o	r
libcob.a	C
libisam.a	S
libsort.a	O
libpl1.a	1
libexc.a	E
libm.a	M
libprof1.a	n
btou, utob	t

−h*path*

Use *path* rather than the directory where the name is normally found.

−B*string*

Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B,** the **−t** option is assumed to be "fkjusmocablyzrCSO1EMnt". This list designates all names. If no **−t** argument has been processed before the **−B** then a **−B***string* is passed to the loader to use with its **−l***x* arguments.

Invoking the compiler with a name of the form **cobol***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/.** If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for library names rather than the default **/.** This affects the standard library, /usr/lib/libc.a. If this is set, the first directory that is searched for libraries, using the **−l***x* option, is COMP_TARGET_ROOT/usr/lib/cmplrs/cc. The standard directories for libraries are then searched, see *ld*(1).

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/ .**

If the environment variable RLS_ID_OBJECT is set, the value is used as the name of an object to link in if a link takes place. This is used to add release identification information to objects. It is always the last object specified to the loader. See *rls_id*(1) for the tools to create this information.

Other arguments are assumed to be either loader options or *cobol*-compatible object files, typically produced by an earlier *cobol* run, or perhaps libraries of *cobol*-compatible routines. These files, together with the results of any compilations specified, are loaded in the order given, producing an executable program with the default name **a.out.**

FILES

file.cob	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/usr/lib/cobfe	Cobol front end
/usr/lib/ulpi	LPI intermediate code to ucode translator
/usr/lib/ujoin	binary ucode and symbol table joiner
/usr/bin/uld	ucode loader
/usr/lib/usplit	binary ucode and symbol table splitter
/usr/lib/umerge	procedure integrator
/usr/lib/uopt	optional global ucode optimizer
/usr/lib/ugen	code generator

/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/lib/crt1.o	runtime startup
/usr/lib/crtn.o	runtime startup
/usr/lib/mcrt1.o	startup for profiling
/usr/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libtermcap.a	terminal capabilities library, see *termcap*(3X)
/usr/lib/libprof1.a	level 1 profiling library
/usr/lib/libcob.a	Cobol library
/usr/lib/libsort.a	Sort library
/usr/lib/libisam.a	Indexed sequential access method library
/usr/lib/libpl1.a	PL/I library
/usr/lib/libexc.a	exception library
/usr/lib/libm.a	math library
/usr/bin/ld	MIPS loader
/usr/lib/ftoc	interface between *prof*(1) and *cord*(1)
/usr/lib/cord	procedure-rearranger
/usr/bin/btou	binary to symbolic ucode translator
/usr/bin/utob	symbolic to binary ucode translator
mon.out	file produced for analysis by *prof*(1)

Runtime startups and libraries for the opposite byte sex of machine the compiler is running on have the same names but are located in different directories. For big-endian runtimes on a little-endian machine the directory is /usr/libeb and for little-endian runtimes on a big-endian machine the directory is /usr/libel.

SEE ALSO

monstartup(3), prof(1), ld(1), dbx(1), what(1), cord(1), pixie(1), ftoc(1)

DIAGNOSTICS

The diagnostics produced by *cobol* are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

NOTES

The standard library, /usr/lib/libc.a, and the terminal capabilities library, /usr/lib/libtermcap.a, are loaded by using the -lc and -ltermcap loader options and not full path names. The wrong ones could be loaded if there are files with the name libc.a*string* or libtermcap.a*string* in the directories specified with the **−L** loader option or in the default directories searched by the loader.

The handling of libc.a is confusing.

NAME

col - filter reverse line-feeds

SYNOPSIS

col [**−b**] [**−f**] [**−x**] [**−p**]

DESCRIPTION

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

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

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

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

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

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

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

SEE ALSO

nroff(1), tbl(1) in the *DOCUMENTER's WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual* .

NOTES

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

ERRORS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.
Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.

NAME

comb – combine SCCS deltas

SYNOPSIS

comb [**-o**] [**-s**] [**-p** *sid*] [**-c** *list*] *files*

DESCRIPTION

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

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

-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.

-pSID 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.

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), sh(1).
sccsfile(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Use *help(1)* for explanations.

ERRORS

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

NAME

comm - select or reject lines common to two sorted files

SYNOPSIS

comm [**–** [**123**]] *file1 file2*

DESCRIPTION

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

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus *comm* **–12** prints only the lines common to the two files; *comm* **–23** prints only lines in the first file but not in the second; *comm* **–123** prints nothing.

SEE ALSO

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

NAME

cord – rearranges procedures in an executable file to facilitate better cache mapping.

SYNOPSIS

cord [**-v**] [**-o** *outfile*] [**-f**] [**-c** *cachesize*] [**-p** *maxphases*] *obj_file reorder_file*

DESCRIPTION

The **cord** command rearranges procedures in an exectable object file to maximize efficiency in a machine's cache. By rearranging the procedures properly, we end up reducing the instruction cache miss rates. **Cord** does not attempt to determine the correct ordering, but is given a *reorder* file containing the desired procedure order. The reorder file is generated by the *ftoc* program, which in turn generates a *reorder* file from a set of profile feedback files (see *prof(1)*).

Processed lines in the *reorder* file are called procedure lines. Each procedure line must be on a separate source line. Each procedure line must contain the source name of the file, followed by a blank followed by a qualified procedure name. Nested procedures must be qualified x.y where x is the outer procedure. A newline or blank can follow the procedure name:

foo.c bar (everything else following is ignored)

Lines beginning with # are comments, lines beginning with $ are considered **cord** directive lines. The only directive currently understood is $*phase*. This directive will consider the rest of the file (until the end of file or next $*phase*) as a new phase of the program and will order the procedures accordingly. A procedure may appear in more than one phase, resulting in more than one copy of it in the final binary. First, **cord** will try to relocate procedure references to a copy of the procedure belonging to the requesting phase; otherwise it will relocate the references to a random copy.

We suggest you use the **-cord** option to a compiler driver like *cc(1)* rather than execute cord directly. **Cord** options can be specified with *-Wz,cordarg0,cordarg1,...*. If you have to run **cord** by hand, you may want to run it once with the driver using the **-v** flag on a simple program. This will enable you to see the exact passes and the arguments involved in using **cord**.

Obj is an executable object file with its relocation information intact. This can be achieved by passing the *-r -z -d* options to the linker, *ld(1)*. The linker option *-r* maintains relocation information in the object file, but will not make it a *ZMAGIC* file (hence *-z*). It also will not allocate common variables (hence *-d*) as it would without the option.

WARNING: Since **cord** works from an input list of procedures generated from profile output, the resulting binary is data dependent. In other words, it may only preform well on the same input data that generated the profile information, and may preform worse than the original binary on other data. Furthermore, if the hot areas in the cache don't fit well into one cachepage, performance can degrade.

The **cord** command accepts these options:

-v Print verbose information. This includes listing those procedures considered part of other procedures and cannot be rearranged (these are basically assembler procedures that may contain relative branches to other procedures rather than relocatable ones). The listing also lists those procedures in the flipped area (if any) and a mapping of old location to new.

-f Flip the first cachepage size procedures. The assumption when cord was written was that procedures would be reordered by procedure density (cycles/byte). This option ensures that the densest part of each page following the first cachepage would conflict with the least-dense part of the first cachepage.

-c cachesize
Specify the cachesize (in bytes) of the machine on which you want to execute. This only affects the **-f** option. If not specified, 65536 is used.

-o outputfile
Specifies the output file. If not specified, *a.out* is used.

-p phasemax
specifies the maximum number phases allowed. The default is 20.

SEE ALSO

prof(1), ftoc(1), cc(1), ld(1), MIPS Languages Programmer Guide

NAME

cord2 – rearranges basic blocks in an executable file to facilitate better cache mapping.

SYNOPSIS

cord2 [-v] [**-o** *outfile]* [**-c** *cachewords*] [**-d**] [**-b** *bridge_limit*] [**-n**] [-A *addersfile*] [[**-C** *countsfile*] **...**] *obj*

DESCRIPTION

The **cord2** command extracts basic blocks from a program and deposits them in a new area in the text, making jumps from and to that area as necessary. By separating the basic blocks, you can reduce instruction cache miss rates. **Cord2** takes the output of a *pixie* profiling run as input (see *pixie(1))*.

Obj is an executable object file. **Cord2** only requires one *addersfile*; it will create the filename by appending *.Bbaddrs* to the *obj* filename if none is specified with **-A**. Many counts files can be specified from many runs with multiple **-C** arguments; if none is specified **cord2** will create the counts filename by appending *.Counts* to the *obj* name. Multiple counts files will be added together into an internal counts array represented with C double-type elements. The counts array elements contain the density of a block or cycles/byte. If you specify **-n**, then the counts are normalized so that each counts array entry is cycles/totalcycles. When one counts is specified, the default is to favor small blocks; **-n** negates that. When many counts files are specified, **-n** also negates favoring one counts file. This is because its totalcycles may exceed the totalcycles of another counts file.

Cord2 determines which basic blocks to insert by sorting the counts array and collecting the blocks with the highest counts that will fit into the new area. **Cord2** may skip over huge blocks that won't fit at the end of the new area.

Once the blocks are determined, they are inserted into the new area, and their original location is modified to jump to the new area. At the end of each block in the new area, a jump is added back to the original block's subsequent or fall-through location, and the branch/jump target (if necessary). Both entering and exiting the new area is optimized to take advantage of other blocks also in the new area, and jump delay slots.

Many times there may be one or more fall-through blocks of a block in the new area which are 1) small, 2) hardly ever used, and 3) not in the new area. If the block following these fall-through blocks is in the new area, the fall-through blocks are called bridge blocks. It may be more costly to generate jumps to and from bridge blocks rather than to just copy them. **Cord2** allows you to specify that bridge blocks be added to the new area if they total less than the *bridge_limit* instructions between two new-area blocks. You may specify the *bridge_limit* with **-b**; the default is zero. Bridge blocks may bump blocks out of the new area that might normally fit into it.

WARNING: Since **cord2** works from profile output, the resulting binary is data dependent. In other words, it may perform well only on the same input data that generated the profile information, and may perform worse than the original binary on other data. Furthermore, if the hot areas in the cache don't fit well into one cachepage, performance can degrade.

The **cord2** command also accepts these options:

-d Fill the delay slots with nops only when adding jumps to and from the new area.

-v Print verbose information. This includes statistics about the **cord2** process.

-v -v Print all of the **-v** information but include detailed dissamblies of the code moved, changed and generated by **cord2**.

-c cachewords
Specify the number of words in the cache of the machine on which you want

to execute. This will actually be the size of the new area. *Cachesize* may be a misnomer, as you can specify a size other than your machine's cache size; however, it is probably the correct number.

-o outputfile
Specifies the output file. If not specifled, *a.out.cord2* is used.

BUGS

Cord2 adds the new area to the end of text so any program using the *etext* (see *ld(1)*) symbol may not work.

SEE ALSO

pixie(1), cord(1), MIPS Languages Programmer Guide

NAME

cp, ln, mv – copy, link or move files

SYNOPSIS

cp [**−p**] *file1 file2*
cp [**−p**] *file1* [*file2...*] *directory*
ln [**−sf**] *file1 file2*
ln [**−sf**] *file1* [*file2*] *directory*
mv [**−fp**] *file-1 file2*
mv [**−fp**] *file1* [*file2*] *directory*
mv [**−fp**] *directory1 directory2*

DESCRIPTION

The source file(s) is(are) copied (linked, moved) to the target file or directory. Under no circumstance can the source file and the corresponding target be the same (take care when using *sh(1)* metacharacters). If the target is a directory, then one or more files are copied (linked, moved) to that directory. Otherwise, its contents are destroyed.

If **mv** or **ln** determines that the mode of target forbids writing, it will print the mode (see *chmod(2)*), ask for a response, and read the standard input for one line; if the line begins with **y**, the **mv** or **ln** occurs, if permissable; if not, the command exits. When the **−f** option is used or if the standard input is not a terminal, no questions are asked and the **mv** or **ln** is done. **ln −s** creates symbolic links.

Only **mv** will allow the source to be a directory, in which case the directory rename will occur only if the two directories have the same parent. If the source file is a file and the target is a link to another file with links, the other links remain and the target becomes a new file.

When using **cp**, if the target is not a file, a new file is created which has the same mode as the source file except that the sticky bit is not set unless you are super-user; the owner and group of the target are those of the user. If the target is a file, copying a file into the target does not change its mode, owner, nor group. The last modification time of the target (and last access time, if the target did not exist) and the last access time of the source file are set to the time the copy was made. If the target is a link to a file, all links remain and the file is changed.

SEE ALSO

chmod(1), cpio(1), rm(1).

WARNINGS

ln will not link across file systems. This restriction is necessary because file systems can be added and removed.

ERRORS

If the source file and the target lie on different file systems, **mv** must copy the file and delete the original. In this case any linking relationship with other files is lost.

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio −o[acvBHL] < *name-list* > *collection*
cpio −o[acvBHL] −O*collection* < *name-list*
cpio −i[bcdmrstuvfB6HLS] [*pattern* ...] < *collection*
cpio −i[bcdmrstuvfB6HLS] −I*collection* [*pattern* ...]
cpio −p[adlmruvHL] *directory* < *name-list*

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. Output is padded to a 512-byte boundary.

cpio −i (copy in) extracts files from the standard input, which is assumed to be the product of a previous *cpio −o*. Only files with names that match *patterns* are selected. *patterns* are regular expressions 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). Each *pattern* should be surrounded by double quotes. The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous *cpio −o*. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous *cpio −o*. NOTE: If *cpio -i* tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **-u** option can be used to unconditionally overwrite the existing file.)

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. Access times are not reset for linked files when *cpio -pla* is specified.

−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 a character special device, e.g. */dev/rmt/0m*).

−d *Directories* are to be created as needed.

−c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

−r Interactively *rename* files. If the user types a null line, the file is skipped. (Not available with *cpio -p*.)

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

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

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

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

−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 within each half word. Use only with the **−i** option.

−S *swap* halfwords within each word. Use only with the **−i** option.

−b Reverses the order of the *bytes* within each word. Use only with the **−i** option.

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

−H Do not follow symbolic links (default). Symbolic link records are saved in the archive to be extracted on the other side. This is not portable to all system types.

−L Follow symbolic links, placing in the archive records for the files they point to.

−O*collection*
Specify the name of the output device. By specifying the device this way instead of using redirection, the prompts for change of media use this as a default.

−I*collection*
Specify the name of the input device. By specifying the device this way instead of using redirection, the prompts for change of media use this as a default.

NOTE: *cpio* assumes four-byte words.

If *cpio* reaches end of medium (end of a diskette for example), when writing to (**-o**) or reading from (**-i**) a character special device, *cpio* will print the message:

If you want to go on, type device/file name when ready.

To continue, you must replace the medium and type the character special device name (/dev/rdiskette for example) and carriage return, unless the **−I** or **−O** option was used, in which case you will be prompted with a default name, which may be changed by typing a new one, accepted by typing return, or you may abort the job by typing a 'q'. You may want to continue by directing *cpio* to use a different device. For example, if you have two floppy drives you may want to switch between them so *cpio* can proceed while you are changing the floppies. (A carriage return alone causes the *cpio* process to exit.)

EXAMPLES

The following examples show three uses of *cpio*.

When standard input is directed through a pipe to *cpio −o,* it groups the files so they can be directed (>) to a single file (*../newfile*). Instead of *ls*, you could use *find*, *echo*, *cat*, etc. to pipe a list of names to cpio. You could direct the output to a device instead of a file.

ls | cpio −o >*../newfile*

cpio −i uses the output file of *cpio −o* (directed through a pipe with cat in the example), takes out those files that match the patterns (*memo/a1, memo/b**), creates directories below the current directory as needed (**-d** option), and places the files in the appropriate directories. If no patterns were given, all files from "newfile" would be placed in the directory.

cat newfile | cpio −id "*memo/a1*" "*memo/b**"

cpio −p takes the file names piped to it and copies or links (**-l** option) those files to another directory on your machine (*newdir* in the example). The **-d** option says to create directories as needed. The **-m** option says retain the modification time. (It is important to use the *-depth* option of *find* to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.)

find . −depth −print | cpio −pdlmv *newdir*

SEE ALSO

ar(1), find(1), ls(1), tar(1).
cpio(4) in the *Programmer's Reference Manual*.

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.

4) The device/inode pair appearing in the headers is created by *cpio* as one 32-bit number, and has no correlation to the real device/inode numbers of the file. The *cpio* number begins with the number 3 and increments sequentually for each file processed by *cpio*.

NAME

cpp – the C language preprocessor

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

−U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes: None of these are defined by *cpp*. Instead, the compiler drivers, *cc(1)*, *as*(1), *pc(1)*, and *f77(1)* define these symbols.

operating system:	unix, ibm, gcos, os, tss, dmert
target hardware:	mips, interdata, pdp11, u370, u3b, u3b5, u3b2, u3b20d, vax
host hardware:	host_mips
languages:	LANGUAGE_C, LANGUAGE_ASSEMBLY, LANGUAGE_PASCAL, LANGUAGE_FORTRAN
UNIX system variant:	RES, RT
lint(1):	lint

−D*name*
−D*name*=*def*
Define *name* as if by a **#define** directive. If no =*def* is given, *name* is defined as 1. The **−D** option has lower precedence than the **−U** option. That is, if the same name is used in both a **−U** option and a **−D** option, the name will be undefined regardless of the order of the options.

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

−I This option changes the algorithm for searching for **#include** files to never look in the standard list.

−M Print, one per line on standard output; the path names of included files. Each is prefixed with *ifile*'s last component name with the suffix changed to '.o' followed by a

':' and a space (for example "hello.o: /usr/include/stdio.h").

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

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

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

#define *name*(*arg*, ..., *arg*) *token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

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

#ident "*string*"
This directive is recognized for compatibility but ignored.

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

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

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

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

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

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

To test whether either of two symbols, *foo* and *fum*, are defined, use

```
#if defined(foo) || defined(fum)
```

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

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

FILES

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

SEE ALSO

cc(1), as(1), pc(1), f77(1), m4(1)

DIAGNOSTICS

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

NOTES

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

NAME

crontab – user crontab file

SYNOPSIS

crontab [file]
crontab −r
crontab −l

DESCRIPTION

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

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

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

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

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

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

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

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

FILES

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

special root file

SEE ALSO

sh(1).
cron(1M) in the *System Administrator's Reference Manual*.

WARNINGS

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

A special **root** file called **periodic** exists in the */usr/spool/cron/crontabs* directory. Do not attempt to update this file using the *crontab* command. If this is done, the **root** *crontab* file will actually be overwritten. This special file can only be submitted to *cron* at initialization time.

NAME

crypt – encode/decode

SYNOPSIS

crypt [*password*]
crypt [**−k**]

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 argument is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. If the **−k** option is used, *crypt* will use the key assigned to the environment variable CRYPTKEY. *crypt* encrypts and decrypts with the same key:

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

Files encrypted by *crypt* are compatible with those treated by the editors *ed(1), edit(1), ex(1),* and *vi(1)* in encryption mode.

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

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

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

If the key is an argument to the *crypt* command, it is potentially visible to users executing *ps(1)* or a derivative. The choice of keys and key security are the most vulnerable aspect of **crypt**.

FILES

/dev/tty	for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), ps(1), stty(1), vi(1).

WARNING

This command is provided with the Security Administration Utilities, which is only available in the United States. If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

ERRORS

If output is piped to *nroff* and the encryption key is *not* given on the command line, *crypt* can leave terminal modes in a strange state (see *stty(1)*).

NAME

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

SYNOPSIS

csh [**−cefinstvVxX**] [*arg* ...]

DESCRIPTION

csh is a command language interpreter incorporating a history mechanism (see History substitutions) and a C-like syntax.

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

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

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

When a login shell terminates, it executes commands from the file *.logout* in the user's home directory.

Lexical Structure

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

Strings enclosed in matched pairs of quotation marks, either single or double quotation marks, form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Such quotations have semantics to be described subsequently.

Within pairs of single or double quotation marks, a newline (carriage return) preceded by a \ gives a true newline character. This is used to set up a file of strings separated by newlines, as for *fgrep*.

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

Commands

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

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

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

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

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

```
1234
```

This line indicates that the process which was started asynchronously was number 1234.

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

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

Command Line Editing
The line editor permits a large number of operations beyond the scope of the current tty driver – most of the simple editing commands available in the EMACS screen editor (not available in 4.3) may be used to move around on and change the current line. In addition, line editing allows interactive expansion of *csh* history items. Typing "!foo" followed by a space, for example, will insert the previous command starting with "foo" into the line at the current location.

The line editing feature, which is off by default, may be enabled by setting the shell variable “lineedit”. (The variable "lineedit" takes precedence over the variable "filec" and consequently disables file name completion, though file name completion is still available under the line editor by using M-ESC, which by default means typing two escape characters.) The variable "lineeditmin" specifies the minimum size of history list commands that will be seen by the line editor. The variable "lineeditchars" gives a character map which allows the default assignment of the keys to be changed. In order to use the history mechanism, you must also set the variable “history” to be the number of previous lines you want remembered.

With “lineedit” set to the empty string, the line editor works on any CRT terminal which meets the following requirements: (1) ASCII linefeed moves the cursor downward (2) ASCII backspace moves the cursor one column to the left without erasing the character in that column (3) ASCII carriage return moves the cursor to the left margin (4) ASCII bell character rings the bell, and (5) ASCII space character replaces the character in the current column with a blank space and moves the cursor one column to the right. By setting “lineedit” to a string beginning with a delimiter and containing character sequences separated by that delimiter, you can customize the line editor for terminals which do not meet these requirements. For example, “set lineedit="/^j/^h/^m/^g/ /"” (where you may use the prefix ’^’ to indicate a control character) is equivalent to the default.

The special characters you set using the *stty(1)* command are still in effect *between* commands. If, for example, you set your Unix QUIT character to be DEL, you can use DEL to get yourself out of a program like *mail(1)*, or to interrupt a program like *vi* , or even to break out of a shell “while” loop. But while you are actually editing a line, DEL has a special meaning described below.

The infallible way to get out of editing is to type ^C^D. Immediately after the prompt, ^D by itself is sufficient.

The line editor maintains a repetition factor which is initially 1. This factor is multiplied by 4 by the ^U command. In the following description, (R) indicates that the command pays attention to the repetition factor. The repetition factor returns to 1 after each command or error, even if the command does not pay attention to the repetition factor.

^@	Mark line (this lets you browse through the remembered lines with ^P and ^N, mark one, and later use the marked line with M-Y).
^A	Move cursor to beginning of line.

^B	Move cursor backward (R).
^C	Clear the entire line and reprompt.
^D	Delete character above cursor. (If you type ^D immediately after the prompt, before you type anything else, it has its usual meaning: quit running the shell). (R)
^E	Move cursor to end of line.
^F	Move cursor forward (R).
^G	Abort the current command and ring the bell.
^H	Delete character preceding cursor (R).
^J	Activate the line.
^K<char>	Delete characters until cursor is under <char>. If <char> is ^K, use the same char as the previous ^R, ^S, or ^K command. To delete until ^K, say "^K^Q^K". To delete until end of line, say "^K^M".
^L	Redisplay line on a clean line.
^M	Activate the line.
^N	Go forward one line in the queue of previously-typed lines and make that be the "line under construction". Only lines greater in size than the shell variable "lineeditmin" are considered. (R)
^P	Go backward one line in the queue of previously-typed lines and make that be the "line under construction". Only lines greater in size than the shell variable "lineeditmin" are considered. (R)
^Q<char>	Insert <char> before the cursor (useful for quoting characters which the line editor itself would otherwise recognize).
^R<char>	Search backward for <char>. If <char> is ^R, then it searches instead for the same character as the previous ^K, ^R or ^S command. To search for ^R, type "^R^Q^R". (R)
^S<char>	Search forward for <char>. If <char> is ^S, then it searches instead for the same character as the previous ^K, ^R or ^S command. To search for ^S, type "^S^Q^S". (R)
^T	Interchange the two characters preceding the cursor.
^U	Multiply the repetition count by 4. Does not take a numeric argument.
^W	Delete the entire line.
^Y	(Yank) Insert in front of the cursor the previous text deleted with ^K, M-D, M-H, or M-DEL.
RUBOUT	Delete character preceding cursor. (R)

Other control characters are illegal, and most send a bell character to your terminal to try to make it beep.

There are also a few meta-commands, which you can invoke by typing ASCII "escape" before the letter.

M-A	Go to the beginning (bottom or most recent) of the history list.
M-B	Move backward by one word. (R)
M-D	Delete next word. (R)
M-E	Go to the end (top or earliest) of the history list.

M-F	Move forward by one word. (R)
M-H	Delete previous word. (R)
M-U	Undo the last non-trivial change.
M-Y	(Yank) Insert in front of the cursor the line marked with ^@.
M-DEL	Delete previous word. (R)
M-ESC	Complete listings with "ls" style output. Same as entering Control-D and ESC in file completion mode.

All ordinary (non-control, non-meta) characters insert themselves before the cursor. Thus to add characters to a line, simply type them.

If you set the shell variable "lineeditmin" to a positive integer, the line editor will no longer consider lines shorter than that number of characters in length. Thus you can prevent ^N and ^P from showing you trivial lines like *vi* or "popd".

The interactive history expansion mechanism is invoked by typing a space or a tab after a word containing the current history character (which defaults to "!"). Any history expansion involving just full commands and arguments thereof (no editing) will be done interactively. In addition to **csh**'s normal "!foo:i-j" (where "i" and "j" are numbers and either may be elided), the line editor also allows either of "i" or "j" to be referenced from the end of the argument list, as in "!foo:$-2-$" which yields the last three arguments of the previous command starting with "foo".

The variable "lineeditchars" may be set to change the default functions for each key, but use great care in doing so! The default value of this variable is:

"^@^a^b^c^d^e^f^g^h^i^j^k^l^m^n^o^p^q^r^s^t^u^v^w^x^y^z^[^\^]^^^_^?".

In this string, control characters are specified by preceding them with an uparrow ("^"), and meta characters are specified by prefixing them with a dollar sign ("$"). In addition, the delete character may be specified as "^?". Each position in this string corresponds to one of the control characters – thus position 0 corresponds to the function for ^@, position 1 to the value for ^A, etc. In addition to the 32 control positions, position 33 controls the function of DEL. The value in a position is the default function binding to be used for that character. Thus, to change the bindings so that ^W does a word delete (M-H), ^X is the line kill character (^W), and DEL is the interrupt character (^C), set lineeditchars to:

"^@^a^b^c^d^e^f^g^h^i^j^k^l^m^n^o^p^q^r^s^t^u^v$h^w^y^z^[^\^]^^^_^c".

Substitutions

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

History substitutions

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

History substitutions begin with the character ^ and may begin anywhere in the input stream (with the proviso that they do not nest.)

This ! may be preceded by a \ to turn off its special meaning; for convenience, a ! is also passed unchanged when it is followed by a blank, tab, newline, = or (.

Therefore, do not put a space after the ! and the command reference when you are invoking the shell's history mechanism. (History substitutions also occur when an input line begins with ^. This special abbreviation will be described later.)

An input line which invokes history substitution is echoed on the terminal before it is executed, as it would look if typed out in full.

The shell's history list, which may be seen by typing the *history* command, contains all commands input from the terminal which consist of one or more words. History substitutions reintroduce sequences of words from these saved commands into the input stream. The *history* variable controls the size of the input stream. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the *history* command:

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

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an ! in the prompt string. This is done by setting prompt = ! and the prompt character of your choice.

For example, if the current event is number 13, we can call up the command recorded as event 11 in several ways: *!-2* [i.e., 13-2]; by the first letter of one of its command words, such as *!c* referring to the c in *cat*; or *!wri* for event 9, or by a string contained in a word in the command as in *!?mic?* also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case *!!* refers to the previous command; thus *!!* alone is essentially a *redo*.

Words are selected from a command event and acted upon according to the following formula:

event:position:action

The *event* is the command you wish to retrieve. As mentioned above, it may be summoned up by event number and in several other ways. All that the *event* notation does is to tell the shell which command you have in mind.

Position picks out the words from the command event on which you want the *action* to take place. The *position* notation can do anything from altering the command completely to making some very minor substitution, depending on which words from the command event you specify with the *position* notation.

To select words from a command event, follow the event specification with a : and a designator (by position) for the desired words.

The words of a command event are picked out by their position in the input line. Positions are numbered from 0, the first word (usually command) being position 0, the second word having position 1, and so forth. If you designate a word from the command event by stating its position, means you want to include it in your revised command. All the words that you want to include in a revised command must be designated by position notation in order to be included.

The basic position designators are:

0	first (command) word
n	*n*th argument
^	first argument, i.e., 1
$	last argument
%	matches the word of an *?s?* search which immediately precedes it; used to strip one word out of a command event for use in another command. Example: *!?four?:%:p* prints *four*.
x−y	range of words (e.g., 1-3 means from position 1 to position 3).
−y	abbreviates 0−*y*
*	stands for ^−$, or indicates position 1 if only one word in event.
x *	abbreviates *x*−$ where *x* is a position number.
x−	like *x** but omitting last word $

The : separating the event specification from the word designator can be omitted if the argument selector begins with a ^, $, *, − or %.

Modifiers, each preceded by a :, may be used to act on the designated words in the specified command event. The following modifiers are defined:

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

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

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

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

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

Quotations with ' and "

The quotation of strings by ' and " can be used to prevent all or some of the remaining substitutions which would otherwise take place if these characters were interpreted as metacharacters or wild card matching characters. Strings enclosed in single quotes, ' are prevented any further interpretation or expansion. Strings enclosed in " may still be variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see Command Substitution below) does a " quoted string yield parts of more than one word;

Alias substitution

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

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

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

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

Variable substitution

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

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

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

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

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

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

Metasequences for variable substitution

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

$name
${name}

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).

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

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

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a [*selector*].

$0

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

$number
${number}

Equivalent to $argv [number].

$∗

Equivalent to $argv [∗]".

The modifiers :h, :t, :r, :q and :x may be applied to the substitutions above as may :gh, :gt and :gr. If braces { } appear in the command form, then the modifiers must appear within the braces. The current implementation allows only one : modifier on each $ expansion.

The following substitutions may not be modified with : modifiers.

$?name
${?name}

Substitutes the string 1 if name is set, 0 if it is not.

$?0

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

$$

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

$<

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

Command substitution

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

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

Filename substitution

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

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

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

The metanotation *a{b,c,d}e* is a shorthand for *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, it is truncated, its previous contents being lost.
>
> If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g., a terminal or */dev/null*) or an error results. This helps prevent accidental destruction of files. In this case the ! forms can be used and suppress this check.
>
> The forms involving &, route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as < input filenames are.

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

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

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& rather than just |. To redirect standard output and standard error to separate files, use (cmd > file1) >& file2; */dev/tty* may be used to redirect input or output to or from your terminal.

Expressions

A number of the built-in 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* (which may contain *, ? 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.

Command executions can be used as primitive operands in expressions. When used in an expression, the command is enclosed in { and }, e.g., {command}. 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.

File enquiries can also be used as primitive operands in expressions. They should be 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
c	character special file
b	block special file
p	named pipe (fifo)
u	set-user-ID bit is set
g	set-group-ID bit is set
k	sticky bit is set
s	size greater than zero
t	open file descriptor for terminal device

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.

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.)

Built-in Commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last, then it is executed in a subshell.

alias
alias name
alias name wordlist

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

break

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

breaksw

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

case label:

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

cd
cd name
chdir
chdir name

Change the shell's 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
dirs −l

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory. In the first form the user's home directory is represented by ~.

echo wordlist
echo −n wordlist

The specified words are written to the shell's standard output, separated by spaces, and terminated with a newline unless the *−n* option or the \c escape is specified. The following C-like escape sequences are available:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab

\\ backslash
\n the character whose ASCII code is the 1-, 2- or 3-digit octal number *n*.

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 in the context of the current shell. 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).

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

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

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

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

history
history n
history −r n
history −h 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. The *−h* option causes the history list to be printed without leading numbers. This is used to produce files suitable for *source*ing using the −h option to *source*.

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*.)

kill pid
kill −sig pid ...

Sends either the TERM (terminate) signal or the specified signal to the specified processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix SIG). There is no default, saying just "kill" does not send a signal to the current process.

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.

nice
nice +number
nice command
nice +number command

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

nohup
nohup command

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

onintr
onintr −
onintr label

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

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

popd
popd +n
　Pops the directory stack, returning to the new top directory. With an 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 *indexth* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

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

setenv name value
　Sets the value of environment variable *name* to be *value*, a single string. The variables PATH, USER, LOGNAME, HOME, and TERM are automatically imported to and exported from the *csh* variables *path*, *user*, *logname*, *home*, and *term*, respectively; 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
source −*h* 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. Normally input during *source* commands is not placed on the history list; the -h option causes the commands to be placed in the history list without being executed.

switch (string)
case str1:
    ...
    breaksw
...
default:
    ...
    breaksw
endsw

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

time
time command

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

umask
umask value

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

unalias pattern

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

unhash

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

unset pattern

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

unsetenv pattern

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

wait

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

while (expr)

...

end

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

%

% user

The first form toggles the user ID and group ID between that of *root* and *user* for all executed commands (except built-ins). The prompt is automatically toggled between # and #%. The second form specifies a user name, listed in */etc/passwd*, that should be toggled to and from.

@

@ 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 HOME into *home*, and copies it back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv	Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e., $1 is replaced by $argv[1], etc.
cdpath	Gives a list of alternate directories searched to find subdirectories in *chdir* commands.
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-built-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.
histchars	Can be given a string value to change the characters used in history substitution.

The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

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 CTRL-ds.

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

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

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

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

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

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

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable, then only full path names will execute. The usual search path is ., */bin* and */usr/bin*, but this may vary from system to system. For the super-user the default search path is */bin, /usr/bin, /etc.* 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. The sequence \\ is replaced with a single \. The prompt should only be set by the user if it is already defined so that it will not be printed when processing shell scripts by using the statement

```
if ( $?prompt ) set prompt='\!% '
```

If the sequence \@**x** appears, where **x** is one of the characters listed below, then it will be replaced by the current time and date in the indicated format.

R　time as HH:MM AM/PM, e.g. 8:40PM

r	time as HH:MM:SS AM/PM, e.g. 08:40:25 PM
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	day of year – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday – Sun to Sat
h	abbreviated month – Jan to Dec
n	insert a new-line character
t	insert a tab character

The default prompt is %, or # for the super-user.

savehist is given a numeric value to control the number of entries of the history list that are saved in ~/.history when the user logs out. Any command which has been referenced in this many events will be saved. During start up the shell sources ~/.history into the history list enabling history to be saved across logins. Too large values of *savehist* will slow down the shell during start up.

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-built-in Command Execution below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status 1, all other built-in 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-built-in Command Execution

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

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

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

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

Argument List Processing

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

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

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

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

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

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

–s　Command input is taken from the standard input.

–t　A single line of input is read and executed. A \ may be used to escape the newline at the end of this line and continue onto another line.

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

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

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

–X　Is to *–x* as *–V* is to *–v*.

After processing of flag arguments, if arguments remain but none of the *–c*, *–i*, *–s*, or *–t* options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by $0. Remaining arguments initialize the variable *argv*. *csh* scripts should always start with

```
#! /bin/csh –f
```

which causes the kernel to fork off */bin/csh* to process them even if invoked by a Bourne shell user and inhibits processing of the *.cshrc* file to prevent interference by the user's differing aliases.

Signal Handling

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

EXAMPLE

csh

creates a new C shell which will accept shell commands.

FILES

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

LIMITATIONS

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

SEE ALSO

sh(1), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), environ(5).
An Introduction to the C Shell, by William Joy.

ERRORS

It suffices to place the sequence of commands in parenthesis 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.

Control structures 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.

Bourne shell scripts which start with # will be executed by *csh* unless they use the kernel's #! facility, e.g.

```
#! /bin/shFR
```

AUTHOR

William Joy.

ORIGIN

4.3 BSD

NAME

csplit – context split

SYNOPSIS

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

DESCRIPTION

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

00: From the start of *file* up to (but not including) the line referenced by *arg1*.
01: From the line referenced by *arg1* up to the line referenced by *arg2*.
•
•
•
n+1: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a **−** then standard input is used.

The options to *csplit* are:

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

−k *csplit* normally removes created files if an error occurs. If the **−k** option is present, *csplit* leaves previously created files intact.

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

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

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

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

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

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

Enclose all *rexp* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit –f cobol file '/procedure division/' /par5./ /par16./
```

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

```
cat cobol0[0–3] > file
```

Note that this example overwrites the original file.

```
        csplit -k file  100  {99}
```

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

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

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

SEE ALSO

ed(1), sh(1).
regexp(5) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Self-explanatory except for:
    arg − out of range
which means that the given argument did not reference a line between the current position and the end of the file.

NAME

ct – spawn getty to a remote terminal

SYNOPSIS

ct [**−wn**] [**−xn**] [**−h**] [**−v**] [**−sspeed**] telno ...

DESCRIPTION

ct dials the telephone number of a modem that is attached to a terminal, and spawns a *getty* process to that terminal. *telno* is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. (The set of legal characters for *telno* is 0 thru 9, -, =, *, and #. The maximum length *telno* is 31 characters). If more than one telephone number is specified, *ct* will try each in succession until one answers; this is useful for specifying alternate dialing paths.

ct will try each line listed in the file **/usr/lib/uucp/Devices** until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, *ct* will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. *ct* will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the **−w***n* option, where *n* is the maximum number of minutes that *ct* is to wait for a line.

The **−x***n* option is used for debugging; it produces a detailed output of the program execution on stderr. The debugging level, *n*, is a single digit; **−x9** is the most useful value.

Normally, *ct* will hang up the current line, so the line can answer the incoming call. The **−h** option will prevent this action. The **−h** option will also wait for the termination of the specified *ct* process before returning control to the user's terminal. If the **−v** option is used, *ct* will send a running narrative to the standard error output stream.

The data rate may be set with the **−s** option, where *speed* is expressed in baud. The default rate is 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of getty is on the line (*getty* or *uugetty*). For the first case, *ct* prompts, **Reconnect?** If the response begins with the letter **n**, the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed. In the second case, there is already a getty (*uugetty*) on the line, so the **login:** message will appear.

To log out properly, the user must type **control D**.

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

FILES

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

SEE ALSO

cu(1C), login(1), uucp(1C).
getty(1M), uugetty(1M) in the *System Administrator's Reference Manual.*

ERRORS

For a shared port, one used for both dial-in and dial-out, the *uugetty* program running on the line must have the **−r** option specified (see *uugetty*(1M)).

NAME

ctags – create a tags file

SYNOPSIS

ctags [**−BFatuwvx**] [**−f** *tagsfile*] name ...

DESCRIPTION

ctags makes a tags file for *ex*(1) from the specified C, Pascal, Fortran, YACC, lex, and lisp 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 objects 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) (currently not supported) 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
```

Normally *ctags* places the tag descriptions in a file called *tags*; this may be overridden with the **−f** option.

Files whose names end in **.c** or **.h** are assumed to be C source files and are searched for C routine and macro definitions. Files whose names end in **.y** are assumed to be YACC source files. Files whose names end in **.l** are assumed to be either lisp files if their first non-blank character is ';', '(', or '[', or lex files otherwise. Other files 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:

−F	use forward searching patterns (/.../) (default).
−B	use backward searching patterns (?...?).
−a	append to tags file.
−t	create tags for typedefs.
−w	suppressing warning diagnostics.
−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.)

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.

NAME

ctrace – C program debugger

SYNOPSIS

ctrace [*options*] [*file*]

DESCRIPTION

The *ctrace* command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the **−x** option. *ctrace* reads the C program in *file* (or from standard input if you do not specify *file*), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc(1)* command does not allow the use of a pipe. You then compile and execute this file. As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs(1)* or *tail(1)* commands. The options commonly used are:

−f *functions* Trace only these *functions.*

−v *functions* Trace all but these *functions.* You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats, if appropriate, with these options:

−o Octal

−x Hexadecimal

−u Unsigned

−e Floating point These options are used only in special circumstances:

−l *n* Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

−s Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

−t *n* Trace *n* variables per statement instead of the default of 10 (the maximum number is 20, which is silently enforced). The Diagnostics section explains when to use this option.

−P Run the C preprocessor on the input before tracing it. You can also use the **−D**, **−I**, and **−U** *cpp(1)* options. These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX System environment:

−b Use only basic functions in the trace code, that is, those in *ctype(3C)*, *printf(3S)*, and *string(3C)*. These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal(2)* or *setjmp(3C)*.

−p *string'*

Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

−r *f* Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **−p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main()        /* count lines in input */
3 {
4       int c, nl;
5
6       nl = 0;
7       while ((c = getchar()) != EOF)
8               if (c = '\n')
9                       ++nl;
10      printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(cntl-d)
```

the program will be compiled and executed. The output of the program will be the number **2**, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6       nl = 0;
        /* nl == 0 */
7       while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 1 */
7       while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 2 */
7       while ((c = getchar()) != EOF)
```

If you now enter an end of file character (cntl-d) the final output will be:

```
            /* c == -1 */
    10      printf("%d\n", nl);
            /* nl == 2 */2
             return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function. The trace output shows that variable **c** is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **−f** or **−v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program. You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the **CTRACE** preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) (currently not supported) if you compile with the **−g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc(1)*, since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

Ctrace Diagnostics

warning: some variables are not traced in this statement
: Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **−t** option to increase this number.

warning: statement too long to trace
: This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use −P option
: This is usually caused by #ifdef/#endif preprocessor statements in the middle of a C statement, or by a semicolon at the end of a #define preprocessor statement.

'if ... else if' sequence too long
: Split the sequence by removing an **else** from the middle.

possible syntax error, try −P option
: Use the **−P** option to preprocess the *ctrace* input, along with any appropriate **−D**, **−I**, and **−U** preprocessor options. If you still get the error message, check the Warnings section below.

Cc Diagnostics

warning: illegal combination of pointer and integer
warning: statement not reached
warning: sizeof returns 0
: Ignore these messages.

compiler takes size of function
: See the *ctrace* "possible syntax error" message above.

yacc stack overflow
: See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression
: Use the **−t** option to reduce the number of traced variables per statement from the default of 10. Ignore the "ctrace: too many variables to trace" warnings you will now get.

redeclaration of signal
: Either correct this declaration of *signal(2)*, or remove it and #include <signal.h>.

SEE ALSO

signal(2), ctype(3C), fclose(3S), printf(3S), setjmp(3C), string(3C).
bfs(1), tail(1) in the *User's Reference Manual.*

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers. Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name. *ctrace* assumes that BADMAG is a preprocessor macro, and that EOF and NULL are #defined constants. Declaring any of these to be variables, e.g., "int EOF;", will cause a syntax error.

ERRORS

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *ctrace* may choose to print the address of an aggregate or use the wrong format (e.g., 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate. Pointer values are always treated as pointers to character strings. The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called. The user must declare gets( ) as char *gets( ) or include <stdio.h>.

FILES

/usr/lib/ctrace/runtime.c	run-time trace package

NAME

cu – call another UMIPS-V system or UNIX system

SYNOPSIS

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

DESCRIPTION

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

cu accepts the following options and arguments:

−s*speed*	Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the **/usr/lib/uucp/Devices** file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.
−l*line*	Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line having the right speed. When the **−l** option is used without the **−s** option, the speed of a line is taken from the Devices file. When the **−l** and **−s** options are both used together, cu will search the Devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/tty***ab*) in which case a telephone number (*telno*) is not required. The specified device need not be in the **/dev** directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with *systemname* rather than *telno* will not give the desired result (see *systemname* below).
−h	Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.
−t	Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.
−d	Causes diagnostic traces to be printed.
−o	Designates that odd parity is to be generated for data sent to the remote system.
−n	For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.
−e	Designates that even parity is to be generated for data sent to the remote system.
telno	When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.
systemname	A uucp system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems.** Note: the *systemname* option should not be used in conjunction with the **−l** and **−s** options as *cu* will connect to

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

The *transmit* process interprets the following user initiated commands:

.	terminate the conversation.
!	escape to an interactive shell on the local system.
!*cmd*...	run *cmd* on the local system (via **sh −c**).
$*cmd*...	run *cmd* locally and send its output to the remote system.
%cd	change the directory on the local system. Note: **!cd** will cause the command to be run by a sub-shell, probably not what was intended.
%take *from* [*to*]	copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
%put *from* [*to*]	copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. For both **%take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.
line	send the line *line* to the remote system.
%break	transmit a **BREAK** to the remote system (which can also be specified as **%b**).
%debug	toggles the **-d** debugging option on or off (which can also be specified as **%d**).
t	prints the values of the termio structure variables for the user's terminal (useful for debugging).
l	prints the values of the termio structure variables for the remote communication line (useful for debugging).
%nostop	toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one which does not respond properly to the DC3 and DC1 characters.

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

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

The use of **%put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

The use of **%take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, *tabs* mode (See *stty(1))* should be set on the remote system if tabs are to be copied without expansion to spaces.

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

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

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

EXAMPLES

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

```
cu  -s1200  9=12015551212
```

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

To login to a system connected by a direct line:

```
cu  -l  /dev/ttyXX
```

or

```
cu -l ttyXX
```

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

```
cu  -s1200  -l  ttyXX
```

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

```
cu  -l  culXX  9=12015551212
```

To use a system name:

```
cu  systemname
```

FILES

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

SEE ALSO

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1).

DIAGNOSTICS

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

WARNINGS

The *cu* command does not do any integrity checking on data it transfers. Data fields with special *cu* characters may not be transmitted properly. Depending on the interconnection hardware, it may be necessary to use a . to terminate the conversion even if **stty 0** has been used. Non-printing characters are not dependably transmitted using either the **%put** or **%take** commands. *cu* between an IMBR1 and a penril modem will not return a login prompt immediately upon connection. A carriage return will return the prompt.

ERRORS

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

NAME

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

SYNOPSIS

cut −clist [file ...]
cut −flist [**−d** char] [**−s**] [file ...]

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. In addition, a file name of "**−**" explicitly refers to standard input.

The meanings of the options are:

list	A comma-separated list of integer field numbers (in increasing order), with optional **−** to indicate 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.

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

EXAMPLES

cut −d: −f1,5 /etc/passwd	mapping of user IDs to names
name=`who am i | cut −f1 −d" "`	to set **name** to current login name.

DIAGNOSTICS

ERROR: line too long
A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option
Missing **−c** or **−f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields
The *list* is empty.

ERROR: no delimeter

Missing *char* on **−d** option.

ERROR: cannot handle multiple adjacent backspaces
Adjacent backspaces cannot be processed correctly.

WARNING: cannot open <filename>
Either *filename* cannot be read or does not exist. If multiple filenames are present, prcessing continues.

SEE ALSO

grep(1), paste(1).

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref [options] files

DESCRIPTION

The *cxref* command analyzes a collection of C files and attempts to build a cross-reference table. *cxref* uses a special version of *cpp* to include **#define**'d information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the **−c** option, in combination. Each symbol contains an asterisk (∗) before the declaring reference.

In addition to the **−D**, **−I** and **−U** options [which are interpreted just as they are by *cc*(1) and *cpp*(1)], the following *options* are interpreted by *cxref*:

−c	Print a combined cross-reference of all input files.
−w<*num*>	Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.
−o file	Direct output to *file*.
−s	Operate silently; do not print input file names.
−t	Format listing for 80-column width.

FILES

LLIBDIR	usually /usr/lib
LLIBDIR/xcpp	special version of the C preprocessor.

SEE ALSO

cc(1), cpp(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

ERRORS

cxref considers a formal argument in a *#define* macro definition to be a declaration of that symbol. For example, a program that *#includes* **ctype.h**, will contain many declarations of the variable **c**.

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] | **+**format]

DESCRIPTION

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

```
date 10080045
```

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

If the argument begins with **+**, the output of *date* is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by **%** and will be replaced in the output by its corresponding value. A single **%** is encoded by **%%**. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	day of year – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday – Sun to Sat
h	abbreviated month – Jan to Dec
r	time in AM/PM notation

EXAMPLE

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

would have generated as output:

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

DIAGNOSTICS

No permission	if you are not the super-user and you try to change the date;
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

FILES

/dev/kmem

WARNING

Should you need to change the date while the system is running multi-user, use *sysadm*(1) *datetime*.

SEE ALSO

sysadm(1).

NAME

dbx - source-level debugger

SYNOPSIS

dbx [−I *directory*] [−c *file*] [−i] [−r] [-pixie] [*object*] [*core*]

DESCRIPTION

Dbx, a source-level debugger, runs under UMIPS-BSD (4.3 BSD) and UMIPS-V (V.3) versions of the operating system. It can handle UMIPS-V shared libraries. This enhanced version of *dbx* works with *cc(1), f77(1), pc(1), as(1),* and MIPS machine code.

The object file used with the debugger is produced by specifying an appropriate option (usually **−g**) to the compiler. The resulting object file contains symbol table information, including the names of all source files that the compiler translated to create the object file. These source files are accessible from the debugger. If **−g** is not specified, limited debugging is possible.

If a core file exists in the current directory or a coredump file is specified, *dbx* can be used to look at the state of the program when it faulted.

Running dbx

If a *.dbxinit* file resides in the current directory or in the user's home directory, the commands in it are executed when *dbx* is invoked.

When invoked, *dbx* recognizes these command line options:

−I *directory* or **−I***directory*
: Tells *dbx* to look in the specified directory for source files. Multiple directories can be specified by using multiple *−I* options. *Dbx* searches for source files in the current directory and in the object file's directory whether or not *−I* is used.

−c *file* Selects a command file other than *.dbxinit*.

−i Uses interactive mode. This option does not treat #s as comments in a file. It prompts for source even when it reads from a file. With this option, *dbx* also has extra formatting as if for a terminal.

−r Runs the object file immediately.

−pixie Uses pixie output. The executable must be 'executable.pixie', and the non-pixie executable must be in the same directory as the pixie executable.

−prom Permits debugging in the standalone environment when using the MIPS System Programmer's Package. For more information, refer to the *System Programmer's Package Reference* manual.

−sable Permits debugging programs running under the processor simulator when the MIPS System Programmer's Package.

The *dbx* monitor offers powerful command line editing. For a full description of these emacs-style editing features, see *csh(1)*.

Multiple commands can be specified on the same command line by separating them with a semicolon (;). If the user types a string and presses the stop character (usually ^z; see **stty(1)**), *dbx* tries to complete a symbol name from the program that matches the string.

dbx can also run under *emacs* as *inferior,* which means under this mode, *dbx* is controlled by *emacs* and communicates with *emacs.* When in *emacs,* command **M-x dbx** starts dbx and will prompt you for filename to be debugged. In MIPS environment, the following keys are bound to commonly used *dbx* commands: **M-n, M-s, M-i, M-u, M-d, C-c C-f, C-x space** represents for *next, step, stepi, up, down, finish, set breakpoint at current line* respectively. Note that in *emacs*, M-x usually means *esc-x*, *C-x* means *ctl-x*. In *emacs* you can define your own key binding.

The Monitor

These commands control the *dbx* monitor:

![*string*] [*integer*] [*−integer]*
: Specifies a command from the history list.

help Prints a list of *dbx* commands, using the UNIX system **more** command to display the list.

history Prints the items from the history list. The default if 20.

quit[**!**] Exit *dbx* after verification. If **!** is specified, verification isn't required.

Controlling dbx

alias [*name(arg1,...argN)"string"*]
: Lists all existing aliases, or, if an argument is specified, defines a new alias.

unalias *alias command_name*
: Removes the specified alias.

delete *expression1, ...expressionN*

delete *all*
: Deletes the specified item from the status list. The argument *all* deletes all items from the status list.

playback input [*file*]
: Replays commands that were saved with the **record input** command in a text file.

playback output [*file*]
: Replays debugger output that was saved with the **record output** command.

record input [*file*]
: Records all commands typed to *dbx*.

record output [*file*]
: Records all *dbx* output.

sh [*shell command*]
: Calls a shell from *dbx* or executes a shell command.

status Lists currently set **stop, record,** and **trace** commands.

tagvalue (*tagname*)
: Returns the value of *tagname*. If the tags extends to more than one line, or if it contains arguments, an error occurs. **tagvalue** can be used in any expression.

set [*variable = expression*]
: Lists existing debugger variables and their values. This command can also be used to assign a new value to an existing variable or to define a new variable.

unset *variable*
: Removes the setting of a specified debugger variable.

Examining Source

/regular expression
: Searches ahead in the source code for the regular expression.

?regular expression
: Searches back in the source code for the regular expression.

edit [*file*]
  Calls an editor from *dbx*.

file [*file*] Prints the current file name, or, if a file name is specified, this command changes the current file to the specified file.

func [*expression*] [*procedure*]
  Moves to the specified procedure (activation level), or, if an expression or procedure isn't specified, prints the current activation level.

list [*expression:integer*]

list [*expression*]
  Lists the specified lines. The default is 10 lines.

tag *tagname*
  Sets the current file/line to the location specified by *tagname*. Operations are similar to the tag operations in **vi(1)**.

use [*directory1 ... directoryN*]
  Lists source directories, or, if a directory name is specified, this command substitutes the new directories for the previous list.

whatis *variable*
  Prints the type declaration for the specified name.

which *variable*
  Finds the variable name currently being used.

whereis *variable*
  Prints all qualifications (the scopes) of the specified variable name.

Controlling Programs

assign *expression1 = expression2*
  Assigns the specified expression to a specified program variable.

[*n*] **cont** [*signal*]

cont [*signal*] **to** *line*

cont [*signal*] **in** *procedure*
  Continues executing a program after a breakpoint. *n* breakpoints are ignored if *n* is specified before stepping; If specified, *signal* is delivered to the processing being debugged.

goto *line*
  Goes to the specified line in the source.

next [*integer*]
  Steps over the specified number of lines. The default is one. This command does not step into procedures.

rerun [*arg1 ... argN*] [*<file1*] [*>file2*]

rerun [*arg1 ... argN*] [*<file1*] [*>&file2*]
  Reruns the program, using the same arguments that were specified to the **run** command. If new arguments are specified, **rerun** uses those arguments.

run [*arg1 ... argN*] [*<file1*] [*>file2*]

run [*arg1 ... argN*] [*<file1*] [*>&file2*]
  Runs the program with the specified arguments.

return [*procedure*]
  Continues executing until the procedure returns. If a procedure isn't specified, *dbx*

assumes the next procedure.

step [*integer*]
Steps the specified number of lines. This command steps into procedures. The default is one line.

Setting Breakpoints

catch [*signal*]
Lists all signals that *dbx* catches, or, if an argument is specified, adds a new signal to the catch list.

ignore [*signal*]
Lists all signals that *dbx* does not catch. If a signal is specified, this command adds the signal to the ignore list.

stop [*variable*]

stop [*variable*] **at** *line* [**if** *expression*]

stop [*variable] **in** *procedure* [*if expression*]

stop [*variable*] **if** *expression*
Sets a breakpoint at the specified point.

trace *variable* [**at** *line* [*if expression*]

trace *variable* [**in** *procedure [if expression*]
Traces the specified variable.

when [*variable*] [**at** *line*] {*command_list*}

when [*variable*] [**in** *procedure*] {*command_list*}
Executes the specified *dbx* comma separated command list.

Examining Program State

dump [*procedure*] [.]
Prints variable information about the procedure. If a dot (.) is specified, this command prints global variable information on all procedures in the stack and the variables of those procedures.

down [*expression*]
Moves down the specified number of activation levels in the stack. The default is one level.

up [*expression*]
Moves up the specified number of activation levels on the stack. The default is one.

print *expression1,...expressionN*
Prints the value of the specified expression. If *expression* is a dbx keyword, it must be enclosed within parentheses. For example, to print out a variable called 'output' (which is also a variable in the playback and record commands) you must type: print (output)

printf *"string", expression1,...expressionN*
Prints the value of the specified expression, using C language string formatting. As in the **print** command, if *expression* is a dbx keyword, you must enclose it within parentheses.

printregs
Prints all register values.

where Does a stack trace, which shows the current activation levels.

where *n* Prints out only the top *n* levels of the stack.

Debugging at the Machine Level

[*n*] **conti** [*signal*]

conti [*signal*] **to** *address*

conti [*signal*] **in** *procedure*

Continues executing assembly code after a breakpoint. *n* breakpoints are ignored if *n* is specified before stepping; If specified, *signal* is delivered to the processing being debugged.

nexti [*integer*]

Steps over the specified number of machine instructions. The default is one. This command does not step into procedures.

stepi [*integer*]

Steps the specified number of machine instructions. This command steps into procedures. The default is one instruction.

stopi [*variable*] **at** *address* [**at** *address* [**if** *expression*]

stopi [*variable*] **in** *procedure* [**if** *expression*]

stopi [*variable*] **if** *expression*

Sets a breakpoint in the machine code at the specified point.

tracei *variable* **at** *address* [**at** *address* **if** *expression*]

tracei *variable* **in** *procedure* [**at** *address* **if** *expression*]

Traces the specified variable in machine instructions.

wheni [*variable*] [**at** *address*] {*command_list*}

wheni [*variable*] [**in** *procedure*] {*command_list*}

Executes the specified *dbx* comma separated command list.

address[?]/<count><mode>

Searching forward (or backward, if ? is specified,) prints the contents *address* or disassembles the code for the instruction *address*; *count* is the number of items to be printed at the specified address. *mode* is one of the characters in the following table producing the indicated result:

d	Print a short word in decimal
D	Print a long word in decimal
o	Print a short word in octal
O	Print a long word in octal
x	Print a short word in hexadecimal
X	Print a long word in hexadecimal
b	Print a byte in octal
c	Print a byte as a character
s	Print a string of characters that ends in a null
f	Print a single precision real number
g	Print a double precision real number
i	Print machine instructions
n	Prints data in typed format.

address/<*countL*><*value*><*mask*>

Searches for a 32-bit word starting at the specified *address*; *count* specifies the number of word to process in the search; an address is printed when the the word at *address*, after an AND operation with *mask*, is equal to *value*.

Predefined dbx Variables

The debugger has these predefined variables:

$addrfmt
: Specifies the format for addresses. This can be set any specification that a C printf statement can format. The default is zero.

$byteaccess
: Same as $addrfmt.

$casesense
: When set to a nonzero value, specifies that uppercase and lowercase letters be taken into consideration during a search. When set to 0, the case is ignored. The default is 0,

$curevent
: Shows the last even number as seen in the status feature. Set only by dbx.

$curline Specifies the current line. Set only by dbx.

$cursrcline
: Shows the last line listed plus 1. Set only by DBX

$curpc Specifies the current address. Used with the *wi* and *li* aliases.

$datacache
: Caches information from the data space so that *dbx* must access data space only once. To debug the operating system, set this variable to 0; otherwise, set it to a nonzero value. The default is 1.

$debugflag
: For internal use by *dbx*.

$defin For internal use by *dbx*.

$defout For internal use by *dbx*.

$dispix For use when debugging pixie code. When set to 0, machine code is show while debugging. When set to 1, pixie code is shown. The default is 0.

$hexchars
: Output characters are printed in hexadecimal format (set, unset).

$hexin Specifies that input constants are hexadecimal.

$hexints
: When set to a nonzero value, changes the default output constants to hexadecimal. Overrides *$octints*.

$hexstrings
: When set to 1, specifies that all strings are printed in hexadecimal; when set to 0, strings are printed in character format.

$historyevent
: Shows the current history line.

$lines number of lines for history. The default is 20

$listwindow
: Specifies how many lines the *list* command prints.

$main Specifies the name of the procedure that *dbx* will start with. This can be set to any procedure. The default is "main"

$maxstrlen
: Specifies how many characters of a string that *dbx* prints for pointers to strings. The default is 128.

$octin
: When set to non-zero, changes the default input constants to octal. When set, *$hexint* overrides this setting.

$octints
: Output integers are printed octal format (set, unset).

$page
: Specifies whether to page long information. A nonzero value turns on paging; a 0 turns it off. The default is 1.

$pagewindow
: Specifies how many lines print when information runs longer than one screen. This can be changed to match the number of lines on any terminal. If set to 0, this variable assumes one line. The default is 22, leaving space for continuation query).

$pdbxport
: port name from /etc/remote[.pdbx] used to connect to target machine for pdbx

$printwhilestep
: For use with the **step**[*n*] and **stepi**[*n*] instructions. A non-zero integer specifies that all *n* lines and/or instructions should be printed out. A zero specifies that only the last line and/or instruction should be printed out. The default is zero.

$pimode
: Prints input when used with the *playback input* command. The default is 0.

$printdata
: When set to a nonzero value, the contents of registers used are printed next to each instruction displayed. The default is 0.

$printwide
: When se to a nonzero value, the contents of variables are printed in a horizontal format. The default is 0.

$prompt
: Sets the prompt for *dbx*.

$readtextfile
: When set to 1, *dbx* tries to read instructions from the object file rather than the process. *dbx* executes faster when debugging remotely using the System Programmer's Package. This variable should always be set to 0 when the process being debugged copies in code during the debugging process. The default is 1.

$regstyle
: A zero value causes registers to be printed out in their normal *r* format (r0,r1,...r31). A nonzero value causes the registers to be printed out in a special format (zero, at, v0, v1,...) commonly used in debugging programs written in assembly language. The default is 0.

$repeatmode
: When set to a nonzero value, after pressing the RETURN key (for an empty line), the last command is repeated. The default is 1.

$rimode
: When set to a nonzero value, input will is recorded while recording output . The default is 0.

$sigtramp
: Tells *dbx* the name of the code called by the system to invoke user signal handlers. This variable is set to sigvec for UMIPS-BSD and to sigtramp for UMIPS-V

$tagfile	Contains a filename, indicating the file in which the tag command and the tabvalue macro are to search for tags.

Predefined dbx Aliases

The debugger has these predefined aliases:

?	Prints a list of all *dbx* commands.
a	Assigns a value to a program variable.
b	Sets a breakpoint at a specified line.
bp	Stops in a specified procedure.
c	Continues program execution after a breakpoint.
d	Deletes the specified item from the status list.
e	Looks at the specified file.
f	Moves to the specified activation level on the stack.
g	Goes to the specified line and begins executing the program there.
h	Lists all items currently on the history list.
j	Shows what items are on the status list.
l	Lists the next 10 lines of source code.
li	Lists the next 10 machine instructions.
n or S	Step over the specified number of lines without stepping into procedure calls.
ni or Si	Step over the specified number of assembly code instructions without stepping into procedure calls.
p	Prints the value of the specified expression or variable.
pd	Prints the value of the specified expression or variable in decimal.
pi	Replays **dbx** commands that were saved with the **record input** command.
po	Prints the value of the specified expression or variable in octal.
pr	Prints values for all registers. **px** Prints the value for the specified variable or expression in hexadecimal.
q	Ends the debugging session.
r	Runs the program again with the same arguments that were specified with the **run** command.
ri	Records in a file every command typed.
ro	Records all debugger output in the specified file.
s	Steps the next number of specified lines.
si	Steps the next number of specified lines of assembly code instructions.
t	Does a stack trace.
u	Lists the previous 10 lines.
w	Lists the 5 lines preceding and following the current line.
W	Lists the 10 lines preceding and following the current line.
wi	Lists the 5 machine instructions preceding and following the machine instruction.

SEE ALSO

MIPS Languages Programmer Guide.

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. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

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

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

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

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

d The top value on the stack is duplicated.

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

P Interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

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

Q Exits the program. The top value on the stack is popped and the string execution level is popped by that value.

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

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

[...] Puts the bracketed ASCII string onto the top of the stack.

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

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

! Interprets the rest of the line as a UNIX system command.

c All values on the stack are popped.

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

I Pushes the input base on the top of the stack.

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

O Pushes the output base on the top of the stack.

k The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.

z The stack level is pushed onto the stack.

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

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

; : are used by *bc*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

bc(1).

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

delta – make a delta (change) to an SCCS file

SYNOPSIS

delta [**−r**SID] [**−s**] [**−n**] [**−g**list] [**−m**[mrlist]] [**−y**[comment]] [**−p**] files

DESCRIPTION

delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

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

delta may issue prompts on the standard output depending upon certain keyletters specified and flags [see *admin*(1)] that may be present in the SCCS file (see **−m** and **−y** keyletters below).

Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *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* 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 the **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

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*(4)] and will cause an error.

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

If the standard input (**−**) is specified on the *delta* command line, the **−m** (if necessary) and **−y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

SEE ALSO

admin(1), cdc(1), get(1), prs(1), rmdel(1), sccsfile(4).
bdiff(1), help(1) in the *User's Reference Manual*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**−m**x] [**−w**] [files]

DESCRIPTION

deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **−m** option may be followed by an **m**, **s**, or **l**. The **−mm** option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The **−ml** option forces the **−mm** option and also causes deletion of lists associated with the **mm** macros.

The **−ms** option causes deletion of the **ms** macro commands.

If the **−w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (**&**), and apostrophes (**'**); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.
The **−ml** option does not handle nested lists correctly.

NAME

diff – differential file comparator

SYNOPSIS

diff [**−efbh**] file1 file2

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is **−**, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

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

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

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

The **−b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **−e** option produces a script of *a, c,* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **−f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **−e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

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

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **−h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **−e** and **−f** are unavailable with **−h**.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

ERRORS

Editing scripts produced under the **−e** or **−f** option are naive about creating lines consisting of a single period (**.**).

WARNINGS

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

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**−ex3**] file1 file2 file3

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

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

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

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

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

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

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

FILES

/tmp/d3∗
/usr/lib/diff3prog

SEE ALSO

diff(1).

ERRORS

Text lines that consist of a single **.** will defeat **−e**.
Files longer than 64K bytes will not work.

NAME

dircmp – directory comparison

SYNOPSIS

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

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

−d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

−s Suppress messages about identical files.

−w*n* Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

dis – disassemble an object file

SYNOPSIS

dis [-h] [-s] [-p procedure] [*file* ...]

DESCRIPTION

Dis disassembles object files into machine instructions. Please note that assember code and machine code can differ on this machine. For a full description of the machine language, see the *R2000 Processor User's Guide*. A *file* can be an object or an archive.

The **−h,** flag causes the general register names to be printed, rather than the software register names. The **−p** flag disassembles only the specified procedure from the object file. The **−S** causes source lisitings to be listed. Otherwise, only instructions will listed.

BUGS

Disassembling an archive is not currently operational.

NAME

domainname − set or display name of current domain system

SYNOPSIS

domainname [*nameofdomain*]

DESCRIPTION

Without an argument, *domainname* displays the name of the current domain. Only the super-user can set the domainname by giving an argument; this is usually done in the startup script */etc/init.d/nfs*.

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\0*n*	where *n* is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.

echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

CAVEATS

When representing an 8-bit character by using the escape convention **\0***n,* the *n* must **always** be preceded by the digit zero (0).

For example, typing: **echo** **'WARNING:\07'** will print the phrase **WARNING:** and sound the "bell" on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the "\" that precedes the "07".

For the octal equivalents of each character, see ascii(5), in the *Programmer's Reference Manual.*

NAME

ed, red – text editor

SYNOPSIS

ed [**−s**] [**−p** string] [**−x**] [file]

red [**−s**] [**−p** string] [**−x**] [file]

DESCRIPTION

ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited.

−s Suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !*shell command*. Also, see the **WARNING** section at the end of this manual page.

−p Allows the user to specify a prompt string.

−x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt*(1)). Also, see the **WARNING** section at the end of this manual page.

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

red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !*shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(4) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in **stty −tabs** or **stty tab3** mode (see *stty*(1)), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
        <:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputing text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line, followed immediately by a carriage return.

ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character RE*s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4 below).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. **$** (dollar sign), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (**/**) is used in the *g* command, below.)

1.3 A period (**.**) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (**−**) may be used to indicate a range of consecutive ASCII characters; for example, **[0−9]** is equivalent to **[0123456789]**. The **−** loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., []**a−f**] matches either a right square bracket (]) or one of the 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 *RE*s from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by **\{***m***\}**, **\{***m***,\}**, or **\{***m***,***n***\}** is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; **\{***m***\}** matches *exactly m* occurrences; **\{***m***,\}** matches *at least m* occurrences; **\{***m***,***n***\}** matches *any number* of occurrences *between m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences **\(** and **\)** is a RE that matches whatever the unadorned RE matches.

2.6 The expression *****n* matches the same string of characters as was matched by an expression enclosed between **\(** and **\)** *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of **\(** counting from the left. For example, the expression **^\(.*\)\1$** matches a line consisting of two repeated appearances of the same string.

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

3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2　A dollar sign (**$**) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction ^*entire RE* **$** constrains the entire RE to match the entire line.

The null RE (e.g., **//**) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

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

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the mark name character *x*, which must be a lowercase letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (**/**) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (**?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (**+**) or a minus sign (**−**) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with **+** or **−**, the addition or subtraction is taken with respect to the current line; e.g, **−5** is understood to mean **.−5**.
9. If an address ends with **+** or **−**, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address **−** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to **−**.) Moreover, trailing **+** and **−** characters have a cumulative effect, so **−−** refers to the current line less 2.
10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; **.** is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c
<text>
.

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; **.** is left at the last line input, or, if there were none, at the first line that was not deleted.

(. , .)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; **.** is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *E*dit command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

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

(1 , $)g/*RE***/***command list*

In the *g*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with **.** initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The **.** terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *ERRORS* and the last paragraph before *FILES* below.

(1 , $)G/*RE*/
In the interactive *G*lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *h*elp command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *H*elp command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>
.

The *i*nsert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(. , .+1)j
The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*
The mar*k* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x* then addresses this line; . is unchanged.

(. , .)l
The *l*ist command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab, backspace*) are represented by visually mnemonic over-strikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(. , .)m*a*
The *m*ove command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(. , .)n
The *n*umber command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(. , .)p
The *p*rint command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see *DIAGNOSTICS*, below.

Q

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

($)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; **.** is set to the last line read in. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(. , .)s/*RE***/***replacement***/** or
(. , .)s/*RE***/***replacement***/g** or
(. , .)s/*RE***/***replacement***/n** n = 1-512

The *s*ubstitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator **g** appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of **/** to delimit the RE and the *replacement*; **.** is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by ****. As a more general feature, the characters *****n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \(and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \(starting from the left. When the character **%** is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The **%** loses its special meaning when it is in a replacement string of more than one character or is preceded by a ****.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by ****. Such substitution cannot be done as part of a *g* or *v* command list.

(. , .)t*a*

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

u

The *u*ndo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1 , $)v/*RE***/***command list*

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

(1 , $)V/*RE***/**

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

(1 , $)w *file*

The *w*rite command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *umask*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); **.** is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

An encryption key is requested from the standard input. Subsequent *e*, *r*, and *w* commands will use this key to encrypt or decrypt the text (see *crypt*(1)). An explicitly empty key turns off encryption. Also, see the **−x** option of *ed*.

($)=

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

!*shell command*

The remainder of the line after the **!** is sent to the UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

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

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

When reading a file, *ed* discards ASCII NUL characters. Files (e.g., **a.out**) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If a file is not terminated by a new-line character, **ed** adds one and outputs a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (e.g., **/**) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/usr/tmp default directory for temporary work file.

$TMPDIR
: if this environmental variable is not null, its value is used in place of /usr/tmp as the directory name for the temporary work file.

ed.hup work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.

?*file* for an inaccessible file.
(use the *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The **−s** command-line option inhibits this feature.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1).
fspec(4), regexp(5) in the *Programmer's Reference Manual.*

ERRORS

A *!* command cannot be subject to a *g* or a *v* command.

The *!* command and the **!** escape from the *e*, *r*, and *w* commands cannot be used if the editor is invoked from a restricted shell (see *sh*(1)).

The sequence **\n** in a RE does not match a new-line character.

Characters are masked to 7 bits on input.

If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

WARNINGS

The **−x** option is provided with the Security Administration Utilities, which is available only in the United States.

The **−** option, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the **−** option to use the **−s** option, instead.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**−r**] [**−x**] *name* ...

DESCRIPTION

edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor.

−r Recover file after an editor or system crash.

−x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt*(1)). Also, see the **WARNING** section at the end of this manual page.

The following brief introduction should help you get started with *edit.* If you are using a CRT terminal you may want to learn about the display editor *vi.*

To edit the contents of an existing file you begin with the command "edit name" to the shell. *edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

edit prompts for commands with the character ':', which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit's* buffer (its name for the copy of the file you are editing). Most commands to *edit* use its "current line" if you do not tell them which line to use. Thus if you say **print** (which can be abbreviated **p**) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you **delete** (**d**) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you **delete** this last line, then the new last line becomes the current one. In general, after a **delete,** the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the **append** (**a**) command can be used. After you give this command (typing a carriage return after the word **append**) *edit* will read lines from your terminal until you give a line consisting of just a "**.**", placing these lines after the current line. The last line you type then becomes the current line. The command **insert** (**i**) is like **append** but places the lines you give before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you give the command "1" then *edit* will type this first line. If you then give the command **delete** *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the **substitute** (**s**) command. You say "s/*old* /*new*/" where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command **file** (**f**) will tell you how many lines there are in the buffer you are editing and will say "[Modified]" if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a **write** (**w**) command. You can then leave the editor by issuing a **quit** (**q**) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to **write** the file back. If you try to **quit** from *edit* after modifying the buffer without writing it out, you will be warned that there has been "No **write** since last change" and *edit* will await another command. If you wish not to **write** the buffer out then you can issue another **quit** command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (**c**) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (**u**) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an *undo* and looking to see what happened. If you decide that the change is ok, then you can *undo* again to get it back. Note that commands such as *write* and *quit* cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit ^D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "''". The **z** command can also be given other following characters "z–" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form **/text/** to search forward for *text* or **?text?** to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form **/^text/** which searches for *text* at the beginning of a line. Similarly **/text$/** searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "**.**"; this is most useful in a range of lines as in ".,$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "$–5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put a" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (**e**) command after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ed(1), ex(1), vi(1).

WARNING

The **−x** option is provided with the Security Administration Utilities, swhich is available only in the United States.

NAME

egrep – search a file for a pattern using full regular expressions

SYNOPSIS

egrep [options] full regular expression [file ...]

DESCRIPTION

egrep (*expression grep*) searches files for a pattern of characters and prints all lines that contain that pattern. *egrep* uses full regular expressions (expressions that have string values that use the full set of alphanumeric and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts full regular expressions as in *ed*(1), except for \(and \), with the addition of:

1. A full regular expression followed by **+** that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters $, *, [, ^, |, (,), and \ in *full regular expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes '...'.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

−b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

−c Print only a count of the lines that contain the pattern.

−i Ignore upper/lower case distinction during comparisons.

−l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

−n Precede each line by its line number in the file (first line is 1).

−v Print all lines except those that contain the pattern.

−e *expression*
Search for the given expression. Useful if the expression begins with a −.

−f *file*
Take the list of *full regular expressions* from *file*.

SEE ALSO

ed(1), fgrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

ERRORS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h.**

NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [**−c**] [**−r**[reason]] printers

DESCRIPTION

enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

−c	Cancel any requests that are currently printing on any of the designated printers.
−r[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **−r** option. If the **−r** option is not present or the **−r** option is given without a reason, then a default reason will be used. *reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).

NAME

env – set environment for command execution

SYNOPSIS

env [**–**] [name=value] ... [command args]

DESCRIPTION

env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name***=***value* are merged into the inherited environment before the command is executed. The **–** flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1).
exec(2), profile(4), environ(5) in the *Programmer's Reference Manual*.

NAME

ex – text editor

SYNOPSIS

ex [**–**] [**–v**] [**–t** *tag*] [**–r**] [**–R**] [**–x**] [**+***command*] *name* ...

DESCRIPTION

ex is the root of a family of editors: *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 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*.

For ed Users

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base (see *Terminal Information Utilities Guide*) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1).

ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join** (**j**) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

–	Suppress all interactive-user feedback. This is useful in processing editor scripts.
–v	Invokes *vi*
–t *tagfR*	Edit the file containing the *tag* and position the editor at its definition.
–r *file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
–R	*Readonly* mode set, prevents accidentally overwriting the file.
–x	Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt*(1)). Also, see the **WARNING** section at the end of this manual page.
+*command*	Begin editing by executing the specified editor search or positioning *command.*

The *name* argument indicates files to be edited.

ex States

Command	Normal and initial state. Input prompted for by **:**. Your kill character cancels partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert is normally terminated by a line having only **.** on it, or abnormally with an interrupt.
Visual	Entered by **vi**, terminates with **Q** or ^\\.

ex command names and abbreviations

abbrev	**ab**	next	**n**	unabbrev	**una**
append	**a**	number	**nu**	undo	**u**
args	**ar**			unmap	**unm**
change	**c**	preserve	**pre**	version	**ve**
copy	**co**	print	**p**	visual	**vi**
delete	**d**	put	**pu**	write	**w**
edit	**e**	quit	**q**	xit	**x**
file	**f**	read	**re**	yank	**ya**
global	**g**	recover	**rec**	window	**z**
insert	**i**	rewind	**rew**	escape	**!**
join	**j**	set	**se**	lshift	**<**
list	**l**	shell	**sh**	print next	CR
map		source	**so**	resubst	**&**
mark	**ma**	stop	**st**	rshift	**>**
move	**m**	substitute	**s**	scroll	**^D**

ex Command Addresses

n	line *n*	*/pat*	next with *pat*

.	current	?*pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
−	previous	´*x*	marked with *x*
+*n*	*n* forward	´´	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment var.
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option
set no*x*	disable option
set *x*=*val*	give value *val*
set	show changed options
set all	show all options
set *x*?	show value of option *x*

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
list		print ^I for tab, $ at end
magic		. [* special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^**D**
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in *vi*
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
$	end of line
.	any character
\<	beginning of word
\>	end of word
[*str*]	any char in *str*
[↑*str*]	... not in *str*
[*x*−*y*]	... between *x* and *y*
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command

/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1).
curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual.*
The *Terminal Information Utilities Guide.*

WARNING

The **−x** option is provided with the Security Administration Utilities, which is available only in the United States.

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line '**−**' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

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

expr **\|** *expr*
: returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr **\&** *expr*
: returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { **=**, **\>**, **\>=**, **\<**, **\<=**, **!=** } *expr*
: returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { **+**, **–** } *expr*
: addition or subtraction of integer-valued arguments.

expr { **\***, **/**, **%** } *expr*
: multiplication, division, or remainder of the integer-valued arguments.

expr **:** *expr*
: The matching operator **:** compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of *ed*(1), except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the **\(...\)** pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1.
```
a=`expr $a + 1`
```
adds 1 to the shell variable **a**.

2.
```
# ´For $a equal to either "/usr/abc/file" or just "file"´
expr $a : ´.*/\(.*\)´ \| $a
```
returns the last segment of a path name (i.e., file). Watch out for **/** alone as an argument: *expr* will take it as the division operator (see BUGS below).

3.
```
# A better representation of example 2.
expr //$a : ´.*/\(.*\)´
```
The addition of the **//** characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4.
```
expr $VAR : ´.*´
```

returns the number of characters in **$VAR**.

SEE ALSO

ed(1), sh(1).

DIAGNOSTICS

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither null nor **0**
1	if the expression *is* null or **0**
2	for invalid expressions.

syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If **$a** is an **=**, the command:

```
expr $a = ´=´
```

looks like:

```
expr = = =
```

as the arguments are passed to *expr* (and they will all be taken as the **=** operator). The following works:

```
expr X$a = X=
```

NAME

factor – obtain the prime factors of a number

SYNOPSIS

factor [integer]

DESCRIPTION

When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. *factor* exits if it encounters a zero or any non-numeric character.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. *factor* will take this time when n is prime or the square of a prime.

ERRORS

No check is made that the argument given is a valid integer. Invalid arguments are interpreted as 0.

NAME

f77 – MIPS Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77, the MIPS *ucode* Fortran 77 compiler, produces files in the following formats: MIPS object code in MIPS extended *coff* format (the normal result), binary or symbolic *ucode*, *ucode* object files and binary or symbolic assembly language. *F77* accepts several types of arguments:

Arguments whose names end with '.f' are assumed to be Fortran 77 source programs. They are compiled, and each object program is left in the file whose name consists of the last component of the source with '.o' substituted for '.f'. The '.o' file is only deleted when a single source program is compiled and loaded all at once. Files ending in '.F' are assumed to contain Fortran code which is to be run through the C preprocessor first.

Arguments whose names end with '.r' or '.e' are assumed to be RATFOR or EFL source programs, respectively. These programs are first transformed by the appropriate preprocessor and then compiled by *f77,* producing '.o' files.

Arguments whose names end with '.s' are assumed to be symbolic assembly language source programs. They are assembled, producing a '.o' file. Arguments whose names end with '.i' are assumed to be Fortran 77 source after being processed by the C preprocessor. They are compiled without being processed by the C preprocessor.

If the highest level of optimization is specified (with the **−O3** flag) or only ucode object files are to be produced (with the **−j** flag) each Fortran 77, RATFOR or EFL source file is compiled into a *ucode* object file. The *ucode* object file is left in a file whose name consists of the last component of the source with '.u' substituted for '.f', '.r', or '.e'.

The suffixes described below primarily aid compiler development and are not generally used. Arguments whose names end with '.B', '.O', '.S', and '.M' are assumed to be binary *ucode*, produced by the front end, optimizer, ucode object file splitter and ucode merger respectively. Arguments whose names end with '.U' are assumed to be symbolic *ucode*. Arguments whose names end with '.G' are assumed to be binary assembly language, which is produced by the code generator and the symbolic to binary assembler.

Files that are assumed to be binary *ucode*, symbolic *ucode*, or binary assembly language by the suffix conventions are also assumed to have their corresponding symbol table in a file with a '.T' suffix.

F77 always defines the C preprocessor macros **mips, host_mips** and **unix** to the C macro preprocessor. If the **−cpp** option is present *f77* defines the C preprocessor macro **LANGUAGE_FORTRAN** when a '.f', '.r', or '.e' file is being compiled. *F77* will define the C preprocessor macro **LANGUAGE_ASSEMBLY** when a '.s' file is being compiled. It also defines **SYSTYPE_SYSV** by default but this changes if the **−systype name** option is specified (see the description below).

The following options are interpreted by *f77* and have the same meaning in *cc*(1). 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.

−g0 Have the compiler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages.

−p0 Do not permit any profiling. This is the default. If loading happens, the standard runtime startup routine (**crt1.o**) is used, no profiling library is searched.

−p1 or **−p**
Set up for profiling by periodically sampling the value of the program counter. This option only effects the loading. When loading happens, this option replaces the standard runtime startup routine with the profiling runtime startup routine (**mcrt1.o**) and searches the level 1 profiling library (**libprof1.a**). When profiling happens, the startup routine calls *monstartup*(3) and produces a file *mon.out* that contains execution-profiling data for use with the postprocessor *prof*(1).

−O0 Turn off all optimizations.

−O1 Turn on all optimizations that can be done quickly. This is the default.

−O or **−O2**
Invoke the global *ucode* optimizer. **−O3** Do all optimizations, including global register allocation. This option must precede all source file arguments. With this option, a *ucode* object file is created for each Fortran 77, RATFOR, or EFL source file and left in a '.u' file. The newly created ucode object files, the ucode object files specified on the command line and the runtime startup routine and all the runtime libraries are ucode linked. Optimization is done on the resulting ucode linked file and then it is linked as normal producing an "a.out" file. No resulting '.o' file is left from the ucode linked result as in previous releases. In fact **−c** can no longer be specified with **−O3**.

−feedback *file*
Used with the **−cord** option to specify *file* to be used as a feedback file. This *file* is produced by *prof*(1) with its **−feedback** option from an execution of the program produced by *pixie*(1).

−cord Run the procedure-rearranger, *cord*(1), on the resulting file after linking. The rearrangement is done to reduce the cache conflicts of the program's text. The output of *cord*(1) is left in the file specified by the **−o** *output* option or 'a.out' by default. At least one **−feedback** *file* must be specified.

−j Compile the specified source programs, and leave the *ucode* object file output in corresponding files suffixed with '.u'.

−ko *output*
Name the output file created by the ucode loader as *output*. This file is not removed. If this file is compiled, the object file is left in a file whose name consists of *output* with the suffix changed to a '.o'. If *output* has no suffix, a '.o' suffix is appended to *output*.

−k Pass options that start with a **−k** to the ucode loader. This option is used to specify ucode libraries (with **−kl***x*) and other ucode loader options.

−S Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with '.s'.

−P Run only the C macro preprocessor and put the result for each source file (by suffix convention, i.e. '.f', '.r', '.e' and '.s') in a corresponding '.i' file after being processed by appropriate preprocessors. The '.i' file has no '#' lines in it. This sets the **−cpp** option.

−E Run only the C macro preprocessor on the files (regardless of any suffix or not), and send the result to the standard output. This sets the **−cpp** option.

−o *output*
Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−D*name=def*
−D*name*
Define the *name* to the C macro 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 specified in **−I** options, and finally in the standard directory (**/usr/include**).

−I This option will cause '#include' files never to be searched for in the standard directory (**/usr/include**).

−G *num*
Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 8 bytes.

−v Print the passes as they execute with their arguments and their input and output files.

−V Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−std Have the compiler produce warnings for things that are not standard in the language.

−cpp Run the C macro preprocessor on all Fortran source files before compiling. This includes Fortran sources created by RATFOR or EFL .

−nocpp
Do not run the C macro preprocessor on any Fortran source files before compiling. This is the default for *mf77*(1). This includes Fortran sources created by RATFOR or EFL .

−Olimit *num*
Specify the maximum size, in basic blocks, of a routine that will be optimized by the global optimizer. If a routine has more than this number of basic blocks it will not be optimized and a message will be printed. An option specifying that the global optimizer is to be run (**−O**, **−O2**, or **−O3**) must also be specified. *Num* is assumed to be a decimal number. The default value for *num* is 500 basic blocks.

Either object file target byte ordering can be produced by *f77*. The default target byte ordering matches the machine where the compiler is running. The options **−EB** and **−EL** specify the target byte ordering (big-endian and little-endian, respectively). The compiler also defines a C preprocessor macro for the target byte ordering. These C preprocessor macros are **MIPSEB** and **MIPSEL** for big-endian and little-endian byte ordering respectively.

If the specified target byte ordering does not match the machine where the compiler is running, then the runtime startups and libraries come from **/usr/libeb** for big-endian runtimes on a little-endian machine and from **/usr/libel** for little-endian runtimes on a big-endian machine.

−EB Produce object files targeted for big-endian byte ordering. The C preprocessor macro **MIPSEB** is defined by the compiler.

−EL Produce object files targeted for little-endian byte ordering. The C preprocessor macro **MIPSEL** is defined by the compiler.

The following options are specific for *f77*:

−i2 Make the default integer constants and variables short. All logical quantities will be short. **−i4** is the default.

−onetrip or **−1**
Compile DO loops that execute at least once if reached. (Fortran 77 DO loops are not executed if the upper limit is smaller than the lower limit.)

−66 Suppress extensions that enhance Fortran 66 compatibility.

−C Generate code for runtime subscript range checking. The default suppresses range checking.

−U Do not "fold" cases. *F77* is normally a no-case language (for example **a** equals **A**). The **−U** option causes *f77* to treat uppercase and lowercase separately.

−u Make the default type of a variable *undefined,* rather than using the default Fortran rules.

−w Suppress all warning messages. If the option is **−w66**, only Fortran 66 compatibility warnings are suppressed.

−w1 Suppress warnings about unused variables (but permit other warnings unless **−w** is also specified).

−F Apply the EFL and RATFOR preprocessors to relevant files and put the result in files whose names have their suffix changed to '.f'. (No '.o' files are created.)

−m Apply the M4 preprocessor to each EFL or RATFOR source file before transforming it with the *ratfor*(1) or *efl*(1) preprocessors. The temporary file used as the output of the *m4*(1) preprocessor is that of the last component of the source file with a '.p' substituted for the '.e' or '.r'. This temporary file is removed unless if the **−K** option is specified.

−E Use any remaining characters in the argument as EFL options whenever processing a '.e' file. The temporary file used as the output of the EFL preprocessor has the last component of the source file with a '.f' substituted for the '.e'. This temporary file is removed unless the **−K** option is specified.

−R Use any remaining characters in the argument as RATFOR options whenever processing a '.r' file. The temporary file used as the output of the RATFOR preprocessor is that of the last component of the source file with a '.f' substituted for the '.r'. This temporary file is removed unless the **−K** option is specified.

−automatic
Place local variables on the runtime stack. The same restrictions apply for this option as they do for the automatic keyword. This is the default.

−static
Cause all local variables to be staticly allocated.

−noextend_source
Pad each source line with blanks or truncate it as need be to make it 72 bytes long.

−extend_source
Pad each source line with blanks if need be to make it 132 bytes long, but do not truncate it if it exceeds 132 bytes.

−d_lines
The d_lines option specifies that lines with a D in column 1 are to be compiled and not to be treated as comment lines. The default is to treat lines with a D in column 1 as comment lines.

−col72 This option sets the SVS Fortran 72 column option mode for source statements.

−col120
This option sets the SVS Fortran default mode for source statements.

−vms
Cause the runtime system to behave like VMS Fortran with regard to interpreting carriage control on unit 6.

−N[qxscnl]*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 401.
c	Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.
n	Maximum number of identifiers. Default is 1009.
l	Maximum number of labels. Default is 125.

The option described below is primarily used to provide UNIX compilation environments other than the native compilation environment.

−systype *name*
Use the named compilation environment *name*. See *compilation*(7) for the compilation environments that are supported and their *name*s. This has the effect of changing the standard directory for '#include' files, the runtime libraries and where runtime libraries are searched for. The new items are located in their usual paths but with */name* prepended to their paths. Also a preprocessor macro of the form **SYSTYPE_***NAME* (with *name* capitalized) is defined in place of the default **SYSTYPE_SYSV.**

The options described below primarily aid compiler development and are not generally used:

−H*c* Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be [**fjusmoca**]. It selects the compiler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T' and is not removed.

−K Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.B' file for binary *ucode,* produced by the front end). These intermediate files are never removed even when a pass encounters a fatal error. When ucode linking is performed and the **−K** option is specified the base name of the files created after the ucode link is 'u.out' by default. If **−ko** *output* is specified, the base name of the object file is *output* without the suffix if it exists or suffixes are appended to *output* if it has no suffix.

−# Converts binary *ucode* files ('.B') or optimized binary ucode files ('.O') to symbolic *ucode* (a '.U' file) using *btou*(1). If a symbolic ucode file is to be produced by converting the binary *ucode* from the Fortran 77 compiler front end then the front end option **−Xu** is used instead of *btou*(1).

−W*c[c...],arg1[,arg2...]*
Pass the argument[s] *argi* to the compiler pass[es] *c[c..]*. The *c's* are one of [**pfjusmocablyz**]. The c's selects the compiler pass in the same way as the **−t** option.

The options **−t[hpfjusmocablyzrFIUSMnt]**, **−h***path,* and **−B***string* select a name to use for a particular pass, startup routine, or standard library. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

The **−EB** or **−EL** options, the **−p[01]** options and the **−systype** option must precede all **−B** options because they can affect the location of runtimes and what runtimes are used.

−t[hpfjusmocablyzrFIUSMnt]
Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
include	h (see note below)
cpp	p
fcom	f
ujoin	j
uld	u
usplit	s
umerge	m
uopt	o
ugen	c
as0	a
as1	b
ld	l
ftoc	y
cord	z
[m]crt[1n].o	r
libF77.a	F
libI77.a	I
libU77.a	U
libisam.a	S
libm.a	M
libprof1.a	n
btou, utob	t

If the character 'h' is in the **−t** argument then a directory is added to the list of directories to be used in searching for '#include' files. This directory name has the form COMP_TARGET_ROOT/usr/include*string* . This directory is to contain the include files for the *string* release of the compiler. The standard directory is still searched.

−h*path*
Use *path* rather than the directory where the name is normally found.

−B*string*
Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B**, the **−t** option is assumed to be "hpfjusmocablyzrFIUSMnt". This list designates all names. If no **−t** argument has been processed before the **−B** then a **−B***string* is passed to the loader to use with its **−l***x* arguments.

Invoking the compiler with a name of the form **f77***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/**. If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for all include and library names rather than the default **/**. This affects the standard directory for '#include' files, /usr/include, and the standard library, /usr/lib/libc.a. If this is set, the first directory that is searched for libraries, using the **−l***x* option, is COMP_TARGET_ROOT/usr/lib/cmplrs/cc. The standard directories for libraries are then searched, see *ld*(1).

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/** .

If the environment variable RLS_ID_OBJECT is set, the value is used as the name of an object to link in if a link takes place. This is used to add release identification information to objects. It is always the last object specified to the loader. See *rls_id*(1) for the tools to create this information.

Other arguments are assumed to be either loader options or *Fortran 77*-compatible object files, typically produced by an earlier *f77* run, or perhaps libraries of *Fortran 77*-compatible routines. These files, together with the results of any compilations specified, are loaded in the order given, producing an executable program with the default name **a.out.**

FILES

file.f	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/usr/lib/cpp	C macro preprocessor
/usr/lib/fcom	Fortran 77 front end
/usr/lib/ujoin	binary ucode and symbol table joiner
/usr/bin/uld	ucode loader
/usr/lib/usplit	binary ucode and symbol table splitter
/usr/lib/umerge	procedure intergrator
/usr/lib/uopt	optional global ucode optimizer
/usr/lib/ugen	code generator
/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/lib/crt1.o	runtime startup
/usr/lib/crtn.o	runtime startup
/usr/lib/mcrt1.o	startup for profiling
/usr/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libprof1.a	level 1 profiling library
/usr/lib/libF77.a	Fortran intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/usr/lib/libU77.a	Fortran UNIX interface library
/usr/lib/libisam.a	Indexed sequential access method library
/usr/lib/libm.a	Math library
/usr/include	standard directory for '#include' files

/usr/bin/ld	MIPS loader
/usr/lib/ftoc	interface between *prof*(1) and *cord*(1)
/usr/lib/cord	procedure-rearranger
/usr/bin/btou	binary to symbolic ucode translator
/usr/bin/utob	symbolic to binary ucode translator
/usr/bin/efl	extended Fortran language preprocessor
/usr/bin/ratfor	rational Fortran dialect preprocessor
mon.out	file produced for analysis by *prof*(1)

Runtime startups and libraries for the opposite byte sex of machine the compiler is running on have the same names but are located in different directories. For big-endian runtimes on a little-endian machine the directory is /usr/libeb and for little-endian runtimes on a big-endian machine the directory is /usr/libel.

SEE ALSO

Languages Programmer's Guide
cc(1), as(1), efl(1), ratfor(1), m4(1), monstartup(3), prof(1), ld(1), dbx(1), what(1), cord(1), pixie(1), ftoc(1)

DIAGNOSTICS

The diagnostics produced by *f77* are intended to be self-explanatory. Occasional messages can be produced by the assembler or loader.

NOTES

The standard library, /usr/lib/libc.a, is loaded by using the -lc loader option and not a full path name. The wrong one could be loaded if there are files with the name libc.a*string* in the directories specified with the **−L** loader option or in the default directories searched by the loader.

The handling of include directories and libc.a is confusing.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

fgrep (fast *grep*) seaches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, *, [, ^, |, (,), and \ are interpreted literally by *fgrep,* that is, *fgrep* does not recognize full regular expressions as does *egrep*. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, *fgrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

−b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

−c Print only a count of the lines that contain the pattern.

−i Ignore upper/lower case distinction during comparisons.

−l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

−n Precede each line by its line number in the file (first line is 1).

−v Print all lines except those that contain the pattern.

−x Print only lines matched entirely.

−e *string*
Search for the given string. Useful is *string* begins with a −).

−f *file*
Take the list of *strings* from *file*.

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

ERRORS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in .BR /usr/include/stdio.h .

NAME

file – determine file type

SYNOPSIS

file [**−c**] [**−f** *ffile*] [**−m** *mfile*] *filename...*

DESCRIPTION

The *file* command tries to classify a file by doing a series of tests. If a file seems to be ASCII, *file* examines the first 512 bytes and tries to guess the language. If a file is an executable **a.out**, *file* prints the version stamp, provided it is greater than 0 (see *ld*(1)).

This version of *file* differentiates between Berkeley (4.2BSD) and MIPS objects and archives.

The *file* command recognizes when files have symbolic links to other files. It lists these files as:

symbolic link to <type>

In the example, <type> shows the type of the file that the symbolic link finally points to (*file* does distinguish symbolic links to other symbolic links). If the symbolic link points to nothing, *file* shows the <type> as "nonexistent filename".

If the **−f** option is given, the next argument is assumed to be a file that contains the names of the files to be examined.

file uses the file **/etc/magic** to identify files that have a *magic number*, that is, any file containing a numeric or string constant that shows its type. Commentary at the beginning of **/etc/magic** explains the format.

The **−m** option instructs *file* to use an alternate magic file.

The **−c** flag causes *file* to check the magic file for format errors. This validation is not normally done for efficiency. No file typing is done under **−c**.

To change the file types that *file* recognizes, you need to change the *file* command itself. The *file* command recognizes these file types:

- English text
- PRESS files
- RCS files
- [nt]roff, tbl, or eqn input text
- ascii text
- ascii text with garbage
- MIPS assembler program text
- block
- C program text
- character
- commands text
- data
- directory
- empty
- executable script for <command>–this refers to files beginning with #!<command>

- fifo
- Fortran program text
- otroff output
- sccs
- socket
- troff intermediate output text

The *file* command reads the /etc/magic file to describe files that can be distinguished by fixed-position data. For example, the first two bytes contain a specific number that distinguishes the file type. The etc/magic file contains these types:

- cpio archive
- ASCII cpio archive
- very old archive
- old archive
- apl workspace
- packed data
- old portable archive
- MMDF mailbox
- executable
- UNIX-rt ldp
- old overlay
- pure executable
- separate I&D
- demand-paged executable
- obsolete text-overlay pure
- obsolete text-overlay separate
- text-overlay pure
- text-overlay separate
- PDP11 kernel overlay
- executable
- pure executable
- BASIC-16 executable
- BASIC-16 executable (TV)
- x86 executable
- x86 executable (TV)
- MC68000 executable
- MC68000 executable (TV)
- 3B20 executable
- 3B20 executable (TV)
- 3B5|DMD executable

- 3B5|DMD executable (TV)
- mipseb
- mipsel
- swapped mipseb
- swapped mipsel
- mipseb ucode
- mipsel ucode
- Berkeley archive random library
- MIPS archive
- archive

The last three files on this list have the same format, except for the existence and name of the first archive element.

SEE ALSO

ld(1), magic(4).

NAME

find – find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

find recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where **+***n* means more than *n*, **−***n* means less than *n* and *n* means exactly *n*. Valid expressions are:

−name *file*	True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and ∗).
[**−perm**] −*onum*	True if the file permission flags exactly match the octal number *onum* (see *chmod(1)*). If *onum* is prefixed by a minus sign, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.
−type *c*	True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **f**, **l**, **p**, or s for block special file, character special file, directory, plain file, symbolic link, fifo, or socket, respectively.
−links *n*	True if the file has *n* links.
−user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.
−group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.
−size *n*[**c**]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
−atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by **find** itself.
−mtime *n*	True if the file has been modified in *n* days.
−ctime *n*	True if the file has been created in *n* days.
−exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{}** is replaced by the current path name.
−ok *cmd*	Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
−print	Always true; causes the current path name to be printed.
−cpio *device*	Always true; write the current file on *device* in *cpio(1)* format (5120-byte records).
−newer *file*	True if the current file has been modified more recently than the argument *file*.
−depth	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when **find** is used with *cpio(1)* to transfer files that are contained in directories without write permission.

−mount	Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.
−local	True if the file physically resides on the local system. The option prevents searches from descending into NFS (Network File System) file systems.
(*expression*)	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).
2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3) Alternation of primaries (**−o** is the *or* operator).

EXAMPLE

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

chmod(1), cpio(1), sh(1), test(1).
stat(2), umask(2), fs(4) in the *Programmer's Reference Manual.*

ERRORS

find / −depth always fails with the message: "find: stat failed: : No such file or directory".

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.

ERRORS

If underlining is present it may be messed up by folding.

NAME

ftoc – interface between prof and cord

SYNOPSIS

ftoc file1 ...

DESCRIPTION

ftoc reads one or more feedback files produced by the **−feedback** option of the profiler *prof*(1) and writes onto stdout a reorder-file for use with the cache-rearranging program *cord*(1). It interprets each feedback file as representing one phase of a program's execution. In other words, if a program behaves in two distinct ways depending on its input, you could create two different feedback files by executing the program twice with different input data, and both *ftoc* and *cord* will understand that the information from the first file is distinct from that of the second file.

As an example, to improve the instruction-cache performance of a program called *hello,* you could generate a new *hello.cord* program by saying:

```
cc -o hello hello.c
pixie -o hello.pixie hello
hello
prof -pixie -feedback hello.feedback hello
ftoc hello.feedback > hello.reorder
cord -o hello.cord hello hello.reorder
```

The reorderfile consists of a list of lines of the form:

```
sourcefile procname.procname... n
```

where "procname.procname..." represents an outer-to-inner list of nested procedures, and *n* is 10 times the percentage of the procedure's "density" with respect to the total of the densities of all procedures. ("Density" is the ratio of a procedure's total cycles to its total static instructions.) A line consisting of "$phase" separates information from different feedback files.

SEE ALSO

cord(1), prof(1)

NAME

ftp – \ARPANT file transfer program

SYNOPSIS

ftp [**−v**] [**−d**] [**−i**] [**−n**] [**−g**] [**host**]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt "ftp>" is provided to the user. The following commands are recognized by *ftp*:

! [*command* [*args*]]	Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.
$ *macro-name* [*args*]	Execute the macro *macro-name* that was defined with the **macdef** command. Arguments are passed to the macro unglobbed.
account [*passwd*]	Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.
append *local-file* [*remote-file*]	Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any *ntrans* or *nmap* setting. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.
ASCII	Set the file transfer *type* to network ASCII. This is the default type.
bell	Arrange that a bell be sounded after each file transfer command is completed.
binary	Set the file transfer *type* to support binary image transfer.
bye	Terminate the FTP session with the remote server and exit *ftp*. An end of file will also terminate the session and exit.
case	Toggle remote computer file name case mapping during **mget** commands. When **case** is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.
cd *remote-directory*	Change the working directory on the remote machine to *remote-directory*.
cdup	Change the remote machine working directory to the parent of the current remote machine working directory.
close	Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.
cr	Toggle carriage return stripping during ASCII type file retrieval. Records are denoted by a carriage return/linefeed sequence during ASCII type file transfer. When **cr** is on (the default), carriage

returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ASCII type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "–>".

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is -, output comes to the terminal.

disconnect
A synonym for **close**.

form *format*
Set the file transfer *form* to *format*. The default format is "file".

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current *case*, *ntrans*, and *nmap* settings. The current settings for *type*, *form*, *mode*, and *structure* are used while transferring the file.

glob
Toggle filename expansion for **mdelete**, **mget** and **mput**. If globbing is turned off with **glob**, the file name arguments are taken literally and not expanded. Globbing for **mput** is done as in **csh**(1). For **mdelete** and **mget**, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing '**mls** *remote-files* -'. Note: **mget** and **mput** are not meant to transfer entire directory subtrees of files. That can be done by transferring a **tar**(1) archive of the subtree (in binary mode).

hash
Toggle hash-sign ("#") printing for each data block transferred. The size of a data block is 1024 bytes.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is -, the output is sent to the terminal.

macdef *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode.

There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a **close** command is executed. The macro processor interprets '$' and '\' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '$'.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like **dir**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mdir** output.

mget *remote-files*
Expand the *remote-files* on the remote machine and do a **get** for each file name thus produced. See **glob** for details on the filename expansion. Resulting file names will then be processed according to *case*, *ntrans*, and *nmap* settings. Files are transferred into the local working directory, which can be changed with '**lcd** directory'; new local directories can be created with '**!** mkdir directory'.

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like **ls**, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving **mls** output.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a **put** for each file in the resulting list. See **glob** for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmap* settings.

nmap [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the **ntrans** and **case** settings). Variable templating is accomplished by including the sequences '$1', '$2', ..., '$9' in *inpattern*. Use '\' to prevent this special treatment of the '$' character. All other characters are treated literally, and are used to determine the **nmap** *inpattern* variable values. For exmaple, given *inpattern* $1.$2 and the remote file name "mydata.data", $1 would have the value

"mydata", and $2 would have the value "data". The *outpattern* determines the resulting mapped filename. The sequences '$1', '$2', ..., '$9' are replaced by any value resulting from the *inpattern* template. The sequence '$0' is replaced by the original filename. Additionally, the sequence '[*seq1*,*seq2*]' is replaced by *seq1* if *seq1* is not a null string; otherwise it is replaced by *seq2*. For example, the command "nmap $1.$2.$3 [$1,$2].[$2,file]" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in *outpattern*, as in the example: nmap $1 |sed "s/ *$//" > $1 . Use the '\' character to prevent special treatment of the '$', '[', ']', and ',' characters.

ntrans [*inchars* [*outchars*]]
Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the file name.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* will attempt to contact an FTP server at that port. If the *auto-login* option is on (default), *ftp* will also attempt to automatically log the user in to the FTP server (see below).

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any **mget** or **mput** will transfer all files, and any **mdelete** will delete all files.

proxy *ftp-command*
Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first **proxy** command should be an **open**, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**: **open** will not define new macros during the auto-login process, **close** will not erase existing macro definitions, **get** and **mget** transfer files from the host on the primary control connection to the host on the secondary control connection, and **put**, **mput**, and **append** transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

put *local-file* [*remote-file*] Store a local file on the remote machine. If *remote-file* is left unspecified, the local file name is used after processing according to any *ntrans* or *nmap* settings in naming the remote file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

pwd Print the name of the current working directory on the remote machine.

quit A synonym for **bye**.

quote *arg1 arg2 ...* The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
A synonym for get.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified it is supplied to the server as well.

rename [*from*] [*to*] Rename the file *from* on the remote machine, to the file *to*.

reset Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be neccesary following a violation of the ftp protocol by the remote server.

rmdir *directory-name* Delete a directory on the remote machine.

runique Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a **get** or **mget** command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that **runique** will not affect local files generated from a shell command (see below). The default value is off.

send *local-file* [*remote-file*]
A synonym for put.

sendport Toggle the use of PORT commands. By default, *ftp* will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, *ftp* will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

status Show the current status of *ftp*.

struct [*struct-name*] Set the file transfer *structure* to *struct-name*. By default "stream" structure is used.

sunique Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

tenex Set the file transfer type to that needed to talk to TENEX machines.

trace Toggle packet tracing.

type [*type-name*] Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*] A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "–" is specified, the **stdin** (for reading) or **stdout** (for writing) is used.

2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell, using *popen*(3S) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""| ls -lt"". A particularly useful example of this mechanism is: "dir |more".

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command. If the *ftp* command expects a single local file (.e.g. **put**), only the first filename generated by the "globbing" operation is used.

4) For **mget** commands and **get** commands with unspecified local file names, the local

filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **sunique** is on.

FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The *type* may be one of "ascii", "image" (binary), "ebcdic", and "local byte size" (for PDP-10's and PDP-20's mostly). *ftp* supports the ASCII and image types of file transfer, plus local byte size 8 for **tenex** mode transfers.

ftp supports only the default values for the remaining file transfer parameters: *mode*, *form*, and *struct*.

OPTIONS

Options may be specified at the command line, or to the command interpreter.

The **−v** (verbose on) option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **−n** option restrains *ftp* from attempting "auto-login" upon initial connection. If auto-login is enabled, *ftp* will check the *.netrc* (see below) file in the user's home directory for an entry describing an account on the remote machine. If no entry exists, *ftp* will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The **−i** option turns off interactive prompting during multiple file transfers.

The **−d** option enables debugging.

The **−g** option disables file name globbing.

THE .netrc FILE

The *.netrc* file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

machine *name*
Identify a remote machine name. The auto-login process searches the .netrc file for a **machine** token that matches the remote machine specified on the *ftp* command line or as an **open** command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end of file is reached or another **machine** token is encountered.

login *name*
Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the .netrc file, *ftp* will abort the auto-login process if the .netrc is readable by anyone besides the user.

account *string*
Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not.

macdef *name*

Define a macro. This token functions like the *ftp* **macdef** command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

ERRORS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2 BSD UNIX ASCII-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using the ASCII type. Avoid this problem by using the binary image type.

ftp 's get has wrong modes for received files. The files belong to root. Use *rcp* rather than *ftp* if at all possible.

ORIGIN

4.3 BSD

NAME

get – get a version of an SCCS file

SYNOPSIS

get [**−r**SID] [**−c**cutoff] [**−i**list] [**−x**list] [**−w**string] [**−a**seq-no.] [**−k**] [**−e**] [**−l**[**p**] [**−p**] [**−m**] [**−n**] [**−s**] [**−b**] [**−g**] [**−t**] file ...

DESCRIPTION

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

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

−r*SID* The *S*CCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table~1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the **−e** keyletter is also used), as a function of the SID specified.

−c*cutoff* *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **−c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "**−c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

```
~!get "-c%E% %U%" s.file
```

−i*list* A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID − SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1.

−x*list* A *list* of deltas to be excluded in the creation of the generated file. See the **−i** keyletter for the *list* format.

−e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The **−e** keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file [see *admin*(1)]. Concurrent use of **get −e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **−e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **−k** keyletter in place of the **−e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file [see *admin*(1)] are enforced when the **−e** keyletter is used.

−b Used with the **−e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file [see *admin*(1)] or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.) Note: A branch *delta* may always be created from a non-leaf *delta*. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

−k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The **−k** keyletter is implied by the **−e** keyletter.

−l[p] Causes a delta summary to be written into an *l-file*. If **−lp** is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.

−p Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the **−s** keyletter is used, in which case it disappears.

−s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

−m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

−n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the **−m** and **−n** keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the **−m** keyletter generated format.

−g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.

−t Used to access the most recently created delta in a given release (e.g., **−r1**), or release and level (e.g., **−r1.2**).

−w *string* Substitute *string* for all occurrences of %W% when getting the file.

−a*seq-no*. The delta sequence number of the SCCS file delta (version) to be retrieved [see *sccsfile*(4)]. This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter. If both the **−r** and **−a** keyletters are specified, only the **−a** keyletter is used. Care should be taken when using the **−a** keyletter in conjunction with the **−e** keyletter, as the SID of the delta to be created may not be what one expects. The **−r** keyletter can be used with the **−a** and **−e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **−e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the **−i** keyletter is used included deltas are listed following the notation "Included"; if the **−x** keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	**−b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	−	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	−	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	−	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	−	Branch succ.	R.L.B.S	R.L.(mB+1).1

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

Successor.

† The **−b** keyletter is effective only if the **b** flag [see *admin*(1)] is present in the file. An entry of **−** means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	*Value*
%M%	Module name: either the value of the **m** flag in the file [see *admin*(1)], or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file [see *admin*(1)].
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file [see *admin*(1)].
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX system program files. %A% = %Z%%Y% %M% %I%%Z%

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

The *g-file*, which contains the generated text, is created in the current directory (unless the **−p** keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the **−k** keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the **−l** keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

a. A blank character if the delta was applied;
 ∗ otherwise.

b. A blank character if the delta was applied or was not applied and ignored;

∗ if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
   "I": Included.
   "X": Excluded.
   "C": Cut off (by a **−c** keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created *delta*.

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

The *p-file* is used to pass information resulting from a *get* with an **−e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **−e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, [see *admin*(1)] is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **−i** keyletter argument if it was present, followed by a blank and the **−x** keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

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

FILES

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*.

SEE ALSO

admin(1), delta(1), prs(1), what(1).
help(1) in the *User's Reference Manual*.

DIAGNOSTICS

Use *help*(1) for explanations.

ERRORS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the **−e** keyletter is used.

NAME

getopt – parse command options

SYNOPSIS

set -- `getopt** optstring **$*`

DESCRIPTION

WARNING: Start using the new command *getopts*(1) in place of *getopt*(1). *getopt*(1) will not be supported in the next major release. For more information, see the **WARNINGS** section, below.

getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *optstring* is a string of recognized option letters (see *getopt*(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option **--** is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters ($1 $2 . . .) of the shell are reset so that each option is preceded by a **–** and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)        FLAG=$i; shift;;
        -o)             OARG=$2; shift 2;;
        --)             shift; break;;
        esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

getopts(1), sh(1).
getopt(3C) in the *Programmer's Reference Manual*.

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

WARNINGS

getopt(1) does not support the part of Rule 8 of the command syntax standard (see *intro*(1)) that permits groups of option-arguments following an option to be separated by white space and quoted. For example,

```
cmd –a –b –o "xxx z yy" file
```

is not handled correctly). To correct this deficiency, use the new command *getopts*(1) in place of *getopt*(1).

getopt(1) will not be supported in the next major release. For this release a conversion tool has been provided, *getoptcvt*. For more information about *getopts* and *getoptcvt*, see the *getopts*(1) manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but using the following command line: cmd -o -a file, *getopt* will always treat **–a** as an option-argument to **–o**; it will never recognize **–a** as an option. For this case, the **for** loop in the example will shift past the *file* argument.

NAME

getopts, getoptcvt – parse command options

SYNOPSIS

getopts optstring name [arg . . .]

/usr/lib/getoptcvt [**−b**] file

DESCRIPTION

getopts is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard (see Rules 3-10, *intro*(1)). It should be used in place of the *getopt*(1) command. (See the **WARNING**, below.)

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "**−−**" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg* . . .) are given on the *getopts* command line, *getopts* will parse them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use *getopts*(1) instead of *getopt*(1), and writes the results on the standard output.

−b the results of running */usr/lib/getoptcvt* will be portable to earlier releases of the UNIX system. */usr/lib/getoptcvt* modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(1) or *getopt*(1).

So all new commands will adhere to the command syntax standard described in *intro*(1), they should use *getopts*(1) or *getopt*(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS**, below).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while getopts abo: c
do
        case $c in
        a | b)          FLAG=$c;;
        o)              OARG=$OPTARG;;
        \?)             echo $USAGE
                        exit 2;;
        esac
done
shift `expr $OPTIND − 1`
```

This code will accept any of the following as equivalent:

```
cmd −a −b −o "xxx z yy" file
cmd −a −b −o "xxx z yy" −− file
cmd −ab −o xxx,z,yy file
cmd −ab −o "xxx z yy" file
cmd −o xxx,z,yy −b −a file
```

SEE ALSO

intro(1), sh(1).
getopt(3C) in the *Programmer's Reference Manual*.
UNIX System V Release 3.0 Release Notes.

WARNING

Although the following command syntax rule (see *intro*(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

```
cmd −aboxxx file  (Rule 5 violation:  options with
        option-arguments must not be grouped with other options)
cmd −ab −oxxx file  (Rule 6 violation:  there must be
        white space after an option that takes an option-argument)
```

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

glossary – definitions of common UNIX system terms and symbols

SYNOPSIS

[**help**] **glossary** [term]

DESCRIPTION

The UNIX system Help Facility command *glossary* provides definitions of common technical terms and symbols.

Without an argument, *glossary* displays a menu screen listing the terms and symbols that are currently included in *glossary.* A user may choose one of the terms or may exit to the shell by typing q (for "quit"). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition may also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following is a table which list the symbols and their escape sequence.

SYMBOL	ESCAPE SEQUENCE
""	\"\"
’’	\’\’
[]	\\[\\]
‘‘	\‘\‘
#	\#
&	\&
*	*
\	\\\\
|	\|

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see *profile*(4)): “export SCROLL ; SCROLL=no”. If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

help(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1).
term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the shell variable **TERM** (see *sh*(1)) is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

grep – search a file for a pattern

SYNOPSIS

grep [options] limited regular expression [file ...]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. *grep* uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed (1)* to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes **'** . . . **'**.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

−b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
−c	Print only a count of the lines that contain the pattern.
−i	Ignore upper/lower case distinction during comparisons.
−l	Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
−n	Precede each line by its line number in the file (first line is 1).
−s	Suppress error messages about nonexistent or unreadable files
−v	Print all lines except those that contain the pattern.

SEE ALSO

ed(1), egrep(1), fgrep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

ERRORS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.
If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

gprof – display call graph profile data

SYNOPSIS

gprof [options] [a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of C, Pascal, or Fortran77 programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs which are compiled with the **–pg** option of *cc*, *pc*, and *f77*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

The following options are available:

–a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (*e.g.*, time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

–b supresses the printing of a description of each field in the profile.

–c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0.

–e *name*
suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **–e** option may be given. Only one *name* may be given with each **–e** option.

–E *name*
suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **–e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **–E** *mcount* **–E** *mcleanup* is the default.)

–f *name*
prints the graph profile entry of only the specified routine *name* and its descendants. More than one **–f** option may be given. Only one *name* may be given with each **–f** option.

–F *name*
prints the graph profile entry of only the routine *name* and its descendants (as **–f**, above) and also uses only the times of the printed routines in total time and

percentage computations. More than one **−F** option may be given. Only one *name* may be given with each **−F** option. The **−F** option overrides the **−E** option.

−s a profile file *gmon.sum* is produced which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of gprof (probably also with a **−s**) to accumulate profile data across several runs of an *a.out* file.

−z displays routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **−c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)
"gprof: A Call Graph Execution Profiler", by Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the gmon.out file.

NAME

havetcp – test system for TCP support

SYNOPSIS

havetcp

DESCRIPTION

havetcp exits with a 0 (true) if the operating system supports the *socket(2)* system call, and a nonzero value otherwise.

SEE ALSO

socket(2)

NAME

help – UNIX system Help Facility

SYNOPSIS

help
[**help**] **starter**
[**help**] **usage** [**–d**] [**–e**] [**–o**] [command_name]
[**help**] **locate** [keyword1 [keyword2] ...]
[**help**] **glossary** [term]
help arg ...

DESCRIPTION

The UNIX system Help Facility provides on-line assistance for UNIX system users, whether they desire general information or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, *help* prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are:

COMMAND	DESCRIPTION
starter	information about the UNIX system for the beginning user
locate	locate UNIX system commands using function-related keywords
usage	UNIX system command usage information
glossary	definitions of UNIX system technical terms

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

With arguments, *help* directly invokes the named on-line assistance command, bypassing the initial *help* menu. The commands *starter*, *locate*, *usage*, and *glossary*, optionally preceded by the word *help*, may also be specified at shell level. When executing *glossary* from shell level some of the symbols listed in the glossary must be escaped (preceded by one or more backslashes, "\") to be understood by the Help Facility. For a list of symbols refer and how many backslashes to use for each, refer to the *glossary*(1) manual page.

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see *profile*(4)): "export SCROLL ; SCROLL=no". If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

The Help Facility can be tailored to a customer's needs by use of the *helpadm*(1M) command.

If the first argument to *help* is different from *starter*, *usage*, *locate*, or *glossary*, *help* assumes information is being requested about the SCCS Facility. The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., ge3 for message 3 from the *get* command).

type2 Does not contain numerics (as a command, such as get).

type3 Is all numeric (e.g., 212).

SEE ALSO

glossary(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1).
admin(1), cdc(1), comb(1), delta(1), get(1), prs(1), rmdel(1), sact(1), sccsdiff(1), unget(1), val(1), vc(1), what(1), profile(4), sccsfile(4), term(5) in the *Programmer's Reference Manual*.

WARNINGS

If the shell variable **TERM** (see *sh*(1)) is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

hostid – set or print identifier of current host system

SYNOPSIS

hostid [identifier]

DESCRIPTION

The *hostid* command prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is commonly set to the host's Internet address. The super-user can set the hostid by giving a hexadecimal number, an internet host address, or the hostname; this is usually done in the startup script /etc/rc2.d/S30tcp.

SEE ALSO

gethostid(2)

NAME

hostname − set or print name of current host system

SYNOPSIS

hostname [nameofhost]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt. The super-user can set the hostname by giving an argument; this is usually done in the startup script /etc/rc2.d/S20sysetup.

SEE ALSO

gethostname(2)

NAME

ident – identify files

SYNOPSIS

ident file ...

DESCRIPTION

ident searches the named files for all occurrences of the pattern *$keyword:...$*, where *keyword* is one of

```
Author
Date
Header
Locker
Log
Revision
Source
State
```

These patterns are normally inserted automatically by the RCS command *co (1)*, but can also be inserted manually.

Ident works on text files as well as object files. For example, if the C program in file f.c contains

```
char rcsid[] = "$Header:  Header information $";
```

and f.c is compiled into f.o, then the command

```
ident  f.c  f.o
```

will print

```
f.c:
        $Header:  Header information $
f.o:
        $Header:  Header information $
```

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.5 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci (1), co (1), rcs (1), rcsdiff(1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig *interface address_family* [*address* [*dest_address*]] [*parameters*]
/etc/ifconfig *interface* [*protocol_family*]

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. The *interface* parameter is a string of the form "name unit", e.g. "en0".

Since an interface may receive transmissions in differing protocols, each of which may require separate naming schemes, it is necessary to specify the *address_family*, which may change the interpretation of the remaining parameters. The address families currently supported are "inet" and "ns".

For the DARPA-Internet family, the address is either a host name present in the host name data base, *hosts(4),* or a DARPA Internet address expressed in the Internet standard "dot notation". For the Xerox Network Systems(tm) family, addresses are *net:a.b.c.d.e.f*, where *net* is the assigned network number (in decimal), and each of the six bytes of the host number, *a* through *f*, are specified in hexadecimal. The host number may be omitted on 10Mb/s Ethernet interfaces, which use the hardware physical address, and on interfaces other than the first.

The following parameters may be set with **ifconfig**:

up	Mark an interface "up". This may be used to enable an interface after an "ifconfig down." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.
trailers	Request the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see *arp(7P)*; currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.
−trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.
−arp	Disable the use of the Address Resolution Protocol.
metric *n*	Set the routing metric of the interface to *n*, default 0. The routing

	metric is used by the routing protocol (*routed(1m)*). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.
rarp	Enable the use of the Reverse Address Resolution Protocol when mapping link level addresses and network level addresses. Ethernet addresses and DARPA Internet addresses.
-rarp	Disable the use of the Reverse Addresses Resolution Protocol (Default).
promarp	Enable proxy replies to Address Resolution Protocol requests. This permits machines connected to multiple 10 mb/s Ethernets to respond to ARP requests for hosts on other Ethernets in a proxy fashion.
-promarp	Disable the use of promiscuous ARP (Deault).
debug	Enable driver dependent debugging code; usually, this turns on extra console error logging.
−debug	Disable driver dependent debugging code.
netmask *mask*	(Inet only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table. The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.
dstaddr	Specify the address of the correspondent on the other end of a point to point link.
broadcast	(Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.
ipdst	(NS only) This is used to specify an Internet host who is willing to receive ip packets encapsulating NS packets bound for a remote network. In this case, an apparent point to point link is constructed, and the address specified will be taken as the NS address and network of the destinee.

ifconfig displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, **ifconfig** will report only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

netstat(1), intro(7N), rc(1M)

ORIGIN

4.3 BSD

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands available in UMIPS-V. Certain distinctions of purpose are made in the headings.

Manual Page Command Syntax

Unless otherwise noted, commands described in the **SYNOPSIS** section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

name [−*option*...] [*cmdarg*...]
where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "−".) *noargletter*... or, *argletter optarg*[,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "−" (Rule 5, below).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (Rule 8, below).
cmdarg	Path name (or other command argument) *not* beginning with "−", or "−" by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. *getopts*(1) should be used by all shell procedures to parse positional parameters and to check for legal options. It supports Rules 3-10 below. The enforcement of the other rules must be done by the command itself.

1. Command names (*name* above) must be between two and nine characters long.
2. Command names must include only lower-case letters and digits.
3. Option names (*option* above) must be one character long.
4. All options must be preceded by "−".
5. Options with no arguments may be grouped after a single "−".
6. The first option-argument (*optarg* above) following an option must be preceded by white space.
7. Option-arguments cannot be optional.
8. Groups of option-arguments following an option must either be separated by commas or separated by white space and quoted (e.g., −o xxx,z,yy or −o "xxx z yy").
9. All options must precede operands (*cmdarg* above) on the command line.

10. "--" may be used to indicate the end of the options.
11. The order of the options relative to one another should not matter.
12. The relative order of the operands (*cmdarg* above) may affect their significance in ways determined by the command with which they appear.
13. "-" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

getopts(1).
exit(2), wait(2), getopt(3C) in the *Programmer's Reference Manual*.

DIAGNOSTICS

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

WARNINGS

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

NAME

ipcrm – remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

−q *msqid*	removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.
−m *shmid*	removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
−s *semid*	removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.
−Q *msgkey*	removes the message queue identifier, created with key *msgkey*, from the system and destroys the message queue and data structure associated with it.
−M *shmkey*	removes the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.
−S *semkey*	removes the semaphore identifier, created with key *semkey*, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in *msgctl*(2), *shmctl*(2), and *semctl*(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1).
msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2) in the *Programmer's Reference Manual*.

NAME

ipcs – report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q	Print information about active message queues.
−m	Print information about active shared memory segments.
−s	Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed subject to these options:

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

−c Print creator's login name and group name. See below.

−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

−a Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

−N *namelist*
The argument will be taken as the name of an alternate *namelist* (**/unix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

q	message queue;
m	shared memory segment;
s	semaphore.

ID (all) The identifier for the facility entry.

KEY	(all)	The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)
MODE	(all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

R	if a process is waiting on a *msgrcv*;
S	if a process is waiting on a *msgsnd*;
D	if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
C	if the associated shared memory segment is to be cleared when the first attach is executed;
–	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r	if read permission is granted;
w	if write permission is granted;
a	if alter permission is granted;
–	if the indicated permission is *not* granted.

OWNER	(all)	The login name of the owner of the facility entry.
GROUP	(all)	The group name of the group of the owner of the facility entry.
CREATOR	(a,c)	The login name of the creator of the facility entry.
CGROUP	(a,c)	The group name of the group of the creator of the facility entry.
CBYTES	(a,o)	The number of bytes in messages currently outstanding on the associated message queue.
QNUM	(a,o)	The number of messages currently outstanding on the associated message queue.
QBYTES	(a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	(a,p)	The process ID of the last process to send a message to the associated queue.
LRPID	(a,p)	The process ID of the last process to receive a message from the associated queue.
STIME	(a,t)	The time the last message was sent to the associated queue.
RTIME	(a,t)	The time the last message was received from the associated queue.
CTIME	(a,t)	The time when the associated entry was created or changed.
NATTCH	(a,o)	The number of processes attached to the associated shared memory segment.
SEGSZ	(a,b)	The size of the associated shared memory segment.
CPID	(a,p)	The process ID of the creator of the shared memory entry.
LPID	(a,p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME	(a,t)	The time the last attach was completed to the associated shared memory segment.
DTIME	(a,t)	The time the last detach was completed on the associated shared memory

		segment.
NSEMS	(a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME	(a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

SEE ALSO

msgop(2), semop(2), shmop(2) in the *Programmer's Reference Manual.*

ERRORS

Things can change while *ipcs* is running; the picture it gives is only a close approximation to reality.

NAME

join – relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is **–**, the standard input is used.

file1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see *sort*(1)].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

–a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2. Multiple **–a** options may be given.

–e *s* Replace empty output fields by string *s*.

–j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with **1**.

–o *list* Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.

–t*c* Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

ERRORS

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous. Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill – terminate a process

SYNOPSIS

kill [**–**signo] PID ...

DESCRIPTION

kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with **&** is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

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

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by **–** is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill –9 . . ." is a sure kill.

SEE ALSO

ps(1), sh(1).
kill(2), signal(2) in the *Programmer's Reference Manual*.

NAME

ld – MIPS link editor
uld – ucode link editor

SYNOPSIS

ld [option] ... file ...
uld [option] ... file ...

DESCRIPTION

Ld, the MIPS link editor, runs on MIPS machines under the UNIX system 4.3bsd and System V. It links MIPS extended *coff* object files. The archive format understood by *ld* is the one created by the MIPS archiver *ar*(1).

The *ld* command combines several object files into one, preforms relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object *files* are given. *Ld* combines them, producing an object module that can be executed or used as input for a subsequent *ld* run. (In the latter case, the **−r** option must be given to preserve the relocation entries.) The output of *ld* is left in **a.out**. By default, this file is executable if no errors occurred during the load.

The argument object files are concatenated in the order specified. The entry point of the output is the beginning of the text segment (unless the **−e** option is specified).

The *uld* command combines several ucode object files and libraries into one ucode object file. It "hides" external symbols for better optimizations by subsequent compiler passes. The symbol tables of *coff* object files loaded with ucode object files are used to determine what external symbols not to "hide" along with files specified by the user that contain lists of symbol names.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see *ar*(1)) is a hash table and is searched to resolved external references that can be satisfied by library members. The ordering of library members is unimportant.

The following options are recognized by both *ld* and *uld* . Those options used by one and not the other are ignored. Any option can be preceded by a '**k**' (for example **−ko** outfile) and except for **−kl***x* have the same meaning with or without the preceding '**k**'. This is done so that these options can be passed to both link editors through compiler drivers.

When searching for libraries the default directories searched are /lib/, /usr/lib/cmplrs/cc, /usr/lib/ and /usr/local/lib/ . If the target byte ordering of the object files being loaded is of the opposite byte ordering of the machine the link editor is running on then the default search directories for libraries are changed. The change is to replace the last name of the directories from "lib/" to "libeb/" or "libel/" to match the target byte ordering of the objects being loaded.

The symbols 'etext', 'edata', 'end', '_ftext', '_fdata', '_fbss', '_gp', '_procedure_table', '_procedure_table_size' and '_procedure_string_table' are reserved. These loader defined symbols if referred to, are set their values as described in *end*(3). It is erroneous to define these symbols.

−o outfile
: Produce an output object file by the name *outfile.* The name of the default object file is **a.out.**

−l*x* Search a library **lib***x***.a,** where *x* is a string. A library is searched when its name is encountered, so the placement of a **−l** is significant.

−kl*x* Search a library **lib***x***.b,** where *x* is a string. These libraries are intended to be ucode object libraries. In all other ways, this option is like the **−l***x* option.

−L*dir* Change the algorithm of searching for **lib***x***.a** or **lib***x***.b** to look in *dir* before looking in the default directories. This option is effective only if it precedes the **−l** options on the command line.

−L Change the algorithm of searching for **lib***x***.a** or **lib***x***.b** to **never** look in the default directories. This is useful when the default directories for libraries should not be searched and only the directories specified by **−L***dir* are to be searched.

−K*dir* Change the default directories to the single directory *dir*. This option is only intended to be used by the compiler driver. Users should use the **−L** and **−L***dir* options to get the effect they desire.

−B*string*
Append *string* to the library names created for the **−l***x* and **−kl***x* when searching for library names. For each directory to be searched the name is first created with the *string* and if it is not found it is created without the *string*.

−p file Preserve (don't "hide") the symbol names listed in *file* when loading ucode object files. The symbol names in the file are separated by blanks, tabs, or newlines.

−s Strip the symbolic information from the output object file.

−x Do not preserve local (non-.globl) symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

−r Retain relocation entries in the output file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. This option also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.

−d Force definition of common storage and define loader defined symbols even if -r is present.

−u symname
Enter *symname* as an undefined in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

−F or **−z**
Arrange for the process to be loaded on demand from the resulting executable file (413 format) rather than preloaded, a ZMAGIC file. This is the default.

−n Arrange (by giving the output file a 0410 "magic number") that when the output file is executed, the text portion will be read-only and shared among all users executing the file, an NMAGIC file. This involves moving the data areas up to the first possible *pagesize* byte boundary following the end of the text.

−N Place the data section immediately after the text and do not make the text portion read only or sharable, an OMAGIC file. (Use "magic number" 0407.)

−T num
Set the text segment origin. The argument *num* is a hexadecimal number. See the notes section for restrictions.

−D num
Set the data segment origin. The argument *num* is a hexadecimal number. See the notes section for restrictions.

−B num
Set the bss segment origin. The argument *num* is a hexadecimal number. This option

can be used only if the final object is an OMAGIC file.

−e epsym
Set the default entry point address for the output file to be that of the symbol *epsym*.

−m Produce a map or listing of the input/output sections on the standard output (UNIX system V-like map).

−M Produce a primitive load map, listing the names of the files that will be loaded (UNIX 4.3bsd-like map).

−S Set silent mode and suppress non-fatal errors.

−v Set verbose mode. Print the name of each file as it is processed.

−ysym Indicate each file in which *sym* appears, *sym*'s type and whether the file defines or references *sym*. Many such options may be given to trace many symbols.

−V Print a message giving information about the version of *ld* being used.

−VS num
Use **num** as the decimal version stamp to identify the a.out file that is produced. The version stamp is stored in the optional and symbolic headers.

−f fill Set the fill pattern for "holes" within an output section. The argument *fill* is a four-byte hexadecimal constant.

−G num
The argument *num* is taken to be a decimal number that is the largest size in bytes of a *.comm* item or literal that is to be allocated in the small bss section for reference off the global pointer. The default is 8 bytes.

−bestGnum
Calculate the best **−G** *num* to use when compiling and linking the files which produced the objects being linked. Using too large a number with the **−G** *num* option may cause the gp (global-pointer) data area to overflow; using too small a number may reduce your program's execution speed.

−count, **−nocount**, **−countall**
These options control which objects are counted as recompilable for the best **−G** *num* calculation. By default, the **−bestGnum** option assumes you can recompile everything with a different **−G** *num* option. If you cannot recompile certain object files or libraries (because, for example, you have no sources for them), use these options to tell the link editor to take this into account in calculating the best **−G** *num* value. **−nocount** says that object files appearing after it on the command line cannot be recompiled; **−count** says that object files appearing after it on the command line can be recompiled; you can alternate the use of **−nocount** and **−count. −countall** overrides any **−nocount** options appearing after it on the command line.

−b Do not merge the symbolic information entries for the same file into one entry for that file. This is only needed when the symbolic information from the same file appears differently in any of the objects to be linked. This can occur when object files are compiled, by means of conditional compilation, with an apparently different version of an include file.

−jmpopt and **−nojmpopt**
Fill or don't fill the delay slots of jump instructions with the target of the jump and adjust the jump offset to jump past that instruction. This allways is disabled for debugging (when the **−g1, −g2** or **−g** flag is present). When this option is enabled it requires that all of the loaded program's text be in memory and could cause the loader to run out of memory. The default is **−nojmpopt.**

−g or **−g[0123]**
These options are accepted and except for **−g1, −g2** or **−g** disabling the **−jmpopt** have no other effect.

−A *file* This option specifies incremental loading, i.e. linking is to be done in a manner so that the resulting object may be read into an already executing program. The next argument, *file,* is the name of a file whose symbol table will be taken as a basis on which to define additional symbols. Only newly linked material will be entered into the text and data portions of **a.out,** but the new symbol table will reflect every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. The **−T** option may be used as well, and will be taken to mean that the newly linked segment will commence at the corresponding address (which must be a correct multiple for the resulting object type). The default resulting object type is an OMAGIC file and the default starting address of the text is the old value of end rounded to SCNROUND as defined in the include file <*scnhdr.h*>. Using the defaults, when this file is read into an already executing program the intial value of the break must also be rounded. All other objects except the argument to the **−A** option must be compiled **−G** 0 and this sets **−G** 0 for linking.

The following options are used by the command *mkshlib*(1) and are not intended for general use.

−c Create a target shared library object file. This is a LIBMAGIC file (443 format). The objects linked must be compiled with **−G** 0 and this sets **−G** 0 for linking. This file is demand paged and the headers are part of the text but on there own page so real text starts on the next page where the text is loaded.

−i *file* The .text section of *file* is moved into the .init section of the resulting object file.

Ld and *uld* accept object files targeted for either byte ordering with their headers and symbolic tables in any byte ordering; however *ld* and *uld* are faster if the headers and symbolic tables have the byte ordering of the machine that they are running on. The default byte ordering of the headers and symbolic tables is the target byte ordering of the output object file. For non-relocatable object files the default byte ordering of the headers and symbolic tables can't be changed.

−EB Produce the output object file with big-endian byte ordered headers and symbolic information tables.

−EL Produce the output object file with little-endian byte ordered headers and symbolic information tables.

FILES

/lib/lib*.a	
/usr/lib/lib*.a	
/usr/local/lib/lib*.a	libraries
a.out	output file

SEE ALSO

cc(1), pc(1), f77(1), as(1), ar(1)

NOTES

Any of the three types of objects can be run on UMIPS-BSD or UMIPS-V systems. On both systems the segments must not overlap and all addresses must be less than 0x80000000. The stack starts below 0x80000000 and grows through lower addresses so space should be left for it. For ZMAGIC and NMAGIC files the default text segment address is 0x00400000 and the default data segment address is 0x10000000. For OMAGIG files the default text segment address is 0x10000000 with the data segment following the text segment. The default for all types of files is that the bss segment follows the data segment.

For OMAGIC files to be run under the operating system the -B flag should not be used because the bss segment must follow the data segment which is the default.

Under UMIPS-BSD the segments must be on 4 megabyte boundaries. Objects linked at addresses other than the default will run under the 2.0 and later UMIPS-BSD releases.

Under UMIPS-V the segments must be on 2 megabyte boundaries. OMAGIC files will run under the 1.1 and later UMIPS-V releases.

NAME

less – file browser

SYNOPSIS

less [-cdepstwmMqQuU] [-h*N***] [-b[fp]***N***] [-x***N***] [-[z]***N***]**
[-P[mM]*string***] [-l***logfile***] [+***cmd***] [***filename***]...**

DESCRIPTION

less is a program similar to *more(1). less* does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like *vi(1). less* uses termcap, so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top of the screen are prefixed with an up-arrow.)

Commands are based on both *more* and *vi.* Commands may be preceeded by a decimal number, called N in the descriptions below. The number is used by some commands, as indicated.

COMMANDS

In the following descriptions, ^X means control-X.

h　Help: display a summary of these commands. If you forget all the other commands, remember this one.

SPACE
Scroll forward N lines, default one window (see option –z below). If N is more than the screen size, only the final screenful is displayed.

f or ^F　Same as SPACE.

b or ^B　Scroll backward N lines, default one window (see option –z below). If N is more than the screen size, only the final screenful is displayed.

RETURN
Scroll forward N lines, default 1. The entire N lines are displayed, even if N is more than the screen size.

e or ^E　Same as RETURN.

j or ^J　Also the same as RETURN.

y or ^Y　Scroll backward N lines, default 1. The entire N lines are displayed, even if N is more than the screen size.

k or ^K
Same as y.

d or ^D
Scroll forward N lines, default 10. If N is specified, it becomes the new default for subsequent d and u commands.

u or ^U
Scroll backward N lines, default 10. If N is specified, it becomes the new default for subsequent d and u commands.

r or ^R or ^L
Repaint the screen.

R　Repaint the screen, discarding any buffered input. Useful if the file is changing while it is being viewed.

g　Go to line N in the file, default 1 (beginning of file). (Warning: this may be slow if N is large.)

G　Go to line N in the file, default the end of the file. (Warning: this may be slow if standard input, rather than a file, is being read.)

p　Go to a position N percent into the file. N should be between 0 and 100. (This is possible if standard input is being read, but only if *less* has already read to the end of the file. It is always fast, but not always useful.)

%　Same as p.

m　Followed by any lowercase letter, marks the current position with that letter.

'　(Single quote.) Followed by any lowercase letter, returns to the position which was previously marked with that letter. Followed by another single quote, returns to the postion at which the last "large" movement command was executed. All marks are lost when a new file is examined.

/pattern
　Search forward in the file for the N-th line containing the pattern. N defaults to 1. The pattern is a regular expression, as recognized by *ed*. The search starts at the second line displayed (but see the −t option, which changes this).

?pattern
　Search backward in the file for the N-th line containing the pattern. The search starts at the line immediately before the top line displayed.

n　Repeat previous search, for N-th line containing the last pattern.

E　Examine a new file. If the filename is missing, the "current" file (see the N and P commands below) from the list of files in the command line is re-examined.

N　Examine the next file (from the list of files given in the command line). If a number N is specified (not to be confused with the command N), the N-th next file is examined.

P　Examine the previous file. If a number N is specified, the N-th previous file is examined.

= or ^G
　Prints some information about the file being viewed, including its name and the byte offset of the bottom line being displayed. If possible, it also prints the length of the file and the percent of the file above the last displayed line.

−　Followed by one of the command line option letters (see below), this will toggle the setting of that option and print a message describing the new setting.

+cmd　Causes the specified cmd to be executed each time a new file is examined. For example, +G causes *less* to initially display each file starting at the end rather than the beginning.

V　Prints the version number of *less* being run.

q　Exits *less*.

The following two commands may or may not be valid, depending on your particular installation.

v　Invokes an editor to edit the current file being viewed. The editor is taken from the environment variable EDITOR, or defaults to "vi".

! shell-command
　Invokes a shell to run the shell-command given.

OPTIONS

Command line options are described below. Most options may be changed while *less* is running, via the "–" command.

Options are also taken from the environment variable "LESS". For example, if you like more-style prompting, to avoid typing "less –m ..." each time *less* is invoked, you might tell *csh:*

```
setenv LESS m
```

or if you use *sh:*

```
LESS=m; export LESS
```

The environment variable is parsed before the command line, so command line options override the LESS environment variable. A dollar sign ($) may be used to signal the end of an option string. This is important only for options like –P which take a following string.

–s　The –s option causes consecutive blank lines to be squeezed into a single blank line. This is useful when viewing *nroff* output.

–t　Normally, forward searches start just after the top displayed line (that is, at the second displayed line). Thus forward searches include the currently displayed screen. The –t option causes forward searches to start just after the bottom line displayed, thus skipping the currently displayed screen.

–m　Normally, *less* prompts with a colon. The –m option causes *less* to prompt verbosely (like *more),* with the percent into the file.

–M　The –M option causes *less* to prompt even more verbosely than *more.*

–P　The –P option provides a way to tailor the three prompt styles to your own preference. You would normally put this option in your LESS environment variable, rather than type it in with each less command. Such an option must either be the last option in the LESS variable, or be terminated by a dollar sign. –P followed by a string changes the default (short) prompt to that string. –Pm changes the medium (–m) prompt to the string, and –PM changes the long (–M) prompt. The string consists of a sequence of letters which are replaced with certain predefined strings, as follows:

F	file name
f	file name, only once
O	file n of n
o	file n of n, only once
b	byte offset
p	percent into file
P	percent if known, else byte offset

Angle brackets, < and >, may be used to surround a literal string to be included in the prompt. The defaults are "fo" for the short prompt, "foP" for the medium prompt, and "Fobp" for the long prompt.
Example: Setting your LESS variable to "PmFOP$PMFObp" would change the medium and long prompts to always include the file name and "file n of n" message.
Another example: Setting your LESS variable to
"mPm<--Less-->FoPe" would change the medium prompt to the string "--Less--" followed by the file name and percent into the file. It also selects the medium prompt as the default prompt (because of the first "m").

–q　Normally, if an attempt is made to scroll past the end of the file or before the beginning of the file, the terminal bell is rung to indicate this fact. The –q option tells *less* not to ring the bell at such times. If the terminal has a "visual bell", it is used instead.

−Q Even if −q is given, *less* will ring the bell on certain other errors, such as typing an invalid character. The −Q option tells *less* to be quiet all the time; that is, never ring the terminal bell. If the terminal has a "visual bell", it is used instead.

−e Normally the only way to exit less is via the "q" command. The −e option tells less to automatically exit the second time it reaches end-of-file.

−u If the −u option is given, backspaces are treated as printable characters; that is, they are sent to the terminal when they appear in the input.

−U If the −U option is given, backspaces are printed as the two character sequence "^H".

If neither −u nor −U is given, backspaces which appear adjacent to an underscore character are treated specially: the underlined text is displayed using the terminal's hardware underlining capability. Also, backspaces which appear between two identical characters are treated specially: the overstruck text is printed using the terminal's hardware boldface capability. Other backspaces are deleted, along with the preceeding character.

−w Normally, *less* uses a tilde character to represent lines past the end of the file. The −w option causes blank lines to be used instead.

−d Normally, *less* will complain if the terminal is dumb; that is, lacks some important capability, such as the ability to clear the screen or scroll backwards. The −d option suppresses this complaint (but does not otherwise change the behavior of the program on a dumb terminal).

−p Normally, *less* will repaint the screen by scrolling from the bottom of the screen. If the −p option is set, when *less* needs to change the entire display, it will clear the screen and paint from the top line down.

−h Normally, *less* will scroll backwards when backwards movement is necessary. The −h option specifies a maximum number of lines to scroll backwards. If it is necessary to move backwards more than this many lines, the screen is repainted in a forward direction. (If the terminal does not have the ability to scroll backwards, −h0 is implied.)

−[z] When given a backwards or forwards window command, *less* will by default scroll backwards or forwards one screenful of lines. The −z*n* option changes the default scrolling window size to *n* lines. If *n* is greater than the screen size, the scrolling window size will be set to one screenful. Note that the "z" is optional for compatibility with *more*.

-x The -x*n* option sets tab stops every *n* positions. The default for *n* is 8.

-l The -l option, followed immediately by a filename, will cause *less* to copy its input to the named file as it is being viewed. This applies only when the input file is a pipe, not an ordinary file.

-b The -b*n* option tells *less* to use a non-standard buffer size. There are two standard (default) buffer sizes, one is used when a file is being read and the other when a pipe (standard input) is being read. The current defaults are 5 buffers for files and 12 for pipes. (Buffers are 1024 bytes.) The number *n* specifies a different number of buffers to use. The -b may be followed by "f", in which case only the file default is changed, or by "p" in which case only the pipe default is changed. Otherwise, both are changed.

-c Normally, when data is read by *less,* it is scanned to ensure that bit 7 (the high order bit) is turned off in each byte read, and to ensure that there are no null (zero) bytes in the data (null bytes are turned into "@" characters). If the data is known to be "clean", the -c option will tell *less* to skip this checking, causing an imperceptible speed improvement. (However, if the data is not "clean", unpredicatable results may occur.)

+　　If a command line option begins with **+**, the remainder of that option is taken to be an initial command to *less*. For example, +G tells *less* to start at the end of the file rather than the beginning, and +/xyz tells it to start at the first occurence of "xyz" in the file. As a special case, +<number> acts like +<number>g; that is, it starts the display at the specified line number (however, see the caveat under the "g" command above). If the option starts with **++**, the initial command applies to every file being viewed, not just the first one. The + command described previously may also be used to set (or change) an initial command for every file.

ERRORS

When used on standard input (rather than a file), you can move backwards only a finite amount, corresponding to that portion of the file which is still buffered. The -b option may be used to expand the buffer space.

NAME

lex – generate programs for simple lexical tasks

SYNOPSIS

lex [**−rctvn**] [file] ...

DESCRIPTION

The *lex* command generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in **[abx−z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators *****, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except new-line. Parentheses for grouping and vertical bar for alternation are also supported. The notation *r*{*d*,*e*} in a rule indicates between *d* and *e* instances of regular expression *r*. It has higher precedence than |, but lower than *, ?, +, and concatenation. Thus **[a−zA−Z]+** matches a string of letters. The character ^ at the beginning of an expression permits a successful match only immediately after a new-line, and the character **$** at the end of an expression requires a trailing new-line. The character **/** in an expression indicates trailing context; only the part of the expression up to the slash is returned in *yytext*, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within **"** symbols or preceded by \.

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

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

EXAMPLE

```
D       [0–9]
%%
if      printf("IF statement\n");
[a–z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"      skipcommnts();
%%
 skipcommnts()
```

```
{
        for (;;)
        {
                while (input() != '*')
                        ;
                if (input() != '/')
                        unput(yytext[yyleng-1]);
            }   else {
                        return;
        }
}
```

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

The flags must appear before any files. The flag **−r** indicates RATFOR actions, **−c** indicates C actions and is the default, **−t** causes the **lex.yy.c** program to be written instead to standard output, **−v** provides a one-line summary of statistics, **−n** will not print out the **−v** summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n*	number of positions is *n* (default 2500)
%n *n*	number of states is *n* (500)
%e *n*	number of parse tree nodes is *n* (1000)
%a *n*	number of transitions is *n* (2000)
%k *n*	number of packed character classes is *n* (1000)
%o *n*	size of output array is *n* (3000)

The use of one or more of the above automatically implies the **−v** option, unless the **−n** option is used.

SEE ALSO

yacc(1).
Programmer's Guide.

ERRORS

The **−r** option is not yet fully operational.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1).
read(2) in the *Programmer's Reference Manual*.

NAME

lint – a C program checker

SYNOPSIS

lint [option] ... file ...

DESCRIPTION

Lint attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **−c** or the **−o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

Lint will take all the **.c**,**.ln**, and **llib-l***x***.ln** (specified by **−l***x*) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **−p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **−c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **−c** option is used, the **.ln** and the **llib-l***x***.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

−a Suppress complaints about assignments of long values to variables that are not long.

−b Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

−h Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

−u Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

−v Suppress complaints about unused arguments in functions.

−x Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

−l*x* Include additional lint library **llib-l***x***.ln**. For example, you can include a lint version of the Math Library **llib-lm.ln** by inserting **−lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

−n Do not check compatibility against either the standard or the portable lint library.

−p Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

−c Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lint*'s first pass only, and are not checked for inter-function compatibility.

−o lib Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **−c** option nullifies any use of the **−o** option. The lint library produced is the input that is given to *lint*'s second pass. The **−o** option simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use of the **−x** option is suggested. The **−v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The **−D**, **−U**, and **−I** options of *cpp*(1) and the **−g** and **−O** options of *cc*(1) are also recognized as separate arguments. The **−g** and **−O** options are ignored, but, by recognizing these options, *lint*'s behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
: at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS*n**/
: suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
: turns on the **−v** option for the next function.

/*LINTLIBRARY*/
: at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the **−v** and **−x** options.

Lint produces its first output on a per-source-file basis. Complaints regarding included files are collected and printed after all source files have been processed. Finally, if the **−c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.

The behavior of the **−c** and the **−o** options allows for incremental use of *lint* on a set of C source files. Generally, one invokes *lint* once for each source file with the **−c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **−c** option), listing all the **.ln** files with the needed **−l***x* options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

FILES

/usr/lib	the directory where the lint libraries specified by the **−l***x* option must exist
/usr/lib/lint[12]	first and second passes
/usr/lib/llib-lc.ln	declarations for C Library functions (binary format; source is in **/usr/lib/llib-lc**)
/usr/lib/llib-port.ln	declarations for portable functions (binary format; source is in **/usr/lib/llib-port**)

/usr/lib/llib-lm.ln	declarations for Math Library functions (binary format; source is in **/usr/lib/llib-lm**)
/usr/tmp/*lint*	temporaries

SEE ALSO

cc(1), cpp(1), make(1).

BUGS

exit(2), *longjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

list – produce C source listing from a common object file

SYNOPSIS

list [**−V**] [**−h**] [**−F** function] source-file . . . [object-file]

DESCRIPTION

The *list* command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, *list* will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **a.out**, will be used.

Line numbers will be printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace (**{**) that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by *list* and may be given in any order:

−V	Print, on standard error, the version number of the *list* command executing.
−h	Suppress heading output.
−F*function*	List only the named function. The **−F** option may be specified multiple times on the command line.

SEE ALSO

as(1), cc(1), ld(1).

CAVEATS

Object files given to *list* must have been compiled with the **−g** option of *cc*(1).

Since *list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

DIAGNOSTICS

list will produce the error message "list: name: cannot open" if *name* cannot be read. If the source file names do not end in **.c** , the message is "list: name: invalid C source name". An invalid object file will cause the message "list: name: bad magic" to be produced. If some or all of the symbolic debugging information is missing, one of the following messages will be printed: "list: name: symbols have been stripped, cannot proceed", "list: name: cannot read line numbers", and "list: name: not in symbol table". The following messages are produced when *list* has become confused by **#ifdef's** in the source file: "list: name: cannot find function in symbol table", "list: name: out of sync: too many }", and "list: name: unexpected end-of-file". The error message "list: name: missing or inappropriate line numbers" means that either symbol debugging information is missing, or *list* has been confused by C preprocessor statements.

NAME

locate – identify a UNIX system command using keywords

SYNOPSIS

[**help**] **locate**
[**help**] **locate** [keyword1 [keyword2] ...]

DESCRIPTION

The *locate* command is part of the UNIX system Help Facility, and provides on-line assistance with identifying UNIX system commands.

Without arguments, the initial *locate* screen is displayed from which the user may enter keywords functionally related to the action of the desired UNIX system commands they wish to have identified. A user may enter keywords and receive a list of UNIX system commands whose functional attributes match those in the keyword list, or may exit to the shell by typing q (for "quit"). For example, if you wish to print the contents of a file, enter the keywords "print" and "file". The *locate* command would then print the names of all commands related to these keywords.

Keywords may also be entered directly from the shell, as shown above. In this case, the initial screen is not displayed, and the resulting command list is printed.

More detailed information on a command in the list produced by *locate* can be obtained by accessing the *usage* module of the UNIX system Help Facility. Access is made by entering the appropriate menu choice after the command list is displayed.

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see *profile*(4)): "export SCROLL ; SCROLL=no". If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), sh(1), starter(1), usage(1).
term(5) in the *Programmer's Reference Manual*.

WARNINGS

If the shell variable **TERM** (see *sh*(1)) is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

login – sign on

SYNOPSIS

login [name [env-var . . .]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an "end-of-file." (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure **/etc/profile** is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and for *sh* (1) the file **.profile** in the working directory is executed, if it exists. These specifications are found in the **/etc/passwd** file entry for the user. The name of the command interpreter is **–** followed by the last component of the interpreter's path name (i.e., **–sh**). If this field in the password file is empty, then the default command interpreter, **/bin/sh** is used. For information about other command interpreters (for example, *csh* (1)), see the appropriate manual pages. If this field is "*", then the named directory becomes the root directory, the starting point for path searches for path names beginning with a **/**. At that point *login* is re-executed at the new level which must have its own root structure, including **/etc/login** and **/etc/passwd**.

The basic *environment* is initialized to:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*
TERM=*your-terminal-type*
LOGNAME=*your-login-name*
USER=*your-login-name*

The environment may be expanded or modified by supplying additional arguments to *login,* either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy.* Arguments without an equal sign are placed in the environment as

L*n*=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	mailbox for user *your-name*
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile (for sh)

SEE ALSO

mail(1), newgrp(1), sh(1), su(1M).
passwd(4), profile(4), environ(5) in the *Programmer's Reference Manual*.

DIAGNOSTICS

login incorrect if the user name or the password cannot be matched.
No shell, cannot open password file, or *no directory*: consult a UNIX system programming counselor.
No utmp entry. You must exec "login" from the lowest level "sh" if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname prints the login name associated with the terminal, regardless of the id of the userid associated with the current process.

The command *id(1M)* can be used to get information about the current userid and groupid.

SEE ALSO

id(1M), login(1), cuserid(3S).

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive *files* [see *ar*(1)]. The standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format [see *ar*(4)] and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files.

```
ar −cr library `lorder *.o | tsort`
```

FILES

TMPDIR/*symref	temporary files
TMPDIR/*symdef	temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the environment variable **TMPDIR** [see **tempnam**() in **tmpnam**(3S)].

SEE ALSO

ar(1), ld(1), tsort(1), ar(4).

CAVEAT

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be **.o**.

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**−c**] [**−d**dest] [**−m**] [**−n**number] [**−o**option] [**−s**] [**−t**title] [**−w**] files
cancel [ids] [printers]

DESCRIPTION

lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name **−** stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

−c	Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **−c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **−c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.
−d*dest*	Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).
−m	Send mail (see *mail(1))* after the files have been printed. By default, no mail is sent upon normal completion of the print request.
−n*number*	Print *number* copies (default of 1) of the output.
−o*option*	Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what is valid for *options*, see **Models** in *lpadmin*(1M).
−s	Suppress messages from *lp*(1) such as "request id is ...".
−t*title*	Print *title* on the banner page of the output.
−w	Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lpstat(1), mail(1).
accept(1M), lpadmin(1M), lpsched(1M) in the *System Administrator's Reference Manual.*

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

lpstat prints information about the current status of the LP spooling system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *lpstat* prints the status of such requests. *Options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
-u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat -o
```

prints the status of all output requests.

−a[*list*] Print acceptance status (with respect to *lp*) of destinations for requests. *List* is a list of intermixed printer names and class names.

−c[*list*] Print class names and their members. *List* is a list of class names.

−d Print the system default destination for *lp*.

−o[*list*] Print the status of output requests. *List* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*] Print the status of printers. *List* is a list of printer names.

−r Print the status of the LP request scheduler

−s Print a status summary, including the system default destination, a list of class names and their members, and a list of printers and their associated devices.

−t Print all status information.

−u[*list*] Print status of output requests for users. *List* is a list of login names.

−v[*list*] Print the names of printers and the path names of the devices associated with them. *List* is a list of printer names.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1).

NAME

ls – list contents of directory

SYNOPSIS

ls [**−RadCLHxmlnogrtucpFbqisf**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **−C** and **−x** options enable multi-column formats, and the **−m** option enables stream output format. In order to determine output formats for the **−C**, **−x**, and **−m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The *ls* command has the following options:

−R	Recursively list subdirectories encountered.
−L	If argument is a symbolic link, list the file or directory the link references rather than the link itself.
−H	If the file is a symbolic link, list the file itself.
−a	List all entries, including those that begin with a dot (.), which are normally not listed.
−d	If an argument is a directory, list only its name (not its contents); often used with **−l** to get the status of a directory.
−C	Multi-column output with entries sorted down the columns.
−x	Multi-column output with entries sorted across rather than down the page.
−m	Stream output format; files are listed across the page, separated by commas.
−l	List in long format, giving mode, number of hard links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
−n	The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.
−o	The same as **−l**, except that the group is not printed.
−g	The same as **−l**, except that the owner is not printed.
−r	Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
−t	Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **−n** and **−c**.)
−u	Use time of last access instead of last modification for sorting (with the **−t** option) or printing (with the **−l** option).

−c	Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).
−p	Put a slash (**/**) after each filename if that file is a directory.
−F	Put a slash (**/**) after each filename if that file is a directory and put an asterisk (*****) after each filename if that file is executable.
−b	Force printing of non-graphic characters to be in the octal **\ddd** notation.
−q	Force printing of non-graphic characters in file names as the character (?).
−i	For each file, print the i-number in the first column of the report.
−s	Give size in blocks, including indirect blocks, for each entry.
−f	Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

The mode printed under the **−l** option consists of ten characters. The first character may be one of the following:

d	the entry is a directory;
b	the entry is a block special file;
c	the entry is a character special file;
p	the entry is a fifo (a.k.a. "named pipe") special file;
s	the entry is a UNIX domain socket
−	the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

ls −l (the long list) prints its output as follows:

```
−rwxrwxrwx  1 smith  dev    10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, you see that the current directory holds one file, named "part2." Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of hard links to file "part2." Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (**x**) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.
The permissions are indicated as follows:

r	the file is readable
w	the file is writable
x	the file is executable
–	the indicated permission is *not* granted
l	mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
s	the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
S	undefined bit-state (the set-user-ID bit is on and the user execution bit if off)
t	the 1000 (octal) bit, or sticky bit, is on (see **chmod(1)**), and execution is on
T	the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than **x** or –. **s** also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

The first set of examples refers to permissions:

```
–rwxr––r––
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

```
–rwsr–xr–x
```

The second example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

```
–rw–rwl–––
```

This example describes a file that is readable and writable only by the user and the group and can be locked during access.

```
ls -a
```

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

```
ls -aisn
```

This command will provide you with quite a bit of information including **all** files, including non-printing ones (**a**), the **i**-number–the memory address of the i-node associated with the file–printed in the left-hand column (**i**); the **s**ize (in blocks) of the files, printed in the column to the right of the i-numbers (**s**); finally, the report is displayed in the **n**umeric version of the long list, printing the **UID** (instead of user name) and **GID** (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	user IDs for **ls −l** and **ls −o**
/etc/group	group IDs for **ls −l** and **ls −g**
/usr/lib/terminfo/?/*	terminal information database

SEE ALSO

chmod(1), find(1).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls −l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

ERRORS

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 – macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

The *m4* command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is **–**, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

–e	Operate interactively. Interrupts are ignored and the output is unbuffered.
–s	Enable line sync output for the C preprocessor (#line . . .)
–B*int*	Change the size of the push-back and argument collection buffers from the default of 4,096.
–H*int*	Change the size of the symbol table hash array from the default of 199. The size should be prime.
–S*int*	Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
–T*int*	Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

–D*name*[**=***val*]
Defines *name* to *val* or to null in *val*'s absence.

–U*name*
undefines *name*.

Macro calls have the form:

name(arg1,arg2, . . ., argn)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define	the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $*n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string;

$# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).

undefine	removes the definition of the macro named in its argument.
defn	returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
pushdef	like *define*, but saves any previous definition.
popdef	removes current definition of its argument(s), exposing the previous one, if any.
ifdef	if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX system versions of *m4*.
shift	returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
changequote	change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., \`').
changecom	change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
divert	*m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
undivert	causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
divnum	returns the value of the current output stream.
dnl	reads and discards characters up to and including the next new-line.
ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	returns the value of its argument decremented by 1.
eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include **+**, **−**, *, **/**, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the

	result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to *include*, except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX system command given in the first argument. No value is returned.
sysval	is the return code from the last call to *syscmd*.
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup( )´)
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.
traceon	with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.
traceoff	turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).
The m4 Macro Processor in the *Support Tools Guide.*

NAME

machid: mips, pdp11, u3b, u3b2, u3b5, vax – get processor type truth value

SYNOPSIS

mips

pdp11

u3b

u3b2

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

mips	True if you are on a Mips computer.
pdp11	True if you are on a PDP-11/45 or PDP-11/70.
u3b	True if you are on a 3B20 computer.
u3b2	True if you are on a 3B2 computer.
u3b5	True if you are on a 3B5 computer. **vax** True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles (see *make*(1)) and shell procedures (see *sh*(1)) to increase portability.

To obtain more specific information, use *uname*(1)).

SEE ALSO

sh(1), test(1), true(1), uname(1).
make(1) in the *Programmer's Reference Manual*.

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**−oswtd**] persons

rmail [**−oswt**] persons

Reading mail:

mail [**−ehpqrl**] [**−f** file] [**−F** persons]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

−o	suppresses the address optimization facility.
−s	suppresses the addition of a <new-line> at the top of the letter being sent. See WARNINGS below.
−w	causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.
−t	causes a **To:** line to be added to the letter, showing the intended recipients.
−d	causes mail to be delivered without going through the *sendmail* program.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the /-F option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line (unless the −s argument was used).

Messages are delivered via the program *sendmail(1M)* if and only if the file */usr/lib/sendmail.ok* exists. Otherwise, addresses containing the character ! or @ are delivered by *uucp(1C)* and other messages are delivered locally.

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line arguments that follow affect READING mail:

−e	causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
−l	causes messages to be printed when there is a lock file and retries are being attempted. Without this option, retries are done silently, resulting in up to a 5 minute wait with no indication.

−h causes a window of headers to be displayed rather than the latest message. The display is followed by the '?' prompt.

−p causes all messages to be printed without prompting for disposition.

−q causes *mail* to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.

−r causes messages to be printed in first-in, first-out order.

−f*file* causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

−F*persons*
entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

mail, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

<new-line>, **+**, or **n**	Go on to next message.
d, or **dp**	Delete message and go on to next message.
d #	Delete message number #. Do not go on to next message.
dq	Delete message and quit *mail*.
h	Display a window of headers around current message.
h #	Display header of message number #.
h a	Display headers of ALL messages in the user's *mailfile*.
h d	Display headers of messages scheduled for deletion.
p	Print current message again.
−	Print previous message.
a	Print message that arrived during the *mail* session.
#	Print message number #.
r [*users*]	Reply to the sender, and other *user(s)*, then delete the message.
s [*files*]	Save message in the named *files* (**mbox** is default).
y	Same as save.
u [#]	Undelete message number # (default is last read).
w [*files*]	Save message, without its top-most header, in the named *files* (**mbox** is default).
m [*persons*]	Mail the message to the named *persons*.
q, or **ctl-d**	Put undeleted mail back in the *mailfile* and quit *mail*.
x	Put all mail back in the *mailfile* unchanged and exit *mail*.
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

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*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **−F** option.

To forward all of one's mail to systema!user enter:

```
mail -Fsystema!user
```

To forward to more than one user enter:

```
mail -F"user1,systema!user2,systema!systemb!user3"
```

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the **−F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding enter:

```
mail −F ""
```

The pair of double quotes is mandatory to set a NULL argument for the −F option.

In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/*user*	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail
$MAIL	variable containing path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), sendmail(1M), write(1).
User's Guide.
System Administrator's Guide.

WARNING

The "Forward to person" feature may result in a loop, if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb*. The symptom is a message saying "unbounded...saved mail in dead.letter."

The **−s** option should be used with caution. It allows the text of a message to be interpreted as part of the postmark of the letter, possibly causing confusion to other *mail* programs. To allow compatibility with *mailx*(1), if the first line of the message is "Subject:...", the addition of a <newline> is suppressed whether or not the **−s** option is used.

ERRORS

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

mailq – print sendmail mail queue

SYNOPSIS

mailq [**−v**]

DESCRIPTION

mailq prints the contents of the mail queue used by *sendmail(1M)*. The **−v** provides more information.

SEE ALSO

aliases(4), sendmail(1M)

NAME

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **−f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (−) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

−e	Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.
−f [*filename*]	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
−F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
−h *number*	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See **addsopt** under ENVIRONMENT VARIABLES)
−H	Print header summary only.
−i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
−n	Do not initialize from the system default *mailx.rc* file.
−N	Do not print initial header summary.
−r *address*	Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under ENVIRONMENT VARIABLES)
−s *subject*	Set the Subject header field to *subject*.
−u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.
−U	Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable. (See **addsopt** under ENVIRONMENT VARIABLES)

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see

COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (˜) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the **alias** command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n−m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:*c*	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, **C**opy, **e**dit, **fo**llowup, **F**ollowup, **ho**ld, **m**ail, **pre**serve, **r**eply, **R**eply, **sh**ell, and **v**isual. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional, and must be constructed locally.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*
　Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

*comment*
　Null command (comment). This may be useful in *.mailrc* files.

=
　Print the current message number.

?
　Prints a summary of commands.

alias *alias name ...*
group *alias name ...*
　Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates *name ...*
　Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alt**ernates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [*directory*]
chdir [*directory*]
　Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
　Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.

Copy [*msglist*]
　Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.

delete [*msglist*]
　Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [*header-field* ...]
ignore [*header-field* ...]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

dp [*msglist*]
dt [*msglist*]

Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a **d**elete command followed by a **p**rint command.

echo *string* ...

Echo the given strings (like *echo*(1)).

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed*(1).

exit
xit

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **q**uit).

file [*filename*]
folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

% the current *mailbox*.
%**user**
the *mailbox* for **user**.
# the previous file.
& the current *mbox*.

Default file is the current *mailbox*.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the **F**ollowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **fo**llowup, **S**ave, and **C**opy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]
Prints the header summary for the specified messages.

group *alias name ...*
alias *alias name ...*
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]
Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the **z** command.

help
Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Holds the specified messages in the *mailbox*.

if *s* | *r*
*mail-command*s
else
*mail-command*s
endif
Conditional execution, where *s* will execute following *mail-command*s, up to an **el**se or **en**dif, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore *header-field ...*
discard *header-field ...*
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **P**rint and **T**ype commands override this command.

list
Prints all commands available. No explanation is given.

mail *name ...*
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **ex**it and **q**uit commands.

next [*message*]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]

Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

print [*msglist*]
type [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit

Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]
respond [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the **C**opy, **fo**llowup, and **F**ollowup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]
save [*msglist*] *filename*
: Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the **exit** and **quit** commands).

set
set *name*
set *name=string*
set *name=number*
: Define a variable called *name*. The variable may be given a null, string, or numeric value. **Set** by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell
: Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [*msglist*]
: Print the size in characters of the specified messages.

source *filename*
: Read commands from the given file and return to command mode.

top [*msglist*]
: Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [*msglist*]
: Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See **ex**it and **q**uit.

Type [*msglist*]
Print [*msglist*]
: Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore command.

type [*msglist*]
print [*msglist*]
: Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [*msglist*]
: Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset *name* ...
: Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version
: Prints the current version and release date.

visual [*msglist*]
: Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [*msglist*] *filename*
: Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the **save** command.

xit
exit
: Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

z[+ | −]
: Scroll the header display forward or backward one screen−full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (˜). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

˜**!** *shell-command*
: Escape to the shell.

˜**.**
: Simulate end of file (terminate message input).

˜**:** *mail-command*
˜_ *mail-command*
: Perform the command-level request. Valid only when sending a message while reading mail.

˜**?**
: Print a summary of tilde escapes.

˜**A**
: Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

˜**a**
: Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

˜**b** *name ...*
: Add the *names* to the blind carbon copy (Bcc) list.

˜**c** *name ...*
: Add the *names* to the carbon copy (Cc) list.

~d

Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e

Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [*msglist*]
Forward the specified messages. The messages are inserted into the message, without alteration.

~h

Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i *string*
Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.'

~m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p

Print the message being entered.

~q

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r *filename*
~~< *filename*
~~< *!shell-command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s *string ...*
Set the subject line to *string*.

~t *name ...*
Add the given *names* to the To list.

~v

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~**w** *filename*
    Write the partial message onto the given file, without the header.

~**x**
    Exit as with ~**q** except the message is not saved in *dead.letter*.

~| *shell-command*
    Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*
    The user's base of operations.

MAILRC=*filename*
    The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **un**set command may be used to erase variables.

addsopt
    Enabled by default. If */bin/mail* is not being used as the deliverer, **noaddsopt** should be specified. (See WARNINGS below)

allnet
    All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alt**ernates command and the "metoo" variable.

append
    Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc
    Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub
    Prompt for subject if it is not specified on the command line with the **−s** option. Enabled by default.

autoprint
    Enable automatic printing of messages after **d**elete and **u**ndelete commands. Default is **noautoprint**.

bang
    Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=*shell-command*
Set the default command for the **pi**pe command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **−U** command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot
Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*
The command to run when the **e**dit or **~e** command is used. Default is *ed*(1).

escape=*c*
Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*
The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header
Enable printing of the header summary when entering *mailx*. Enabled by default.

hold
Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore
Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (.) on a

line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep
: When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
: Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
: The name of the file to save messages which have been read. The **x**it command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
: If your login appears as a recipient, do not delete it from the list. Default is **nometoo.**

LISTER=*shell-command*
: The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop
: When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
: Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the **S**ave, **C**opy, **fo**llowup, and **F**ollowup commands.

page
: Used with the **pi**pe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
: The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
: Set the *command mode* prompt to *string*. Default is "? ".

quiet
: Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screen–full of headers for the **headers** command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is *mail*(1).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is *sh*(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also **~i** (TILDE ESCAPES)).

Sign=*string*
The variable inserted into the text of a message when the **~A** command is given. No default (see also **~i** (TILDE ESCAPES)).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

ls(1), mail(1), pg(1).

WARNINGS

The **−h**, **−r** and **−U** options can be used only if *mailx* is built with a delivery program other than */bin/mail*.

ERRORS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**−f** makefile] [**−p**] [**−i**] [**−k**] [**−s**] [**−r**] [**−n**] [**−b**] [**−e**] [**−u**] [**−t**] [**−q**] [names]

DESCRIPTION

The *make* command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

−f *makefile*	Description file name. *makefile* is assumed to be the name of a description file.
−p	Print out the complete set of macro definitions and target descriptions.
−i	Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.
−k	Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
−s	Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.
−r	Do not use the built-in rules.
−n	No execute mode. Print commands, but do not execute them. Even lines beginning with an **@** are printed.
−b	Compatibility mode for old makefiles.
−e	Environment variables override assignments within makefiles.
−u	Force an unconditional update.
−t	Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
−q	Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
.DEFAULT	If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.
.PRECIOUS	Dependents of this target will not be removed when quit or interrupt are hit.
.SILENT	Same effect as the **−s** option.
.IGNORE	Same effect as the **−i** option.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **−f** option is present, **makefile**, **Makefile**, and the Source Code Control System(SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is **−**, the standard input is taken. More than one **−** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target (unless the **−u** option is used to force an unconditional update). All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a **:**, then a (possibly null) list of prerequisite files or dependencies. Text following a **;** and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or **#** begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
        echo a\
        b
```

will produce

```
        ab
```

exactly the same as the shell would.

Sharp (#) and new-line surround comments.

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
        pgm: a.o b.o
                cc a.o b.o −o pgm
        a.o: incl.h a.c
                cc −c a.c
        b.o: incl.h b.c
                cc −c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell *make* should use to execute commands. The default is */bin/sh*. The first one or two characters in a command can be the following: **−**, **@**, **−@**, or **@−**. If **@** is present, printing of the command is suppressed. If **−** is present, *make* ignores an error. A line is printed when it is executed unless the **−s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **−n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **−t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **−i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **−**. the error is ignored. If the **−k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **−b** option allows old makefiles (those written for the old version of *make*) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **−e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **−f** and **−p**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **−n** option is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make −n** recursively on a whole software system to see what would have been executed. This is because the **−n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string *include* appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1* = *string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c  .c~  .f  .f~  .sh  .sh~
.c.o  .c.a  .c~.o  .c~.c  .c~.a
.f.o  .f.a  .f~.o  .f~.f  .f~.a
```

```
        .h~.h  .s.o  .s~.o  .s~.s  .s~.a  .sh~.sh
        .l.o  .l.c  .l~.o  .l~.l  .l~.c
        .y.o  .y.c  .y~.o  .y~.y  .y~.c
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
        make -fp - 2>/dev/null </dev/null
```

A tilde in the above rules refers to an SCCS file [see *sccsfile*(4)]. Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x***.c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
        pgm: a.o b.o
                cc a.o b.o -o pgm
        a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form **.*XX*.a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile
/bin/sh

SEE ALSO

cc(1), lex(1), yacc(1), printf(3S), sccsfile(4).
cd(1), sh(1) in the *User's Reference Manual.*

NOTES

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

ERRORS

File names with the characters **= : @** will not work. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** does not work. Named pipes are not handled well.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, **.**, **/**, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO

ed(1), crypt(1), vi(1).
passwd(4) in the *Programmer's Reference Manual.*

NAME

man - view online manual pages

SYNOPSIS

man [*−section*] [**−c**] [**−d**] [**−w**] [**−T***termtype*] [**−12**] *name...*

DESCRIPTION

man looks for manual pages in a base directory, which defaults to */usr/man,* and prints them out using a pager.

By default, the manual page is filtered by *ul(1)* and *more(1)* is used as the pager. This may be overridden by the PAGER environment variable, which may in turn be overridden by the MANPAGER environment variable. If the pager name ends with the word "more", the options **−f** and **−s** are added. If it ends with the word "less", the option **−s** is added, and **ul** processing is turned off.

If there are multiple manual pages that match a given *name,* all matching files are printed, but each is only printed once.

If more than one manual page is printed, the message

[Hit return for next manual page, q to quit]

is printed. At this point, the user may hit return to see the next manual page, or may type a word beginning with a 'q' or 'Q' to quit.

OPTIONS

−section	Look for the manual page in the specified section number (1-7) only.
−c	Run each manual page through the *col(1)* postprocessor. Most manual pages have been run through this processor before being placed on the system, so this option isn't generally required.
−d	Search for manual entries in the current directory instead of */usr/man.*
−w	Print the names of the manual page files found. These will be relative to the base search directory (usually */usr/man).*
−T*termtype*	Set the terminal type to the named type. Special terminals (such as the Tektronix 4014) cause the special greek character postprocessor (if available) to be used.
−12	Causes the special 12 lines/inch version of the given terminal type to be used.

SEE ALSO

col(1), less(1), more(1), ul(1).

NAME

merge – three-way file merge

SYNOPSIS

merge [**-p**] file1 file2 file3

DESCRIPTION

merge incorporates all changes that lead from *file2* to *file3* into *file1*. The result goes to std. output if **-p** is present, into *file1* otherwise. *merge* is useful for combining separate changes to an original. Suppose *file2* is the original, and both *file1* and *file3* are modifications of *file2*. Then *merge* combines both changes.

An overlap occurs if both *file1* and *file3* have changes in a common segment of lines. *merge* prints how many overlaps occurred, and includes both alternatives in the result. The alternatives are delimited as follows:

```
<<<<<<< file1
lines in file1
=======
lines in file3
>>>>>>> file3
```

If there are overlaps, the user should edit the result and delete one of the alternatives.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.4 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

diff3 (1), diff (1), rcsmerge (1), co (1).

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*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir – make directories

SYNOPSIS

mkdir [**−m** mode] [**−p**] dirname ...

DESCRIPTION

mkdir creates the named directories in mode 777 (possibly altered by *umask*(1)).

Standard entries in a directory (e.g., the files **.**, for the directory itself, and **..**, for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

−m	This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).
−p	With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

sh(1), rm(1), umask(1).
intro(2), mkdir(2) in the *Programmer's Reference Manual*.

DIAGNOSTICS

mkdir returns exit code 0 if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns non-zero.

NAME

mkshlib – create a shared library

SYNOPSIS

mkshlib −s specfil [**−t** target] [**−h** host] [**−n**] [**−q**] [**−v**]

DESCRIPTION

The *mkshlib* command builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs that link with a shared library will share the library code during execution. Programs that link with a non-shared library will get their own copies of each library routine used.

The host shared library is an archive that is used to link-edit user programs with the shared library [see *ar*(4)]. A host shared library can be treated exactly like a non-shared library and should be included on compiler driver (*cc*(1), etc.) command lines in the usual way. Further, all operations that can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is attached to the user's process during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to *mkshlib* consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The *mkshlib* command invokes other tools, such as the archiver, *ar*(1), the assembler, *as*(1), and the link editor, *ld*(1). Tools are invoked through the use of *execvp*(3), which searches directories in the user's PATH. Also, suffixes to *mkshlib* are parsed in the same manner as suffixes to the compiler drivers, and invoked tools are given the suffix, where appropriate. For example, mkshlib*1.0* will invoke ld*1.10*.

The following command line options are recognized by *mkshlib*:

−s specfil	Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library. Its contents include the branch table specifications for the target, the pathname in which the target should be installed, the start addresses of text and data for the target, the initialization specifications for the host, and the list of object files to be included in the shared library (see details below).
−t target	Specifies the name, *target*, of the target shared library produced on the host machine. When *target* is moved to the target machine, it should be installed at the location given in the specification file (see the **#target** directive below). If the **−n** option is used, a new target shared library will not be generated.
−h host	Specifies the name of the host shared library, *host*. If this options is not given, the host shared library will not be produced.
−n	Do not generate a new target shared library. This option is useful when producing only a new host shared library. The **−t** option must still be supplied since a version of the target shared library is needed to build the host shared library.
−q	Quiet warning messages. This option is useful when warning messages are expected, but not desired.
−v	Set the verbose option. This option prints the command lines it executes as

in the compiler drivers.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the following directives:

#address segname address
: Specifies the start address, *address*, of the segment *segname* for the target. This directive is used to specify the start addresses of the text and data segments. Since the headers part of the text segment of target shared libraries they are put on there own page. The real text starts on the next page from where the text segment is specified.

#target pathname
: Specifies the absolute pathname, *pathname*, of the target shared library on the target machine. This pathname is copied to **a.out** files and is the location where the operating system will look for the shared library when executing a file that uses it.

#branch
: Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

funcname < white space > position

where *funcname* is the name of the symbol given a branch table entry and *position* specifies the position of *funcname's* branch table entry. *Position* may be a single integer integer or a range of integers of the form *position1-position2*. Each *position* must be greater than or equal to one, the same position cannot be specified more than once, and every position from one to the highest given position must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library.

Finally, only functions should be given branch table entries, and those functions must be external.

This directive can be specified only once per shared library specification file.

#objects
: Specifies the names of the object files constituting the target shared library. The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each filename followed by white space. This list is also used to determine the input object files for the host shared library.

This directive can be specified only once per shared library specification file.

#init object
: Specifies that the object file, *object,* requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

pimport < white space > import

Pimport is a pointer to the associated imported symbol, *import,* and must be defined in the current specified object file, *object*. The initialization code generated for each such line is of the form:

pimport = &import;

where *pimport* is the absolute address of *import*.

All initializations for a particular object file must be given at once and multiple specifications of the same object file are not allowed.

#ident string — Specifies a string, *string,* to be included in the .comment section of the target shared library. This directive can be specified only once per shared library specification file. This is ignored but allowed for compatibility.

— Specifies a comment. All information on a line following this directive is ignored.

All directives that may be followed by multi-line specifications are valid until the next directive of the end of the file.

FILES

*TEMPDIR/**	temporary files

TEMPDIR is usually /tmp, but can be redefined by setting the environment variable **TMPDIR** [see *tempnam*() in *tmpnam*(3S)].

SEE ALSO

ar(1), as(1), cc(1), ld(1), a.out(5), ar(5)

NOTES

The addresses of the text and data segments must meet the boundary requirements of the operating system. For UMIPS-V the segments must be on 2 megabyte boundaries.

Because of jump instructions on MIPS machines, all the text making up the program should be in the same 256 megabyte segment so that all the text can be reached by normal jumps. It is suggested that shared library text segments be allocated from the top of the first 256 megabyte segment (0x10000000) through lower addresses. User program's text segments would continued to be link at the bottom (0x00400000) which is the default. This is suggested so that maximum distance be obtained between user's text and shared library text.

The target shared library data segments are suggested to be allocated from where the normal default data segment is loaded (0x10000000) through higher addresses. This will result in the user having to load his data segment after the target shared library he uses with the highest data segment address. This suggestion will allow the maximum space for the *sbrk(2)* arena and the stack to grow without interference of target shared library segments.

NAME

mkstr – create an error message file by massaging C source

SYNOPSIS

mkstr [**–**] messagefile prefix file ...

DESCRIPTION

mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

mkstr will process each of the specified *files,* placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. 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 a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char    efilname[] =  "/usr/lib/pi_strings";
int     efil = -1;

error(a1, a2, a3, a4)
{
        char buf[256];

        if (efil < 0) {
                efil = open(efilname, 0);
                if (efil < 0) {
oops:
                        perror(efilname);
                        exit(1);
                }
        }
        if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
                goto oops;
        printf(buf, a2, a3, a4);
}
```

The optional **–** causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

SEE ALSO

lseek(2), xstr(1)

NAME

more, page − file perusal filter for crt viewing

SYNOPSIS

more [**−cdflrsu**] [*−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 "Press space to continue, ′q′ to quit." at the end of each screenful, and will respond to subsequent illegal user input by printing "Press ′h′ for instructions." instead of ringing the bell. 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 positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.
−l	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.
−r	Do not delete carriage returns from the input. Normally, *more* will delete carriage returns in order to properly view files with lines ending in carriage returns and linefeeds, such as those produced by the *script(1)* command. Note that the **−r** option is implied when the standard output is not a terminal.
−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
i b	skip back *i* screenfuls and print a screenful of lines
i ^B	same as b
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
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

csh(1), man(1), script(1), sh(1), environ(5)

ERRORS

Skipping backwards is too slow on large files.

NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [**−f** *tapename*] [**−t** *tapename*] *command* [*count*]

DESCRIPTION

mt is used to give commands to a magnetic tape drive. If a tape name is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device */dev/rmt12*. Note that *tapename* must reference a raw (not block) tape device. By default *mt* performs the requested operation once. Operations may be performed multiple times by specifying *count*.

The available commands are listed below. Only as many characters as are required to uniquely identify a command need be specified.

eof, weof	Write *count* end-of-file marks at the current position on the tape.
fsf	Forward space *count* files.
fsr	Forward space *count* records.
bsf	Back space *count* files.
bsr	Back space *count* records.
rewind	Rewind the tape (*Count* is ignored).
offline, rewoffl	Rewind the tape and place the tape unit off-line (*Count* is ignored).
status	Print status information about the tape unit.
ret	Retension the tape (recommended for new tapes).
online	Place tape unit online (load tape).
append	space to end of recorded data to allow append to tape

mt returns a 0 exit status when the operation(s) were successful, 1 if the command was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt*	Raw magnetic tape interface

SEE ALSO

mtio(7), ioctl(2), environ(5)

NAME

multivol – handle multivolume files

SYNOPSIS

multivol −o [**−vtw**] [**−b** *blocksize*] [**−n** *count*] [**−l** *label*] [*device*]

multivol −i [**−vtw**] [**−b** *blocksize*] [**−n** *count*] [**−l** *label*] [*device*]

multivol −t [*device*]

DESCRIPTION

multivol allows for the convenient use of multiple volume files such as when backing up output from *cpio/tar* over several floppy disks.

Option **−o** reads the standard input file, breaking it into volumes on the output device. Option **−i** reads volumes from the device, concatenates them and writes the result to standard output.

The device to use is given by the *device* argument, or */dev/multivol* if none is specified. Normally this will be a character special file corresponding to a raw disk or tape unit. It does not have to be a random access device: *multivol* only accesses it sequentially. However, *multivol* may close and re-open a volume in order to rewind devices such as magnetic tape. Note that this device may not exist on your system. If you create a default device, you may wish to set up an entry in the *MKDEV(1M)* device database to keep the information available in case the device directory needs to be rebuilt.

The **−b** *blocksize* argument specifies the physical block size to be used. This number may end with **k** or **b** to specify multiplication by 1024 or 512 respectively. With a floppy disk it can usefully be made the size of one track. The default physical block size is 512 bytes. If the device or its driver does not allow successful writing all the way to the end of a physical volume, such as with magnetic tape, the **−n** *count* options should be used to specify the maximum number of physical blocks to be written on one volume.

The **−l** *label* option allows the user to supply a short string to be written onto the volume for identification.

Before reading or writing a new volume, *multivol* prompts on the controlling terminal for the user to insert the appropriate volume into the physical device and waits for a new-line to be typed. With **−i**, if the volume inserted is not a *multivol* volume, is not the next volume in sequence, or does not have the same date stamp as the previous volume, *multivol* verifies that you really want to read it.

multivol writes an ASCII header on each volume, and checksum info in each block hence the actual amount of data stored on each volume will be slightly less than its physical size. The **−t** option prints some of this header information on standard error. It includes:

- the date that the volume was written
- the sequence number of the volume
- any *label* string supplied at the time the volume was written

If you specify **−w** with **−i** or **−o** *multivol* assumes the **−t** option, and asks you to verify that each volume really is the required one.

The **−v** option tells *multivol* to write various other verbose information on standard error such as an indication of how many blocks it has read or written.

EXAMPLES

Tell me what you know about this volume:

```
        multivol -t
```

Backup a directory to the default device:

```
        tar cf – mydir | multivol –o
```

and retrieve it again:

```
        cd mydir
        multivol –i | tar xf –
```

Backup to tape all files changed since last time:

```
        touch /etc/multivoldate1
        find / –newer /etc/multivoldate2 –print | cpio –ov |
                multivol –o –l WEEKLY –b 20k –n 1000 /dev/rmt0
        mv /etc/multivoldate1 /etc/multivoldate2
```

Retrieve a file from that backup:

```
        cd /
        multivol –i /dev/rmt0 | cpio –idmv usr/myname/myfile
```

FILES

/dev/tty	where prompts are written and responses read
/dev/multivol	the default device (NOTE: may not be set up)

SEE ALSO

tar(1), cpio(1), MKDEV(1M)

DIAGNOSTICS

A message is written on standard error and *multivol* terminates in the event of

- incorrect usage
- not being able to open the device
- not being able to open /dev/tty
- a device I/O error

ERRORS

In the event of an I/O error you may have to start again with the first volume depending on the nature of the file and the program which produced it. In many cases it simply means the end of the volume has been reached, and no more can be read/written. If a block limit has been set *multivol* will indicate this condition.

When reading or writting, *multivol* attempts to read the volumes header to display what is being written over, or to determine the block size recorded in the header. Some raw devices will return an I/O error when the volume has not been written before, or has been previously written with a different block size. Hence the volume header cannot be read until the original block size is also specified. The first time a volume is written, *multivol* may signal an I/O error as it attempts to read the header, but will proceed to allow the volume to be written.

Volume labels may not contain white space and are limited to 14 characters.

Some tape device drivers cannot handle a read/write request while the tape is rewinding, for such drivers the **–w** switch is recommended in place of just **–t**

When using commands such as *cpio(1)* or *tar(1)*, do not use any of the blocking factor options of those commands. Instead, specify the blocking factor to *multivol* for the device.

NAME

netstat – show network status

SYNOPSIS

netstat [**−Aan**] [**−f** *address_family*] [*system*] [*core*]
netstat [**−himnrs**] [**−f** *address_family*] [*system*] [*core*]
netstat [**−n**] [**−I** *interface*] *interval* [*system*] [*core*]

DESCRIPTION

The *netstat* command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an *interval* specified, *netstat* will continuously display the information regarding packet traffic on the configured network interfaces.

The options have the following meaning:

−A	With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.
−a	With the default display, show the state of all sockets; normally sockets used by server processes are not shown.
−h	Show the state of the IMP host table.
−i	Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown).
−I *interface*	Show information only about this interface; used with an *interval* as described below.
−m	Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).
−n	Show network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.
−s	Show per-protocol statistics.
−r	Show the routing tables. When **−s** is also present, show routing statistics instead.
−f *address_family*	Limit statistics or address control block reports to those of the specified *address family*. The following address families are recognized: *inet*, for **AF_INET**, *ns*, for **AF_NS**, and *unix*, for **AF_UNIX**.

The arguments, *system* and *core* allow substitutes for the defaults "/unix" and "/dev/kmem". **AF_UNIX** is not supported by Silicon Graphics, Inc.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "host.port" or "network.port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases */etc/hosts* and */etc/networks*, respectively. If a symbolic name for an address is unknown, or if the **−n** option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to *inet*(3N). Unspecified, or "wildcard", addresses and ports appear as "*".

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G"), and whether the route was created dynamically by a redirect ("D"). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When *netstat* is invoked with an *interval* argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the −*I* option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

SEE ALSO

hosts(4), networks(4), protocols(4), services(4)

ERRORS

The notion of errors is ill-defined. Collisions mean something else for the IMP.

ORIGIN

4.3 BSD

NAME

newaliases – rebuild the data base for the mail aliases file

SYNOPSIS

newaliases

DESCRIPTION

newaliases rebuilds the random access data base for the mail aliases file */usr/lib/aliases*. It must be run each time */usr/lib/aliases* is changed in order for the change to take effect.

SEE ALSO

aliases(4), sendmail(1M)

NAME

newform – change the format of a text file

SYNOPSIS

newform [**−s**] [**−i**tabspec] [**−o**tabspec] [**−b**n] [**−e**n] [**−p**n] [**−a**n] [**−f**] [**−c**[char]] [**−l**n] [files]

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect. Except for **−s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "**−e**15 **−l**60" will yield results different from "**−l**60 **−e**15". Options are applied to all *files* on the command line.

−s
: Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a ∗ and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

```
newform -s -i -l -a -e file-name
```

−i*tabspec*
: Input tab specification: expands tabs to spaces, according to the tab specifications given. In addition, *tabspec* may be **−−**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** expects no tabs; if any are found, they are treated as **−1**.

−o*tabspec*
: Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **−i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **−8**. A *tabspec* of **−0** means that no spaces will be converted to tabs on output.

−b*n*
: Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **−l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **−b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

```
newform -l1 -b7 file-name
```

−e*n*
: Same as **−b***n* except that characters are truncated from the end of the line.

−p*n* Prefix *n* characters (see **−c***[k]*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

−a*n* Same as **−p***n* except characters are appended to the end of a line.

−f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **−o** option. If no **−o** option is specified, the line which is printed will contain the default specification of **−8**.

−c*[k]*
Change the prefix/append character to *k*. Default character for *k* is a space.

−l*n* Set the effective line length to *n* characters. If *n* is not entered, **−l** defaults to 72. The default line length without the **−l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **−i** to expand tabs to spaces).

The **−l1** must be used to set the effective line length shorter than any existing line in the file so that the **−b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...	*newform* was called with a bad option.
not −s format	There was no tab on one line.
can't open file	Self-explanatory.
internal line too long	A line exceeds 512 characters after being expanded in the internal work buffer.
tabspec in error	A tab specification is incorrectly formatted, or specified tab stops are not ascending.
tabspec indirection illegal	A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0 – normal execution
1 – for any error

SEE ALSO

csplit(1)
fspec(4) in the *Programmer's Reference Manual*.

ERRORS

newform normally only keeps track of physical characters; however, for the **−i** and **−o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **−i−−** or **−o−−**).

If the **−f** option is used, and the last **−o** option specified was **−o−−**, and was preceded by either a **−o−−** or a **−i−−**, the tab specification format line will be incorrect.

NAME

news – print news items

SYNOPSIS

news [**−a**] [**−n**] [**−s**] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

−a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

−n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

−s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5) in the *Programmer's Reference Manual*.

NAME

nice – run a command at low priority

SYNOPSIS

nice [**–**increment] command [arguments]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **−−10**.

SEE ALSO

nohup(1).
nice(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

nice returns the exit status of the subject command.

ERRORS

An *increment* larger than 19 is equivalent to 19.

The *csh* (1) has a built-in *nice* (1) command, which has a different syntax.

NAME

nl – line numbering filter

SYNOPSIS

nl [**−h***type*] [**−b***type*] [**−f***type*] [**−v***start#*] [**−i***incr*] [**−p**] [**−l***num*] [**−s***sep*] [**−w***width*] [**−n***format*] [**−d***delim*] [*file*]

DESCRIPTION

nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect. *nl* views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body). The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body. Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

−b*type*
Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

−h*type* Same as **−b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

a	number all lines
t	number lines with printable text only
n	no line numbering
p*string*	**number only lines that contain the regular expression specified in** *string.*

Default *type* **for logical page body is t** (text lines numbered).

−f*type* Same as **−b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

−v*start#* *start#* is the initial value used to number logical page lines. Default is **1**.

−i*incr* *Incr* is the increment value used to number logical page lines. Default is **1**.

−p Do not restart numbering at logical page delimiters.

−l*num* *num* is the number of blank lines to be considered as one. For example, **−l2** results in only the second adjacent blank being numbered (if the appropriate **−ha**, **−ba**, and/or **−fa** option is set). Default is **1**.

−s*sep* *sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

−w*width* *width* is the number of characters to be used for the line number.

Default *width* is **6**.

−n*format* — *format* is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

−d*xx* — The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl −v20 −i20 −d!+ file1
```

will number file1 starting at line number 20 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

NAME

nm – name list dump of MIPS object files

SYNOPSIS

nm [-abdefghnopruvxABTV] [file1 ... fileN]

DESCRIPTION

The **nm** command prints listings formats for the symbol and external sections of the symbol table. A *file* can be an object or an archive. If you do not specify a file, this command assumes *a.out*.

The **−A** and **−B** options specify AT&T System V style output or Berkeley (4.3 BSD) style output, respectively. The version of UNIX running at your site determines the default. **NOTE:** Some options can change the version-specific defaults. These options change the meaning of overloaded flags after -A or -B is specified.

A normal Berkeley system produces the *address* or *value* field followed by a *letter* showing what section the symbol or external is in and the *name* of the symbol or external.

These section letters describe the information that **nm** generates

N	nil storage class, compiler internal usage
T	external text
t	local text
D	external initialized data
d	local initialized data
B	external zeroed data
b	local zeroed data
A	external absolute
a	local absolute
U	external undefined
G	external small initialized data
g	local small initialized data
S	external small zeroed data
s	local small zeroed data
R	external read only
r	local read only
C	common
E	small common
V	external small undefined

The standard System V format and the **−a** specified Berkeley format provide an expanded listing with these columns:

Name	the symbol or external name
Value	the value field for the symbol or external, usually an address or interesting debugging information
Class	the symbol type
Type	the symbol's language declaration

Size	unused
Index	the symbol's index field
Section	the symbol's storage class

NOTE: Every effort was made to map the field's functionality into System V nomenclature.

The **nm** command accepts these options:

-a	prints debugging information, effectively turning Berkeley into System V format
-b	prints the value field in octal
-d	prints the value field in decimal (the System V default)
-e	prints external and statics only
-f	produces full output–**nm** still accepts this old option, but ignores it
-h	does not print headers
-n	for System V, sorts external symbols by name (default for Berkeley), and for Berkeley, sorts all symbols by value
-o	for System V, prints the value field in octal, and for Berkeley prepends the filename to each symbol–good for grepping through **nm** of libraries
-p	prints symbols as they are found in the file (the System V default)
-r	reverses the sense of a value or name sort
-u	prints only undefined symbols
-v	sorts external symbols by value
-x	prints value field in hexadecimal (Berkeley default)
-T	truncates long names, inserting a '*' as the last printed character
-V	prints version information on stderr

SEE ALSO

MIPS System Programmer Guide, *MIPS Languages Programmer Guide*

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup command [arguments]

DESCRIPTION

nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1),
signal(2) in the *Programmer's Reference Manual.*

WARNINGS

In the case of the following command

```
nohup command1; command2
```

nohup applies only to command1. The command

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

od – octal dump

SYNOPSIS

od [**−bcdosx**] [file] [[**+**]offset[**.**][**b**]]

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **−o** is default. The meanings of the format options are:

−b	Interpret bytes in octal.
−c	Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.
−d	Interpret words in unsigned decimal.
−o	Interpret words in octal.
−s	Interpret 16-bit words in signed decimal.
−x	Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

NAME

odump – dumps selected parts of an object file

SYNOPSIS

odump [-a -c -f -g -h -i -l -o -r -s -t -F -P -R -z -L] file1 ... fileN

DESCRIPTION

The **odump** command dumps selected parts of each object *file*.

This command works for object files and archives of object files. It accepts one or more of these options:

−a	Dumps the archive header for each member of the specified archive file.
−f	Dumps each file header.
−g	Dumps the global symbols from the symbol table of a MIPS archive.
−o	Dumps each optional header.
−h	Dumps section headers.
−i	Dumps the symbolic information header.
−s	Dumps section contents.
−r	Dumps relocation information.
−l	Dumps line number information.
−t	Dumps symbol table entries.
−z*name*	Dumps line number entries for the specified function *name*.
−c	Dumps the string table.
−L	Interpret and print the contents of the *.lib* sections.
−F	Dumps the file descriptor table.
−P	Dumps the procedure descriptor table.
−R	Dumps the relative file index table.

The **odump** command accepts these modifiers with the options:

−d *number*	Dumps the section number or a range of sections starting at *number* and ending either at the last section number or the *number* you specify with **+d.**
+d *number*	Dumps sections in the range beginning with the first section or beginning with the section you specify with **−d**.
−n *name*	Dumps information only about the specified *name*. This modifier works with **−h**, **−s**, **−r**, **−l**, and **−t.**
−p	Does not print headers
−t *index*	Dumps only the indexed symbol table entry. You can also specify a range of symbol table entries by using the modifier **−t** with the **+t** option.
+t *index*	Dumps the symbol table entries in the specified range. The range begins at the first symbol table entry or at the entry specified by **−t.** The range ends with the specified indexed entry.
−u	Underlines the name of the file for emphasis.
−v	Dumps information symbolically rather than numerically (for example, Static rather than **0X02**). You can use **−v** with all the options except

−s.

−z *name,number*
Dumps the specified line number entry or a range of line numbers. The range starts at the *number* for the named function.

+z *number* Dumps line numbers for a specified range. The range starts at either the *name* or *number* specified by **−z** The range ends with the *number* specified by **+z**.

Optionally, an option and its modifier can be separated by using blanks. The name can be separated from the number that modifies **−z** by replacing the comma with a blank.

The **odump** command tries to format information in a helpful way, printing information in character, hexadecimal, octal, or decimal, as appropriate.

SEE ALSO

a.out(4), ar(4).

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [**–**] [**–f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name*.**z** with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the **–** argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of **–** in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name*.**z** already exists;
the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

```
pcat name.z
```

or just:

```
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** (without destroying *name*.**z**) use the command:

```
pcat name >nnn
```

pcat returns the number of files it was unable to unpack. Failure may occur if:

> the file name (exclusive of the **.z**) has more than 12 characters;
> the file cannot be opened;
> the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

> a file with the "unpacked" name already exists;
> if the unpacked file cannot be created.

SEE ALSO

cat(1).

NAME

passwd – change login password

SYNOPSIS

passwd [*name*]

DESCRIPTION

This command changes or installs a password associated with the login *name*. Ordinary users may change only the password which corresponds to their login *name*. *passwd* prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4). Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times. Passwords must be constructed to meet certain requirements. By default, the following rules apply:

> Each password must have at least six characters. Only the first eight characters are significant. Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters. Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent. New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

Your system administrator may tighten or relax these rules by setting up the file /etc/passwd.conf. One whose effective user ID is zero is called a super-user; see *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

login(1).
crypt(3C), passwd(4), passwd.conf(4) in the *Programmer's Reference Manual*.
su(1M) in the *System Administrator's Reference Manual*.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste [**−s**] [**−d***list*] *file1 file2...*

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if **−** is used in place of a file name.

The meanings of the options are:

−d	Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
list	One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), \ (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use "" *−d* "\\\\").
−s	Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
−	May be used in place of any file name, to read a line from the standard input. (There is no prompting).

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

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **−s** option, no more than 12 input files may be specified.

NAME

patch - a program for applying a diff file to an original

SYNOPSIS

patch [*options*] *orig patchfile* [**+** [*options*] *orig*]
patch

DESCRIPTION

patch will take a patch file containing any of the three forms of difference listing produced by the *diff* program and apply those differences to an original file, producing a patched version. By default, the patched version is put in place of the original, with the original file backed up to the same name with the extension ".orig", or as specified by the **-b** switch. You may also specify where you want the output to go with a **-o** switch. If *patchfile* is omitted, or is a hyphen, the patch will be read from standard input.

Upon startup, patch will attempt to determine the type of the diff listing, unless over-ruled by a **-c**, **-e**, or **-n** switch. Context diffs and normal diffs are applied by the *patch* program itself, while ed diffs are simply fed to the *ed* editor via a pipe.

patch will try to skip any leading garbage, apply the diff, and then skip any trailing garbage. Thus you could feed an article or message containing a diff listing to *patch*, and it should work. If the entire diff is indented by a consistent amount, this will be taken into account.

With context diffs, and to a lesser extent with normal diffs, *patch* can detect when the line numbers mentioned in the patch are incorrect, and will attempt to find the correct place to apply each hunk of the patch. As a first guess, it takes the line number mentioned for the hunk, plus or minus any offset used in applying the previous hunk. If that is not the correct place, *patch* will scan both forwards and backwards for a set of lines matching the context given in the hunk. First *patch* looks for a place where all lines of the context match. If no such place is found, and it's a context diff, and the maximum fuzz factor is set to 1 or more, then another scan takes place ignoring the first and last line of context. If that fails, and the maximum fuzz factor is set to 2 or more, the first two and last two lines of context are ignored, and another scan is made. (The default maximum fuzz factor is 2.) If *patch* cannot find a place to install that hunk of the patch, it will put the hunk out to a reject file, which normally is the name of the output file plus ".rej". (Note that the rejected hunk will come out in context diff form whether the input patch was a context diff or a normal diff. If the input was a normal diff, many of the contexts will simply be null.) The line numbers on the hunks in the reject file may be different than in the patch file: they reflect the approximate location patch thinks the failed hunks belong in the new file rather than the old one.

As each hunk is completed, you will be told whether the hunk succeeded or failed, and which line (in the new file) *patch* thought the hunk should go on. If this is different from the line number specified in the diff you will be told the offset. A single large offset MAY be an indication that a hunk was installed in the wrong place. You will also be told if a fuzz factor was used to make the match, in which case you should also be slightly suspicious.

If no original file is specified on the command line, *patch* will try to figure out from the leading garbage what the name of the file to edit is. In the header of a context diff, the filename is found from lines beginning with "***" or "---", with the shortest name of an existing file winning. Only context diffs have lines like that, but if there is an "Index:" line in the leading garbage, *patch* will try to use the filename from that line. The context diff header takes precedence over an Index line. If no filename can be intuited from the leading garbage, you will be asked for the name of the file to patch.

(If the original file cannot be found, but a suitable SCCS or RCS file is handy, *patch* will attempt to get or check out the file.)

Additionally, if the leading garbage contains a "Prereq: " line, *patch* will take the first word from the prerequisites line (normally a version number) and check the input file to see if that word can be found. If not, *patch* will ask for confirmation before proceeding.

The upshot of all this is that you should be able to say, while in a news interface, the following:

```
        | patch -d /usr/src/local/blurfl
```

and patch a file in the blurfl directory directly from the article containing the patch.

If the patch file contains more than one patch, *patch* will try to apply each of them as if they came from separate patch files. This means, among other things, that it is assumed that the name of the file to patch must be determined for each diff listing, and that the garbage before each diff listing will be examined for interesting things such as filenames and revision level, as mentioned previously. You can give switches (and another original file name) for the second and subsequent patches by separating the corresponding argument lists by a '+'. (The argument list for a second or subsequent patch may not specify a new patch file, however.)

patch recognizes the following switches:

−b causes the next argument to be interpreted as the backup extension, to be used in place of ".orig".

−c forces *patch* to interpret the patch file as a context diff.

−d causes *patch* to interpret the next argument as a directory, and cd to it before doing anything else.

−D causes *patch* to use the "#ifdef...#endif" construct to mark changes. The argument following will be used as the differentiating symbol. Note that, unlike the C compiler, there must be a space between the **−D** and the argument.

−e forces *patch* to interpret the patch file as an ed script.

−f forces *patch* to assume that the user knows exactly what he or she is doing, and to not ask any questions. It does not suppress commentary, however. Use **−s** for that.

−F<number> sets the maximum fuzz factor. This switch only applied to context diffs, and causes *patch* to ignore up to that many lines in looking for places to install a hunk. Note that a larger fuzz factor increases the odds of a faulty patch. The default fuzz factor is 2, and it may not be set to more than the number of lines of context in the context diff, ordinarily 3.

−l causes the pattern matching to be done loosely, in case the tabs and spaces have been munged in your input file. Any sequence of whitespace in the pattern line will match any sequence in the input file. Normal characters must still match exactly. Each line of the context must still match a line in the input file.

−n forces *patch* to interpret the patch file as a normal diff.

−N causes *patch* to ignore patches that it thinks are reversed or already applied. See also **−R** .

−o causes the next argument to be interpreted as the output file name.

−p<number> sets the pathname strip count, which controls how pathnames found in the patch file are treated, in case the you keep your files in a different directory than the person who sent out the patch. The strip count specifies how many backslashes are to be stripped from the front of the pathname. (Any intervening directory names also go away.) For example, supposing the filename in the patch file was

    /u/howard/src/blurfl/blurfl.c

setting **−p** or **−p0** gives the entire pathname unmodified, **−p1** gives

    u/howard/src/blurfl/blurfl.c

without the leading slash, **−p4** gives

    blurfl/blurfl.c

and not specifying **−p** at all just gives you "blurfl.c". Whatever you end up with is looked for either in the current directory, or the directory specified by the **−d** switch.

−r causes the next argument to be interpreted as the reject file name.

−R tells *patch* that this patch was created with the old and new files swapped. (Yes, I'm afraid that does happen occasionally, human nature being what it is.) *patch* will attempt to swap each hunk around before applying it. Rejects will come out in the swapped format. The **−R** switch will not work with ed diff scripts because there is too little information to reconstruct the reverse operation.

If the first hunk of a patch fails, *patch* will reverse the hunk to see if it can be applied that way. If it can, you will be asked if you want to have the **−R** switch set. If it can't, the patch will continue to be applied normally. (Note: this method cannot detect a reversed patch if it is a normal diff and if the first command is an append (i.e. it should have been a delete) since appends always succeed, due to the fact that a null context will match anywhere. Luckily, most patches add or change lines rather than delete them, so most reversed normal diffs will begin with a delete, which will fail, triggering the heuristic.)

−s makes *patch* do its work silently, unless an error occurs.

−S causes *patch* to ignore this patch from the patch file, but continue on looking for the next patch in the file. Thus

```
patch -S + -S + <patchfile
```

will ignore the first and second of three patches.

−v causes *patch* to print out it's revision header and patch level.

−x<number> sets internal debugging flags, and is of interest only to *patch* patchers.

FILES

/tmp/patch∗

SEE ALSO

diff(1)

NOTES FOR PATCH SENDERS

There are several things you should bear in mind if you are going to be sending out patches. First, you can save people a lot of grief by keeping a patchlevel.h file which is patched to increment the patch level as the first diff in the patch file you send out. If you put a Prereq:

line in with the patch, it won't let them apply patches out of order without some warning. Second, make sure you've specified the filenames right, either in a context diff header, or with an Index: line. If you are patching something in a subdirectory, be sure to tell the patch user to specify a **−p** switch as needed. Third, you can create a file by sending out a diff that compares a null file to the file you want to create. This will only work if the file you want to create doesn't exist already in the target directory. Fourth, take care not to send out reversed patches, since it makes people wonder whether they already applied the patch. Fifth, while you may be able to get away with putting 582 diff listings into one file, it is probably wiser to group related patches into separate files in case something goes haywire.

DIAGNOSTICS

Too many to list here, but generally indicative that *patch* couldn't parse your patch file.

The message "Hmm..." indicates that there is unprocessed text in the patch file and that *patch* is attempting to intuit whether there is a patch in that text and, if so, what kind of patch it is.

CAVEATS

patch cannot tell if the line numbers are off in an ed script, and can only detect bad line numbers in a normal diff when it finds a "change" or a "delete" command. A context diff using fuzz factor 3 may have the same problem. Until a suitable interactive interface is added, you should probably do a context diff in these cases to see if the changes made sense. Of course, compiling without errors is a pretty good indication that the patch worked, but not always.

patch usually produces the correct results, even when it has to do a lot of guessing. However, the results are guaranteed to be correct only when the patch is applied to exactly the same version of the file that the patch was generated from.

BUGS

Could be smarter about partial matches, excessively deviant offsets and swapped code, but that would take an extra pass.

If code has been duplicated (for instance with #ifdef OLDCODE ... #else ... #endif), *patch* is incapable of patching both versions, and, if it works at all, will likely patch the wrong one, and tell you that it succeeded to boot.

If you apply a patch you've already applied, *patch* will think it is a reversed patch, and offer to un-apply the patch. This could be construed as a feature.

The UMIPS-V version of **diff** doesn't produce context diffs. Use */usr/lib/rdiff* with the **−c** option for this.

NAME

pc – MIPS Pascal compiler

SYNOPSIS

pc [option] ... file ...

DESCRIPTION

Pc, the MIPS *ucode* Pascal compiler, produces files in the following formats: MIPS object code in MIPS extended *coff* format (the normal result), binary or symbolic *ucode*, *ucode* object files and binary or symbolic assembly language. *Pc* accepts several types of arguments:

Arguments whose names end with '.p' are assumed to be Pascal source programs. They are compiled, and each object program is left in the file whose name consists of the last component of the source with '.o' substituted for '.p'. The '.o' file is only deleted when a single source program is compiled and loaded all at once.

Arguments whose names end with '.s' are assumed to be symbolic assembly language source programs. They are assembled, producing a '.o' file. Arguments whose names end with '.i' are assumed to be Pascal source after being processed by the C preprocessor. They are compiled without being processed by the C preprocessor.

If the highest level of optimization is specified (with the **−O3** flag) or only ucode object files are to be produced (with the **−j** flag) each Pascal source file is compiled into a *ucode* object file. The *ucode* object file is left in a file whose name consists of the last component of the source with '.u' substituted for '.p'.

The suffixes described below primarily aid compiler development and are not generally used. Arguments whose names end with '.B', '.O', '.S', and '.M' are assumed to be binary *ucode*, produced by the front end, optimizer, ucode object file splitter and ucode merger respectively. Arguments whose names end with '.U' are assumed to be symbolic *ucode*. Arguments whose names end with '.G' are assumed to be binary assembly language, which is produced by the code generator and the symbolic to binary assembler.

Files that are assumed to be binary *ucode*, symbolic *ucode*, or binary assembly language by the suffix conventions are also assumed to have their corresponding symbol table in a file with a '.T' suffix.

Pc always defines the C preprocessor macros **mips, host_mips** and **unix** to the C macro preprocessor. *Pc* defines the C preprocessor macro **LANGUAGE_PASCAL** when a '.p' file is being compiled. *Pc* will define the C preprocessor macro **LANGUAGE_ASSEMBLY** when a '.s' file is being compiled. It also defines **SYSTYPE_SYSV** by default but this changes if the **−systype name** option is specified (see the description below).

The following options are interpreted by *pc* and have the same meaning in *cc*(1). 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.

−g0 Have the compiler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages.

−p0 Do not permit any profiling. This is the default. If loading happens, the standard runtime startup routine (**crt1.o**) is used, no profiling library is searched.

−p1 or **−p**
Set up for profiling by periodically sampling the value of the program counter. This option only effects the loading. When loading happens, this option replaces the standard runtime startup routine with the profiling runtime startup routine (**mcrt1.o**) and searches the level 1 profiling library (**libprof1.a**). When profiling happens, the startup routine calls *monstartup*(3) and produces a file *mon.out* that contains execution-profiling data for use with the postprocessor *prof*(1).

−O0 Turn off all optimizations.

−O1 Turn on all optimizations that can be done quickly. This is the default.

−O or **−O2**
Invoke the global *ucode* optimizer.

−O3 Do all optimizations, including global register allocation. This option must precede all source file arguments. With this option, a *ucode* object file is created for each Pascal source file and left in a '.u' file. The newly created ucode object files, the ucode object files specified on the command line and the runtime startup routine and all the runtime libraries are ucode linked. Optimization is done on the resulting ucode linked file and then it is linked as normal producing an "a.out" file. No resulting '.o' file is left from the ucode linked result as in previous releases. In fact **−c** can no longer be specified with **−O3**.

−feedback *file*
Used with the **−cord** option to specify *file* to be used as a feedback file. This *file* is produced by *prof*(1) with its **−feedback** option from an execution of the program produced by *pixie*(1).

−cord Run the procedure-rearranger, *cord*(1), on the resulting file after linking. The rearrangement is done to reduce the cache conflicts of the program's text. The output of *cord*(1) is left in the file specified by the **−o** *output* option or 'a.out' by default. At least one **−feedback** *file* must be specified.

−j Compile the specified source programs, and leave the *ucode* object file output in corresponding files suffixed with '.u'.

−ko *output*
Name the output file created by the ucode loader as *output*. This file is not removed. If this file is compiled, the object file is left in a file whose name consists of *output* with the suffix changed to a '.o'. If *output* has no suffix, a '.o' suffix is appended to *output*.

−k Pass options that start with a **−k** to the ucode loader. This option is used to specify ucode libraries (with **−kl***x*) and other ucode loader options.

−S Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with '.s'.

−P Run only the C macro preprocessor and put the result for each source file (by suffix convention, i.e. '.p' and '.s') in a corresponding '.i' file. The '.i' file has no '#' lines in it. This sets the **−cpp** option.

−E Run only the C macro preprocessor on the files (regardless of any suffix or not), and send the result to the standard output. This sets the **−cpp** option.

−o *output*
Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−D*name*=*def*
−D*name*
Define the *name* to the C macro 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 specified in **−I** options, and finally in the standard directory (**/usr/include**).

−I This option will cause '#include' files never to be searched for in the standard directory (**/usr/include**).

−G *num*
Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 8 bytes.

−v Print the passes as they execute with their arguments and their input and output files.

−V Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−std Have the compiler produce warnings for things that are not standard in the language. The C macro preprocessor is not run on '.p' files if this option is present.

−cpp Run the C macro preprocessor on Pascal and assembly source files before compiling. This is the default for *pc*(1).

−nocpp
Do not run the C macro preprocessor on Pascal and assembly source files before compiling.

−Olimit *num*
Specify the maximum size, in basic blocks, of a routine that will be optimized by the global optimizer. If a routine has more than this number of basic blocks it will not be optimized and a message will be printed. An option specifying that the global optimizer is to be run (**−O**, **−O2**, or **−O3**) must also be specified. *Num* is assumed to be a decimal number. The default value for *num* is 500 basic blocks.

Either object file target byte ordering can be produced by *pc*. The default target byte ordering matches the machine where the compiler is running. The options **−EB** and **−EL** specify the target byte ordering (big-endian and little-endian, respectively). The compiler also defines a C preprocessor macro for the target byte ordering. These C preprocessor macros are **MIPSEB** and **MIPSEL** for big-endian and little-endian byte ordering respectively.

If the specified target byte ordering does not match the machine where the compiler is running, then the runtime startups and libraries come from **/usr/libeb** for big-endian runtimes on a little-endian machine and from **/usr/libel** for little-endian runtimes on a big-endian machine.

−EB Produce object files targeted for big-endian byte ordering. The C preprocessor macro **MIPSEB** is defined by the compiler.

−EL Produce object files targeted for little-endian byte ordering. The C preprocessor macro **MIPSEL** is defined by the compiler.

The following option is specific for *pc*:

−C Generate code for runtime range checking. The default suppresses range checking.

The option described below is primarily used to provide UNIX compilation environments other than the native compilation environment.

−systype *name*
: Use the named compilation environment *name.* See *compilation*(7) for the compilation environments that are supported and their *name*s. This has the effect of changing the standard directory for '#include' files, the runtime libraries and where runtime libraries are searched for. The new items are located in their usual paths but with */name* prepended to their paths. Also a preprocessor macro of the form **SYSTYPE_***NAME* (with *name* capitalized) is defined in place of the default **SYSTYPE_SYSV.**

The options described below primarily aid compiler development and are not generally used:

−H*c* Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be [**fjusmoca**]. It selects the compiler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T' and is not removed.

−K Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.B' file for binary *ucode,* produced by the front end). These intermediate files are never removed even when a pass encounters a fatal error. When ucode linking is performed and the **−K** option is specified the base name of the files created after the ucode link is 'u.out' by default. If **−ko** *output* is specified, the base name of the object file is *output* without the suffix if it exists or suffixes are appended to *output* if it has no suffix.

−# Converts binary *ucode* files ('.B') or optimized binary ucode files ('.O') to symbolic *ucode* (a '.U' file) using *btou*(1).

−W*c[c...],arg1[,arg2...]*
: Pass the argument[s] *argi* to the compiler pass[es] *c[c..].* The *c's* are one of [**pfjusmocablyz**]. The c's selects the compiler pass in the same way as the **−t** option.

The options **−t[hpfjusmocablyzrPMnt]**, **−h***path,* and **−B***string* select a name to use for a particular pass, startup routine, or standard library. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

The **−EB** or **−EL** options, the **−p[01]** options and the **−systype** option must precede all **−B** options because they can affect the location of runtimes and what runtimes are used.

−t[hpfjusmocablyzrPMnt]
: Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
include	h (see note below)
cpp	p
upas	f
ujoin	j
uld	u
usplit	s

umerge	m
uopt	o
ugen	c
as0	a
as1	b
ld	l
ftoc	y
cord	z
[m]crt[1n].o	r
libexc.a	E
libp.a	P
libm.a	M
libprof1.a	n
btou, utob	t

If the character 'h' is in the **−t** argument then a directory is added to the list of directories to be used in searching for '#include' files. This directory name has the form COMP_TARGET_ROOT/usr/include*string* . This directory is to contain the include files for the *string* release of the compiler. The standard directory is still searched.

−h*path*
Use *path* rather than the directory where the name is normally found.

−B*string*
Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B,** the **−t** option is assumed to be "hpfjusmocablyzrPMnt". This list designates all names. If no **−t** argument has been processed before the **−B** then a **−B***string* is passed to the loader to use with its **−l***x* arguments.

Invoking the compiler with a name of the form **pc***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/.** If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for all include and library names rather than the default **/.** This affects the standard directory for '#include' files, /usr/include, and the standard library, /usr/lib/libc.a. If this is set, the first directory that is searched for libraries, using the **−l***x* option, is COMP_TARGET_ROOT/usr/lib/cmplrs/cc. The standard directories for libraries are then searched, see *ld*(1).

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/ .**

If the environment variable RLS_ID_OBJECT is set, the value is used as the name of an object to link in if a link takes place. This is used to add release identification information to objects. It is always the last object specified to the loader. See *rls_id*(1) for the tools to create this information.

Other arguments are assumed to be either loader options or *Pascal*-compatible object files, typically produced by an earlier *pc* run, or perhaps libraries of *Pascal*-compatible routines. These files, together with the results of any compilations specified, are loaded in the order given, producing an executable program with the default name **a.out.**

FILES

file.p	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary

/usr/lib/cpp	C macro preprocessor
/usr/lib/upas	Pascal front end
/usr/lib/ujoin	binary ucode and symbol table joiner
/usr/bin/uld	ucode loader
/usr/lib/usplit	binary ucode and symbol table splitter
/usr/lib/umerge	procedure intergrator
/usr/lib/uopt	optional global ucode optimizer
/usr/lib/ugen	code generator
/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/lib/crt1.o	runtime startup
/usr/lib/crtn.o	runtime startup
/usr/lib/mcrt1.o	startup for profiling
/usr/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libprof1.a	level 1 profiling library
/usr/lib/libexc.a	Exception handling library
/usr/lib/libp.a	Pascal library
/usr/lib/libm.a	Math library
/usr/include	standard directory for '#include' files
/usr/bin/ld	MIPS loader
/usr/lib/ftoc	interface between *prof*(1) and *cord*(1)
/usr/lib/cord	procedure-rearranger
/usr/bin/btou	binary to symbolic ucode translator
/usr/bin/utob	symbolic to binary ucode translator
mon.out	file produced for analysis by *prof*(1)

Runtime startups and libraries for the opposite byte sex of machine the compiler is running on have the same names but are located in different directories. For big-endian runtimes on a little-endian machine the directory is /usr/libeb and for little-endian runtimes on a big-endian machine the directory is /usr/libel.

SEE ALSO

Languages Programmer's Guide
cc(1), as(1), monstartup(3), prof(1), ld(1), dbx(1), what(1), cord(1), pixie(1), ftoc(1)

DIAGNOSTICS

The diagnostics produced by *pc* are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

NOTES

The standard library, /usr/lib/libc.a, is loaded by using the -lc loader option and not a full path name. The wrong one could be loaded if there are files with the name libc.*astring* in the directories specified with the **−L** loader option or in the default directories searched by the loader.

The handling of include directories and libc.a is confusing.

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [**–***number*] [**–p** *string*] [**–cefns**] [**+***linenumber*] [**+/***pattern***/**] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–number	An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
–p *string*	Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".
–c	Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.
–e	Causes *pg* *not* to pause at the end of each file.
–f	Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *–f* option inhibits *pg* from splitting lines.
–n	Normally, commands must be terminated by a *<newline>* character. This option causes an automatic end of command as soon as a command letter is entered.
–s	Causes *pg* to print all messages and prompts in standout mode (usually inverse video).
+*linenumber*	Start up at *linenumber*.
+/*pattern***/**	Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<*newline*> or <*blank*>
: This causes one page to be displayed. The address is specified in pages.

(+1) **l** With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
: Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L** Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the *−n* option is specified.

*i***/***pattern***/**
: Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
: Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n** Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p** Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w** Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
: Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <*newline*>, even if the *−n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit *pg*.

!*command*
: *Command* is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <*newline*>, even if the *−n* option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1).
terminfo(4) in the *Programmer's Reference Manual* .

ERRORS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

pixie – add profiling code to a program

SYNOPSIS

pixie in_prog_name [options]

DESCRIPTION

Pixie reads an executable program, partitions it into basic blocks, and writes an equivalent program containing additional code that counts the execution of each basic block. (A basic block is a region of the program that can be entered only at the beginning and exited only at the end). Pixie also generates a file containing the address of each of the basic blocks.

When you run the pixie-generated program, it will (provided it terminates normally or via a call to *exit*(2)) generate a file containing the basic block counts. The name of the file is that of the original program with any leading directory names removed and ".Counts" appended. *prof*(1) and *pixstats*(1) can analyze these files and produce a listing of profiling data.

−[no]quiet
: [Permits] or suppresses messages summarizing the binary-to-binary translation process. Default: **−noquiet.**

-[no]branchcounts
: **-branchcounts** inserts extra counters to track whether each branch instruction is taken or not taken. When this option is used, *pixstats* will automatically print more statistics. Default: **-nobranchcounts**.

−[no]idtrace
: [Disable] or enable tracing of instruction and data memory references. **−idtrace** is equivalent to using both **−itrace** and **−dtrace** together. Default: **−noidtrace**

−[no]itrace
: [Disable] or enable tracing of instruction memory references. Default: **−noitrace**

−[no]dtrace
: [Disable] or enable tracing of data memory references. For the moment, **−dtrace** requires **−itrace**. Default: **−nodtrace**

−idtrace_file *number*
: Specify a UNIX file descriptor number for the trace output file. Default: 19.

−bbaddrs *name*
: Specify a name for the file of basic block addresses. Default is to remove any leading directory names from the in_prog_name and append ".Addrs".

-bbcounts *name*
: Specifies the full filename of the basic block counts file. Default: **objfile.Counts.**

-mips1 Use the MIPS1 instruction set (R2000, R3000) for output executable. This is the default.

-mips2 Use the MIPS2 instruction set (a superset of MIPS1) for output executable.

SEE ALSO

prof(1), *pixstats*(1).
The MIPS Languages Programmer's Guide.

BUGS

The handler function address to the signal system calls is not translated, and so programs that receive signals will not work pixified.

Programs that call *vfork()* will not work pixified because the child process will modify the parent state required for pixie operation. Use *fork()* instead.

Pixified code is substantially larger than the original code. Conditional branches that used to fit in the 16-bit branch displacement field may no longer fit, generating a pixie error.

NAME

pixstats – analyze program execution

SYNOPSIS

pixstats program [options]

DESCRIPTION

Pixstats analyzes a program's execution characteristics. To use *pixstats,* first use *Pixie*(1) to translate and instrument the executable object module for the program. Next, execute the translation on an appropriate input. This produces a *.Counts* file. Finally, use *pixstats* to generate a detailed report on opcode frequencies, interlocks, a mini-profile, and more.

−cycle *ns*

Assume a *ns* cycle time when converting cycle counts to seconds.

−r2010

Use r2010 floating point chip operation times and overlap rules. This is the default.

−r2360

Use r2360 floating point board operation times and overlap rules.

−disassemble

Disassemble and show the analyzed object code.

SEE ALSO

pixie(1), prof(1), *The MIPS Languages Programmer's Guide.*

BUGS

Pixstats models execution assuming a perfect memory system. Cache misses etc. will increase execution above the *pixstats* predictions.

NAME

pl1 – MIPS PL/I compiler

SYNOPSIS

pl1 [option] ... file ...

DESCRIPTION

Pl1, the MIPS PL/I compiler, produces either relocatable object ('.o') files or linked executable ('a.out') files. It can also produce binary or symbolic intermediate code called *ucode*, *ucode* object files, and binary or symbolic assembly language. *Pl1* accepts several types of arguments:

Arguments whose names end with '.pl1' or '.pli' are assumed to be PL/I source programs. They are compiled, and each object program is left in the file whose name consists of the last component of the source with '.o' substituted for '.pl1' or '.pli'. The '.o' file is only deleted when a single source program is compiled and loaded all at once.

The suffixes described below primarily aid compiler development and are not generally used. Arguments whose names end with '.F', '.O', '.S', and '.M' are assumed to be binary *ucode*, produced by the front end, optimizer, ucode object file splitter and ucode merger respectively. Arguments whose names end with '.U' are assumed to be symbolic *ucode*. Arguments whose names end with '.G' are assumed to be binary assembly language, which is produced by the code generator and the symbolic to binary assembler.

Files that are assumed to be binary *ucode*, symbolic *ucode*, or binary assembly language by the suffix conventions are also assumed to have their corresponding symbol table in a file with a '.T' suffix.

The following options are interpreted by *pl1*(1). 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.

−g0 Have the compiler produce no symbol table information for symbolic debugging. This is the default.

−g1 Have the compiler produce additional symbol table information for accurate but limited symbolic debugging of partially optimized code.

−g or **−g2**
Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging.

−g3 Have the compiler produce additional symbol table information for full symbolic debugging for fully optimized code. This option makes the debugger inaccurate.

−w Suppress warning messages.

−p0 Do not permit any profiling. This is the default. If loading happens, the standard runtime startup routine (**crt0.o**) is used, no profiling library is searched.

−p1 or **−p**
Set up for profiling by periodically sampling the value of the program counter. This option only affects the loading. When loading happens, this option replaces the standard runtime startup routine with the profiling runtime startup routine (**mcrt0.o**) and searches the level 1 profiling library (**libprof1.a**). When profiling happens, the startup routine calls *monstartup*(3) and produces a file *mon.out* that contains execution-profiling data for use with the postprocessor *mprof*(1).

−O0 Turn off all optimizations.

−O1 Turn on all optimizations that can be done quickly. This is the default.

−O or **−O2**
: Invoke the global *ucode* optimizer.

−S
: Compile the specified source programs and leave the symbolic assembly language output in corresponding files suffixed with '.s'.

−P
: Run only the C macro preprocessor and put the result for each source file (by suffix convention, i.e. '.c' and '.s') in a corresponding '.i' file. The '.i' file has no '#' lines in it.

−o *output*
: Name the final output file *output*. If this option is used, the file 'a.out' is undisturbed.

−G *num*
: Specify the maximum size, in bytes, of a data item that is to be accessed from the global pointer. *Num* is assumed to be a decimal number. If *num* is zero, no data is accessed from the global pointer. The default value for *num* is 512 bytes.

−v
: Print the passes as they execute with their arguments and their input and output files. Also prints resource usage in the C-shell *time* format.

−V
: Print the version of the driver and the versions of all passes. This is done with the *what*(1) command.

−5
: Use the BRL System V emulation include files and libraries instead of the default include files and libraries. The include files are in /usr/5include and the runtime libraries are in /usr/5lib, as modified by other options if appropriate.

The following options are specific to *pl1*:

−ipath *directory*
: Search for %include files in the specified directory rather than the current directory. You may specify a list of directories either by using separate −ipath options, or by placing multiple directory names (separated by ':' characters) after a single −ipath option. The directories will be searched in order.

−defext
: Pay attention to *init* clauses associated with external declarations (the default is to ignore them).

The options described below primarily aid compiler development and are not generally used:

−H*c*
: Halt compiling after the pass specified by the character *c*, producing an intermediate file for the next pass. The *c* can be [**fkjusmoca**]. It selects the compiler pass in the same way as the **−t** option. If this option is used, the symbol table file produced and used by the passes, is the last component of the source file with the suffix changed to '.T' and is not removed.

−K
: Build and use intermediate file names with the last component of the source file's name replacing its suffix with the conventional suffix for the type of file (for example '.F' file for binary *ucode*, produced by the front end). These intermediate files are never removed, even when a pass encounters a fatal error.

−W*c,arg1[,arg2...]*
: Pass the argument[s] *argi* to the compiler pass *c*. The *c* is one of [**pfkjusmocabl**]. The c selects the compiler pass in the same way as the **−t** option.

The options **−t**[**hpfjusmocablrnt**], **−h***path,* and **−B***string* select a name to use for a particular pass, startup routine, or standard library. These arguments are processed from left to right so their order is significant. When the **−B** option is encountered, the selection of names takes place using the last **−h** and **−t** options. Therefore, the **−B** option is always required when using **−h** or **−t**. Sets of these options can be used to select any combination of names.

The **−EB** or **−EL** options, any of the **−p[0123]** options and any of the **−g[123]** options must precede all **−B** options because they can affect the location of runtimes and what runtimes are used.

−t[**hpfjusmocablrnt**]

Select the names. The names selected are those designated by the characters following the **−t** option according to the following table:

Name	Character
include	h (see note below)
cpp	p
pl1fe	f
ulpi	k
ujoin	j
uld	u
usplit	s
umerge	m
uopt	o
ugen	c
as0	a
as1	b
ld	l
[m]crt[01n].o	r
libpl1.a	1
libprof1.a	n
btou, utob	t

If the character 'h' is in the **−t** argument then a directory is added to the list of directories to be used in searching for '#include' files. This directory name has the form ROOTDIR/include*string* . This directory is to contain the include files for the *string* release of the compiler. The standard directory is still searched.

−h*path*

Use *path* rather than the directory where the name is normally found.w

−B*string*

Append *string* to all names specified by the **−t** option. If no **−t** option has been processed before the **−B,** the **−t** option is assumed to be "hpfjusmocablrint". This list designates all names. If no **−t** argument has been processed before the **−B** then a **−B***string* is passed to the loader to use with it's **−l***x* arguments.

Note: The compiler front-end provides a number of options specified in the format **−Wf,***option* where *option* can be one or more of the following:

-exp Produces an expanded listing. The default is no expanded listing.

-l,*filename*

produces a compiler listing, where *filename* is the name of the file where the listing is to be placed. If *filename* isn't specified, the listing is placed in *source* the *pl1* suffix.

-longint

Changes the default precision for FIXED BINARY variables from (15) to (31).

-lowercase

Converts all uppercase names of internal and external variable, constants, and procedure names to lowercase. When calling routines written in languages other than PL/I, lowercase names should be used. Refer to the section EXTERNAL NAMES in Part I, Chapter 3 *MIPS-PL/I Programmer's Guide and Language Reference* manual for additional information.

-nest indicates the nesting level of DO ... END, PROC ... END, etc., on the listing. This option must be combined with the -l option.

-noincludes
Directs the compiler not to include the contents of %INCLUDE files in the listing file. The default is to include file contents in the listing file.

-range Generates code to check all subscript references to see if they are valid.

-si Suppresses WARNING (level 1) error messages. Appendix A in Part I of the *MIPS-PL/I Programmer's Guide and Language Reference* manual contains an explanation of how to interpret compiler error messages.

-setnull,*n*
The built-in function NULL() returns a pointer whose integer value is zero unless this option is used and *n* specified. For example, to cause the value returned by the pointer to be -1, use:

```
-Wf,-setnull,-1
```

-force_unalign
The example below causes the compiler to treat all formal and actual; arguments of type *bit* as if they were unaligned:

```
-Wk,-force_unalign
```

This degrades execution speed of the compiled program but relaxes requirement that formal and *bit* parameter types match exactly.

Invoking the compiler with a name of the form **pl1***string* has the same effect as using a **−B***string* option on the command line.

If the environment variable COMP_HOST_ROOT is set, the value is used as the root directory for all pass names rather than the default **/**. If the environment variable COMP_TARGET_ROOT is set, the value is used as the root directory for library names rather than the default **/**. This affects the standard library, /usr/lib/libc.a. If this is set, the first directory that is searched for libraries, using the **−l***x* option, is COMP_TARGET_ROOT/usr/lib/cmplrs/cc. The standard directories for libraries are then searched, see *ld*(1).

If the environment variable TMPDIR is set, the value is used as the directory to place any temporary files rather than the default **/tmp/** .

Other arguments are assumed to be either loader options or *pl1*-compatible object files, typically produced by an earlier *pl1* run, or perhaps libraries of *pl1*-compatible routines. These files, together with the results of any compilations specified, are loaded in the order given, producing an executable program with the default name **a.out.**

FILES

file.pl1	input file
file.o	object file
a.out	loaded output
/tmp/ctm?	temporary
/usr/lib/cpp	C macro preprocessor
/usr/lib/pl1fe	pl1 front end

/usr/lib/ulpi	front-end to ucode translator
/usr/lib/ujoin	binary ucode and symbol table joiner
/usr/bin/uld	ucode loader
/usr/lib/usplit	binary ucode and symbol table splitter
/usr/lib/umerge	procedure intergrator
/usr/lib/uopt	optional global ucode optimizer
/usr/lib/ugen	code generator
/usr/lib/as0	symbolic to binary assembly language translator
/usr/lib/as1	binary assembly language assembler and reorganizer
/usr/lib/crt0.o	runtime startup
/usr/lib/mcrt0.o	startup for profiling
/usr/lib/libc.a	standard library, see *intro*(3)
/usr/lib/libprof1.a	level 1 profiling library
/usr/include	standard directory for '#include' files
/usr/bin/ld	MIPS loader
/usr/bin/btou	binary to symbolic ucode translator
/usr/bin/utob	symbolic to binary ucode translator
mon.out	file produced for analysis by *prof*(1)

Runtime startups and libraries for the opposite byte sex of machine the compiler is running on have the same names but are located in different directories. For big-endian runtimes on a little-endian machine the directory is /usr/libeb and for little-endian runtimes on a big-endian machine the directory is /usr/libel.

SEE ALSO

coff(5), monstartup(3), prof(1), ld(1), dbx(1), what(1)

DIAGNOSTICS

The diagnostics produced by *pl1* are intended to be self-explanatory. Occasional messages may be produced by the assembler or loader.

NOTES

The standard library, /usr/lib/libc.a, is loaded by using the -lc loader option and not a full path name. The wrong one could be loaded if there are files with the name libc.a*string* in the directories specified with the **−L** loader option or in the default directories searched by the loader.

The handling of include directories and libc.a is confusing.

NAME

pr – print files

SYNOPSIS

pr [[-column] [**−**wwidth] [**−a**]] [**−e**ck] [**−i**ck] [**−drtfp**] [+page] [**−n**ck] [**−o**offset] [**−l**length] [**−s**separator] [**−h** header] [file ...]

pr [[**−m**] [**−w**width]] [**−e**ck] [**−i**ck] [**−drtfp**] [+page] [**−n**ck] [**−o**offset] [**−l**length] [**−s**separator] [**−h** header] file1 file2 ...

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is **−**, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **−h**), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **−s** option is used, lines are not truncated and columns are separated by the *separator* character.

Either −*column* or **−m** should be used to produce multi-column output. **−a** should only be used with −*column* and not **−m**.

OPTIONS

Command line options are

+*page*	Begin printing with page numbered *page* (default is 1).
−*column*	Print *column* columns of output (default is 1). Output appears as if **−e** and **−i** are turned on for multi-column output. May not use with **-m**.
−a	Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
−m	Merge and print all files simultaneously, one per column. The maximum number of files that may be specifed is eight. If a line is too long to fit in a column, it is truncated. May not use with −*column*.
−d	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
−e*ck*	Expand input tabs to character positions k+1, 2$*k$+1, 3$*k$+1, etc. If k is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If c (any non-digit character) is given, it is treated as the input tab character (default for c is the tab character).
−i*ck*	In output, replace white space wherever possible by inserting tabs to character positions k+1, 2$*k$+1, 3$*k$+1, etc. If k is 0 or is omitted, default tab settings at every eighth position are assumed. If c (any non-digit character) is given, it is treated as the output tab character (default for c is the tab character).

−n*ck*	Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of **−m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
−w*width*	Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and **-m**). There is no line limit for single column output.
−o*offset*	Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
−l*length*	Set the length of a page to *length* lines (default is 66). **−l0** is reset to **−l66**. When the value of *length* is 10 or less, **−t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **−l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.
−h *header*	Use *header* as the text line of the header to be printed instead of the file name. **−h** is ignored when **−t** is specified or **−l***length* is specified and the value of *length* is 10 or less. (**−h is the only** *pr* option requiring space between the option and argument.)
−p	Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).
−f	Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
−r	Print no diagnostic reports on files that will not open.
−t	Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **−t** overrides the **−h** option.
−s*separator*	Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **-w** is specified.

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by “file list”:

```
pr −3dh "file list" file1 file2
```

Copy **file1** to **file2**, expanding tabs to columns 10, 19, 28, 37, . . . :

```
pr −e9 −t <file1 >file2
```

Print **file1** and **file2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr −t -n file1 |pr -t -m -n file2 -
```

FILES

/dev/tty*　　to delay messages enabling them to print at the bottom of files rather than interspersed throughout printed output.

SEE ALSO

cat(1), pg(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

sh(1), environ(7), csh(1)

NAME

prof – analyze profile data

SYNOPSIS

prof [options] [prog_name [pcsampling_data_file ...]]
prof −note "*comment string*" **−pixie** [*options*] [*prog_name* [*bbaddrs_file* [*bbcounts_file* ...]]]

DESCRIPTION

Prof analyzes one or more data files generated by the MIPS compiler's execution-profiling system and produces a listing. Prof can also combine those data files or produce a feedback file that lets the optimizer take into account the program's runtime behavior during a subsequent compilation. Profiling is a three-step process: first compile the program, then execute it, and finally run *prof* to analyze the data.

The compiler system provides two kinds of profiling:

1. *pc-sampling* interrupts the program periodically, recording the value of the program counter.

2. *basic-block counting* divides the program into blocks delimited by labels, jump instructions, and branch instructions. It counts the number of times each block executes. This provides more detailed (line by line) information than pc-sampling.

Using pc-sampling

To use pc-sampling, compile your program with the option **−p** (strictly speaking, it is sufficient to use this option only when linking the program.) Then run the program, which allocates extra memory to hold the profile data, and (provided the program terminates normally or calls *exit*(2)) records the data in a file at the end of execution.

The environment variable PROFDIR determines the name of the pc-sampling data file and determines whether pc-sampling takes place: if it is not set, the pc-sampling data file is named "mon.out"; if it is set to the empty string, no profiling occurs; if it is set to a non-empty string, the file is named "string/pid.progname," where "pid" is the process id of the executing program and "progname" is the program's name, as it appears in argv[0]. The subdirectory "string" must already exist.

After running your program, use *prof* to analyze the pc-sampling data file.

For example:

```
cc -c myprog.c
cc -p -o myprog myprog.o
myprog                      (generates "mon.out")
prof myprog mon.out
```

When you use *prof* for pc-sampling, the program name defaults to *a.out* and the pc-sampling data file name defaults to *mon.out*; if you specify more than one pc-sampling data file, *prof* reports the sum of the data.

Using basic-block counting

To use basic-block counting, compile your program without the option **−p**. Use *pixie*(1) to translate your program into a profiling version and generate a file, whose name ends in ".Addrs", containing block addresses. Then run the profiling version, which (assuming the program terminates normally or calls *exit*(2)) will generate a file, whose name ends in ".Counts", containing block counts. Then use *prof* with the **−pixie** option to analyze the bbaddrs and bbcounts files. Notice that you must tell *prof* the name of your original program, not the name of the profiling version.

For example:

```
cc -c myprog.c
cc -o myprog myprog.o
pixie -o myprog.pixie myprog          (generates "myprog.Addrs")
myprog.pixie                          (generates "myprog.Counts")
prof -pixie myprog myprog.Addrs myprog.Counts
```

When you use *prof* with the **−pixie** option, the program name defaults to *a.out*, the bbaddrs file name defaults to "*program_name*.Addrs", and the bbcounts file name defaults to "*program_name*.Counts". If you specify more than one bbcounts file (never specify more than one bbaddrs file), *prof* reports the sum of the data. **−note** "*comment string*" If you use this argument, the "*comment string*" appears near the beginning of the listing as a comment.

Options to *prof*

For each *prof* option, you need type only enough of the name to distinguish it from the other options (usually the first character is sufficient). Unless otherwise noted, each part of the listing operates only on the set of procedures that results from the combination of the **−exclude** and **−only** options.

If the options you specify would neither produce a listing nor generate a file, *prof* uses **−procedures** plus **−heavy** by default.

−pixie Selects pixie mode, as opposed to pc-sampling mode.

−procedures
Reports time spent per procedure (using data obtained from pc-sampling or basic-block counting; the listing tells which one). For basic-block counting, this option also reports the number of invocations per procedure.

−heavy
Reports the most heavily used lines in descending order of use (requires basic-block counting).

−lines Like **−heavy**, but gives the lines in order of occurrence.

−invocations
For each procedure, reports how many times the procedure was invoked from each of its possible callers (requires basic-block counting). For this listing, the **−exclude** and **−only** options apply to callees, but not to callers.

−zero Prints a list of procedures that were never invoked (requires basic-block counting).

−testcoverage
Reports all lines that never executed (requires basic-block counting). .

−feedback *filename*
Produces a file with information that the compiler system can use to decide what parts of the program will benefit most from global optimization and what parts will benefit most from in-line procedure substitution (requires basic-block counting). See *umerge*(1) and *uopt*(1).

−merge *filename*
Sums the pc-sampling data files (or, in pixie mode, the bbcounts files) and writes the result into a new file with the specified name. The **−only** and **−exclude** options have no affect on the merged data.

−only *procedure_name*
If you use one or more **−only** options, the profile listing includes only the named procedures, rather than the entire program. If any option uses an uppercase "O" for "Only," *prof* uses only the named procedures, rather than the entire program, as the

base upon which it calculates percentages.

−exclude *procedure_name*
If you use one or more **−exclude** options, the profiler omits the specified procedure and its descendents from the listing. If any option uses an uppercase "E" for "Exclude," *prof* also omits that procedure from the base upon which it calculates percentages.

−clock *megahertz*
Alters the appropriate parts of the listing to reflect the clock speed of the CPU. If you do not specify *megahertz*, it defaults to "8.0".

−quit *n*
Truncates the **−procedures** and **−heavy** listings. It can truncate after *n* lines (if *n* is an integer), after the first entry that represents less than *n* percent of the total (if *n* is followed immediately by a "%" character), or after enough entries have been printed to account for *n* percent of the total (if *n* is followed immediately by "cum%"). For example, "−quit 15" truncates each part of the listing after 15 lines of text, "−quit 15%" truncates each part after the first line that represents less than 15 percent of the whole, and "−quit 15cum%" truncates each part after the line that brought the cumulative percentage above 15 percent.

FILES

crt0.o	normal startup code
mcrt0.o	startup code for pc-sampling
libprof1.a	library for pc-sampling
mon.out	default pc-sampling data file

SEE ALSO
monitor(3), profil(2), pixie(2), cc(1), pc(1), f77(1), as(1), *The MIPS Languages Programmer's Guide.*

FEATURES
Provided you do not use **−pixie**, *prof* processes "mon.out" files produced by earlier versions of the compiler system using the obsolete **−p2** or **−p3** options.

BUGS
Prof does not yet take into account interactions among floating-point instructions.

NAME

prs – print an SCCS file

SYNOPSIS

prs [**−d**[dataspec]] [**−r**[*SID*]] [**−e**] [**−l**] [**−c***date-time*] [**−a**] *file...*

DESCRIPTION

prs prints, on the standard output, parts or all of an SCCS file [see *sccsfile*(4)] in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with **s.**), and unreadable files are silently ignored. If a name of **−** is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

−d*[dataspec]* Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.

−r*[SID]* Used to specify the *SCCS IDentification* (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

−e Requests information for all deltas created *earlier* than and including the delta designated via the **−r** keyletter or the date given by the **−c** option.

−l Requests information for all deltas created *later* than and including the delta designated via the **−r** keyletter or the date given by the **−c** option.

c date-time The cutoff date-time **−c**cutoff is in the form:

YY[MM[DD[HH[MM[SS]]]]]

−c*date-time* Units omitted from the date-time default to their maximum possible values; that is, **−c7502** is equivalent to **-c750228235959**. Any number of non-numeric characters may separate the various 2-digit pieces of the *cutoff* date in the form: **"−c77/2/2 9:22:25"**.

−a Requests printing of information for both removed, i.e., delta type = *R*, [see *rmdel*(1)] and existing, i.e., delta type = *D*, deltas. If the **−a** keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file [see *sccsfile*(4)] have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.
A tab is specified by **\t** and carriage return/new-line is specified by **\n**. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

TABLE 1. SCCS Files Data Keywords

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D*˜or˜*R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS:˜:DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS:˜:DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS:˜:DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes*˜or˜*no*	S

TABLE 1. SCCS Files Data Keywords (continued)

Keyword	*Data Item*	*File Section*	*Value*	*Format*
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes*˜or˜*no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes*˜or˜*no*	S
:J:	Joint edit flag	"	*yes*˜or˜*no*	S
:LK:	Locked releases	"	:R: ...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes*˜or˜*no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y:˜:M:˜:I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt:˜=˜:DT:˜:I:˜:D:˜:T:˜:P:˜:DS:˜:DP:

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),
sccsfile(4).
help(1) in the *User's Reference Manual*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – report process status

SYNOPSIS

ps [options]

DESCRIPTION

ps prints certain information about active processes. Without *options*, information is printed about processes associated with the controlling terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

−e	Print information about **e**very process now running.
−d	Print information about all processes except process group leaders.
−a	Print information about **a**ll processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
−f	Generate a **f**ull listing. (See below for significance of columns in a full listing.)
−l	Generate a **l**ong listing. (See below.)
−n *name*	Take argument signifying an alternate system *name* in place of **/unix.**
−t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or, if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).
−p *proclist*	List only process data whose process ID numbers are given in *proclist*.
−u *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **−f** option, which prints the login name.
−g *grplist*	List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the **−f** option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **−f** option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F	(l)	Flags (hexadecimal and additive) associated with the process

MIPS Computers

00	Process has terminated: process table entry now available.
01	A system process: always in primary memory.
02	Parent is tracing process.
04	Tracing parent's signal has stopped process: parent is waiting [*ptrace*(2)].
08	Process cannot wakeup by signal.
10	Process currently in primary memory.
20	Process currently in primary memory; locked until an event completes.

S	(l)	The state of the process:

O	Process is running on a processor.
S	Sleeping: process is waiting for an event to complete.
R	Runnable: process is on run queue.
I	Idle: process is being created.
Z	Zombie state: process terminated and parent not waiting.
T	Traced: process stopped by a signal because parent is tracing it.
X	SXBRK state: process is waiting for more primary memory.

UID	(f,l)	The user ID number of the process owner (the login name is printed under the **−f** option).
PID	(all)	The process ID of the process (this datum is necessary in order to kill a process).
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process (higher numbers mean lower priority).
NI	(l)	Nice value, used in priority computation.
ADDR	(l)	The memory address of the process.
SZ	(l)	The size (in pages or clicks) of the swappable process's image in main memory.
WCHAN	(l)	The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).
STIME	(f)	The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)
TTY	(all)	The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).
TIME	(all)	The cumulative execution time for the process.

COMMAND(all) The command name (the full command name and its arguments are printed under the **−f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **<defunct>**.

FILES

/dev	
/dev/sxt/*	
/dev/tty*	
/dev/xt/*	terminal ("tty") names searcher files
/dev/kmem	kernel virtual memory
/dev/swap	the default swap device
/dev/mem	memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/unix	system namelist

SEE ALSO

kill(1), nice(1).
getty(1M) in the *System Administrator's Reference Manual*.

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

On a heavily loaded system, *ps* may report an *lseek(2)* error and exit. *ps* may seek to an invalid user area address: having got the address of a process' user area, *ps* may not be able to seek to that address before the process exits and the address becomes invalid.

ps −ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to a UNIX system administrator.

NAME

ranlib – convert archives to random libraries

SYNOPSIS

ranlib *filename* ...

DESCRIPTION

ranlib executes the command **ar ts** on each of the named files, which are expected to be archive files. This command is not strictly needed, as the *ar(1)* command builds the random library table into the archive by default. It is provided for makefiles that execute it.

SEE ALSO

ar(1)

NAME

rcp – remote file copy

SYNOPSIS

rcp [**−p**] file1 file2
rcp [**−p**] [**−r**] file ... directory

DESCRIPTION

rcp copies files between machines. Each *file* or *directory* argument is either a remote file name of the form "rhost:path", or a local file name (containing no ':' characters, or a '/' before any ':'s).

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

By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the *umask*(2) on the destination host is used. The **−p** option causes *rcp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the *umask*.

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A *path* on a remote host may be quoted (using \, ", or ') so that the metacharacters are interpreted remotely.

rcp does not prompt for passwords; your current local user name must exist on *rhost* and allow remote command execution via *rsh*(1C).

rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form "rname@rhost" to use *rname* rather than the current user name on the remote host. The destination hostname may also take the form "rhost.rname" to support destination machines that are running 4.2BSD versions of *rcp*.

SEE ALSO

cp(1), ftp(1C), rsh(1C), rlogin(1C)

ERRORS

Doesn't detect all cases where the target of a copy might be a file in cases where only a directory should be legal.
Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.

ORIGIN

4.3 BSD

NAME

rcs – change RCS file attributes

SYNOPSIS

rcs [*options*] *file* ...

DESCRIPTION

rcs creates new RCS files or changes attributes of existing ones. An RCS file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. For **rcs** to work, the caller's login name must be on the access list, except if the access list is empty, the caller is the owner of the file or the superuser, or the **-i** option is present.

Files ending in ',v' are RCS files, all others are working files. If a working file is given, **rcs** tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co(1)*.

−i	creates and initializes a new RCS file, but does not deposit any revision. If the RCS file has no path prefix, **rcs** tries to place it first into the subdirectory ./RCS, and then into the current directory. If the RCS file already exists, an error message is printed.
−a*logins*	appends the login names appearing in the comma-separated list *logins* to the access list of the RCS file.
−A*oldfile*	appends the access list of *oldfile* to the access list of the RCS file.
−e[*logins*]	erases the login names appearing in the comma-separated list *logins* from the access list of the RCS file. If *logins* is omitted, the entire access list is erased.
−c*string*	sets the comment leader to *string*. The comment leader is printed before every log message line generated by the keyword Log during checkout (see *co(1)*). This is useful for programming languages without multi-line comments. During **rcs -i** or initial *ci(1)*, the comment leader is guessed from the suffix of the working file.
−l[*rev*]	locks the revision with number *rev*. If a branch is given, the latest revision on that branch is locked. If *rev* is omitted, the latest revision on the trunk is locked. Locking prevents overlapping changes. A lock is removed with *ci(1)* or **rcs -u** (see below).
−u[*rev*]	unlocks the revision with number *rev*. If a branch is given, the latest revision on that branch is unlocked. If *rev* is omitted, the latest lock held by the caller is removed. Normally, only the locker of a revision may unlock it. Somebody else unlocking a revision breaks the lock. This causes a mail message to be sent to the original locker. The message contains a commentary solicited from the breaker. The commentary is terminated with a line containing a single '.' or control-D.
−b	causes all first branches to be followed to the end.
−L	sets locking to *strict*. Strict locking means that the owner of an RCS file is not exempt from locking for checkin. This option should be used for files that are shared.
−U	sets locking to non-strict. Non-strict locking means that the owner of a file need not lock a revision for checkin. This option should NOT be used for files that are shared. The default (**-L** or **-U**) is determined by your system administrator.
−n*name*[:*rev*]	associates the symbolic name *name* with the branch or revision *rev*. *Rcs*

prints an error message if *name* is already associated with another number. If *rev* is omitted, the symbolic name is deleted.

−N*name*[:*rev*]
same as **-n**, except that it overrides a previous assignment of *name*.

−o*range*
deletes ("outdates") the revisions given by *range*. A range consisting of a single revision number means that revision. A range consisting of a branch number means the latest revision on that branch. A range of the form *rev1−rev2* means revisions *rev1* to *rev2* on the same branch, *−rev* means from the beginning of the branch containing *rev* up to and including *rev*, and *rev−* means from revision *rev* to the end of the branch containing *rev*. None of the outdated revisions may have branches or locks.

−q
quiet mode; diagnostics are not printed.

−s*state*[:*rev*]
sets the state attribute of the revision *rev* to *state*. If *rev* is omitted, the latest revision on the trunk is assumed; If *rev* is a branch number, the latest revision on that branch is assumed. Any identifier is acceptable for *state*. A useful set of states is *Exp* (for experimental), *Stab* (for stable), and *Rel* (for released). By default, *ci(1)* sets the state of a revision to *Exp*.

−t[*txtfile*]
writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, **rcs** prompts the user for text supplied from the std. input, terminated with a line containing a single '.' or control-D. Otherwise, the descriptive text is copied from the file *txtfile*. If the **-i** option is present, descriptive text is requested even if **-t** is not given. The prompt is suppressed if the std. input is not a terminal.

DIAGNOSTICS

The RCS file name and the revisions outdated are written to the diagnostic output. The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

FILES

The caller of the command must have read/write permission for the directory containing the RCS file and read permission for the RCS file itself. *rcs* creates a semaphore file in the same directory as the RCS file to prevent simultaneous update. For changes, **rcs** always creates a new file. On successful completion, **rcs** deletes the old one and renames the new one. This strategy makes links to RCS files useless.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.7 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

co(1), ci(1), ident(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4), sccstorcs(1M).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rcsdiff – compare RCS revisions

SYNOPSIS

rcsdiff [**−biwt**] [**−cefhn**] [**−r***rev1*] [**−r***rev2*] file ...

DESCRIPTION

rcsdiff runs *diff* (1) to compare two revisions of each RCS file given. A file name ending in ',v' is an RCS file name, otherwise a working file name. *Rcsdiff* derives the working file name from the RCS file name and vice versa, as explained in *co* (1). Pairs consisting of both an RCS and a working file name may also be specified.

All options except **−r** have the same effect as described in *diff*(1).

If both *rev1* and *rev2* are omitted, *rcsdiff* compares the latest revision on the trunk with the contents of the corresponding working file. This is useful for determining what you changed since the last checkin.

If *rev1* is given, but *rev2* is omitted, *rcsdiff* compares revision *rev1* of the RCS file with the contents of the corresponding working file.

If both *rev1* and *rev2* are given, *rcsdiff* compares revisions *rev1* and *rev2* of the RCS file.

Both *rev1* and *rev2* may be given numerically or symbolically.

EXAMPLES

The command

```
rcsdiff  f.c
```

runs *diff* on the latest trunk revision of RCS file f.c,v and the contents of working file f.c.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.5 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci (1), co (1), diff (1), ident (1), rcs (1), rcsintro (1), rcsmerge (1), rlog (1), rcsfile (4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rcsintro – introduction to RCS commands

DESCRIPTION

The Revision Control System (RCS) manages multiple revisions of text files. RCS automates the storing, retrieval, logging, identification, and merging of revisions. RCS is useful for text that is revised frequently, for example programs, documentation, graphics, papers, form letters, etc.

The basic user interface is extremely simple. The novice only needs to learn two commands: *ci* and *co*. *Ci*, short for "checkin", deposits the contents of a text file into an archival file called an RCS file. An RCS file contains all revisions of a particular text file. *Co*, short for "checkout", retrieves revisions from an RCS file.

SEE ALSO

ci(1), co(1), ident(1), merge(1), rcs(1), rcsdiff(1), rcsmerge(1), rlog(1), rcsfile(4).
Walter F. Tichy, "An Introduction to the Revision Control System", Programmer Supplementary Documents, Volume 1 (PS1), #13

NAME

rcsmerge − merge RCS revisions

SYNOPSIS

rcsmerge -r*rev1* [**-r***rev2*] [**-p**] file

DESCRIPTION

rcsmerge incorporates the changes between *rev1* and *rev2* of an RCS file into the corresponding working file. If **-p** is given, the result is printed on the std. output, otherwise the result overwrites the working file.

A file name ending in ',v' is an RCS file name, otherwise a working file name. *Merge* derives the working file name from the RCS file name and vice versa, as explained in *co* (1). A pair consisting of both an RCS and a working file name may also be specified.

rev1 may not be omitted. If *rev2* is omitted, the latest revision on the trunk is assumed. Both *rev1* and *rev2* may be given numerically or symbolically.

rcsmerge prints a warning if there are overlaps, and delimits the overlapping regions as explained in *co -j*. The command is useful for incorporating changes into a checked-out revision.

EXAMPLES

Suppose you have released revision 2.8 of f.c. Assume furthermore that you just completed revision 3.4, when you receive updates to release 2.8 from someone else. To combine the updates to 2.8 and your changes between 2.8 and 3.4, put the updates to 2.8 into file f.c and execute

```
rcsmerge -p -r2.8 -r3.4 f.c >f.merged.c
```

Then examine f.merged.c. Alternatively, if you want to save the updates to 2.8 in the RCS file, check them in as revision 2.8.1.1 and execute *co -j*:

```
ci -r2.8.1.1 f.c
co -r3.4 -j2.8:2.8.1.1 f.c
```

As another example, the following command undoes the changes between revision 2.4 and 2.8 in your currently checked out revision in f.c.

```
rcsmerge -r2.8 -r2.4 f.c
```

Note the order of the arguments, and that f.c will be overwritten.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.4 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci (1), co (1), merge (1), ident (1), rcs (1), rcsdiff (1), rlog (1), rcsfile (4).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

ERRORS

rcsmerge does not work for files that contain lines with a single '.'.

NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [**–**] files

DESCRIPTION

The *regcmp* command performs a function similar to *regcmp*(3X) and, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file***.i**. If the **–** option is used, the output will be placed in *file***.c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *file***.i** files may thus be *included* in C programs, or *file***.c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name    "([A–Za–z][A–Za–z0–9_]*)$0"

telno   "\({0,1}([2–9][01][1–9])$0\){0,1} *"
        "([2–9][0–9]{2})$1[ –]{0,1}"
        "([0–9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X).

NAME

rlog – print log messages and other information about RCS files

SYNOPSIS

rlog [options] file ...

DESCRIPTION

rlog prints information about RCS files. Files ending in ',v' are RCS files, all others are working files. If a working file is given, *rlog* tries to find the corresponding RCS file first in directory ./RCS and then in the current directory, as explained in *co* (1).

rlog prints the following information for each RCS file: RCS file name, working file name, head (i.e., the number of the latest revision on the trunk), access list, locks, symbolic names, suffix, total number of revisions, number of revisions selected for printing, and descriptive text. This is followed by entries for the selected revisions in reverse chronological order for each branch. For each revision, *rlog* prints revision number, author, date/time, state, number of lines added/deleted (with respect to the previous revision), locker of the revision (if any), and log message. Without options, *rlog* prints complete information. The options below restrict this output.

−L ignores RCS files that have no locks set; convenient in combination with **-R**, **-h**, or **-l**.

−R only prints the name of the RCS file; convenient for translating a working file name into an RCS file name.

−h prints only RCS file name, working file name, head, access list, locks, symbolic names, and suffix.

−t prints the same as **-h**, plus the descriptive text.

−d*dates* prints information about revisions with a checkin date/time in the ranges given by the semicolon-separated list of *dates*. A range of the form *d1*<*d2* or *d2*>*d1* selects the revisions that were deposited between *d1* and *d2*, (inclusive). A range of the form <*d* or *d*> selects all revisions dated *d* or earlier. A range of the form *d*< or >*d* selects all revisions dated *d* or later. A range of the form *d* selects the single, latest revision dated *d* or earlier. The date/time strings *d, d1,* and *d2* are in the free format explained in *co* (1). Quoting is normally necessary, especially for < and >. Note that the separator is a semicolon.

−l[*lockers*] prints information about locked revisions. If the comma-separated list *lockers* of login names is given, only the revisions locked by the given login names are printed. If the list is omitted, all locked revisions are printed.

−r*revisions* prints information about revisions given in the comma-separated list *revisions* of revisions and ranges. A range *rev1*−*rev2* means revisions *rev1* to *rev2* on the same branch, −*rev* means revisions from the beginning of the branch up to and including *rev*, and *rev*− means revisions starting with *rev* to the end of the branch containing *rev*. An argument that is a branch means all revisions on that branch. A range of branches means all revisions on the branches in that range.

−s*states* prints information about revisions whose state attributes match one of the states given in the comma-separated list *states*.

−w[*logins*] prints information about revisions checked in by users with login names appearing in the comma-separated list *logins*. If *logins* is omitted, the user's login is assumed.

rlog prints the intersection of the revisions selected with the options **-d**, **-l**, **-s**, **-w**, intersected with the union of the revisions selected by **-b** and **-r**.

EXAMPLES

```
rlog -L -R RCS/*,v
rlog -L -h RCS/*,v
rlog -L -l RCS/*,v
rlog RCS/*,v
```

The first command prints the names of all RCS files in the subdirectory 'RCS' which have locks. The second command prints the headers of those files, and the third prints the headers plus the log messages of the locked revisions. The last command prints complete information.

DIAGNOSTICS

The exit status always refers to the last RCS file operated upon, and is 0 if the operation was successful, 1 otherwise.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 1.5 ; Release Date: 89/01/28 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci (1), co (1), ident(1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rcsfile (4), sccstorcs (1M).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revision Control System," in *Proceedings of the 6th International Conference on Software Engineering*, IEEE, Tokyo, Sept. 1982.

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**−e** *c*] [**−8**] [**−L**] [**−l** username]
rhost [**−e***c*] [**−8**] [**−L**] [**−l** username]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file .rhosts in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the .rhosts file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The terminal or window size is also copied to the remote system if the server supports the option, and changes in size are reflected as well. All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. The optional argument **−8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q. The argument **−L** allows the rlogin session to be run in litout mode. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A different escape character may be specified by the **−e** option. There is no space separating this option flag and the argument character.

SEE ALSO

rsh(1C)

FILES

/usr/hosts/*	for *rhost* version of the command

ERRORS

More of the environment should be propagated.

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**−e** *c*] [**−8**] [**−L**] [**−l** username]

DESCRIPTION

rlogin connects your terminal on the current local host system *lhost* to the remote host system *rhost*.

Each host has a file */etc/hosts.equiv* which contains a list of *rhost*'s with which it shares account names. (The host names must be the standard names as described in *rsh*(1C).) When you *rlogin* as the same user on an equivalent host, you don't need to give a password. Each user may also have a private equivalence list in a file .rhosts in his login directory. Each line in this file should contain an *rhost* and a *username* separated by a space, giving additional cases where logins without passwords are to be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the .rhosts file must be owned by either the remote user or root.

The remote terminal type is the same as your local terminal type (as given in your environment TERM variable). The TERM value "wsiris" is converted to "rwsiris" when sent to the host. All echoing takes place at the remote site, so that (except for delays) the rlogin is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. The optional argument **−8** allows an eight-bit input data path at all times; otherwise parity bits are stripped except when the remote side's stop and start characters are other than ^S/^Q. The argument **−L** allows the rlogin session to be run in litout mode. A line of the form "~." disconnects from the remote host, where "~" is the escape character. A line starting with "\~ !" starts a shell on the IRIS. A different escape character may be specified by the **−e** option. There is no space separating this option flag and the argument character.

If D.B −e is given without a "~" character, no escapes can be done.

SEE ALSO

rsh(1C)

FILES

/usr/hosts/*	for *rhost* version of the command

ERRORS

More of the environment should be propagated.

ORIGIN

4.3 BSD

NAME

rls_id – generate release identification file

SYNOPSIS

rls_id [**−c**] *message* [*output-file*]

DESCRIPTION

Starting with MIPS compiler release 1.30, the compiler driver will use the value of the environment variable RLS_ID_OBJECT as the name of a file to link in as the last item in the link line (after all libraries). The intention is that this will be used to mark executable files with identification strings specific to a given release or set of executables.

The command **rls_id** is used to generate a file that contains strings that are understood by the RCS command *ident(1)* and the SCCS command *what(1)*. The strings are static, so they do not affect the running of the executable.

The message argument may contain any alphanumeric characters, space, tab, or punctuation with the exception of $, ", and > (RCS and SCCS control characters). Invalid messages will be rejected.

Unless **−c** is given, the data is compiled and placed in the file named by *output-file* or, if no *output-file* is given, the file named by the variable RLS_ID_OBJECT.

OPTIONS

−c Generate the C source on the standard output. This is useful when the compiler to be used to generate the executables is not the standard compiler (*/bin/cc* or */usr/bin/cc*).

EXAMPLES

The following three sets of commands will generate a file called **rls_id.o** with the identification string "UMIPS-BSD 3.0":

```
rls_id "UMIPS-BSD 3.0" rls_id.o
```

```
RLS_ID_OBJECT ="rls_id.o"
export RLS_ID_OBJECT
rls_id "UMIPS-BSD 3.0"
```

```
rls_id -c "UMIPS-BSDs+1 3.0" > rls_id.c
cc -c rls_id.c
```

Note that the second set uses Bourne Shell (*/bin/sh*) syntax and not that of the C-shell.

SEE ALSO

cc(1), ident(1), what(1).

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**−f**] [**−i**] file ...

rm −r [**−f**] [**−i**] dirname ... [file ...]

rmdir [**−p**] [**−s**] dirname ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **−f** option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to *rm*:

−f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

−r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **−f** option is used, or if the standard input is not a terminal and the **−i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **−f** option is used), resulting in an error message.

−i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **−f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

−p This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

−s This option is used to suppress the message printed on standard error when **−p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.

It is forbidden to remove the files "**.**" and "**..**" in order to avoid the consequences of inadvertently doing something like the following:

```
rm −r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2) in the *Programmer's Reference Manual*.

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel −rSID files

DESCRIPTION

rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* [see *get*(1)] exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

The **−r** option is used for specifying the *SID* (SCCS **ID**entification) level of the delta to be removed.

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.

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), prs(1), sccsfile(4).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rsh, remsh – remote shell

SYNOPSIS

/usr/net/rsh *host* [**−l** username] [**−n**] command
/usr/net/remsh *host* [rsh options]
/usr/hosts/*hostname* [rsh options]

DESCRIPTION

rsh connects to the specified *host,* and executes the specified *command*. *rsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *rsh* normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the **−l** option. This remote name must be equivalent (in the sense of *rlogin*(1C)) to the originating account; no provision is made for specifying a password with a command.

If you are using *csh*(1) and put a *rsh*(1C) in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command. If no input is desired you should redirect the input of *rsh* to /dev/null using the **−n** option.

If you omit *command,* then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1C).

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command

```
rsh otherhost cat remotefile >> localfile
```

appends the remote file *remotefile* to the localfile *localfile,* while

```
rsh otherhost cat remotefile ">>" otherremotefile
```

appends *remotefile* to *otherremotefile*.

Host names are given in the file */etc/hosts*. Each host has one standard name (the first name given in the file), which is rather long and unambiguous, and optionally one or more nicknames. The host names for local machines are also commands in the directory */usr/hosts*; if you put this directory in your search path then the **rsh** can be omitted.

FILES

/etc/hosts
/usr/hosts/*

SEE ALSO

rlogin(1C)

ERRORS

rsh does not return the exit status of the remote command.

You cannot run an interactive command (like *vi*(1)); use *rlogin*(1C).

Stop signals stop the local *rsh* process only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.

NAME

ruptime − show host status of local machines

SYNOPSIS

ruptime [**−a**]

DESCRIPTION

ruptime gives a status line for each machine on the local network. The status lines are formed from packets broadcast by each host on the network once a minute. They include a count of the number of users on a system and an indication of the load on each system, if available.

Machines for which no status report has been received for five minutes are shown as being down.

Users idle one hour or more are not counted unless the **−a** flag is given.

FILES

/usr/spool/rwho/whod.*	data files

SEE ALSO

rwho(1C), rwhod(1M)

ERRORS

Not all systems keep load statistics that are usable by *rwhod*(1M).

ORIGIN

4.3 BSD

NAME

rwall – write to all users over a network

SYNOPSIS

rwall *host1 host2 ...*
rwall −n *netgroup1 netgroup2 ...*
rwall −h *host* **−n** *netgroup*

DESCRIPTION

rwall reads a message from standard input until end-of-file. It then sends this message, preceded by the line "Broadcast Message ...", to all users logged in on the specified host machines. With the **-n** option, it sends to the specified network groups, which are defined in *netgroup(4)*.

A machine can only receive such a message if it is running *rwalld(1M)*, which is normally started up by the daemon *inetd(1M)*.

SEE ALSO

wall(1), netgroup(4), rwalld(1M), shutdown(1M)

ERRORS

The timeout is fairly short in order to be able to send to a large group of machines (some of which may be down) in a reasonable amount of time. Thus the message may not get through to a heavily loaded machine.

ORIGIN

Sun Microsystems

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [**−a**]

DESCRIPTION

The *rwho* command produces output similar to *who,* but for all machines on the local network. If no report has been received from a machine for 5 minutes then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user hasn't typed to the system for a minute or more, then *rwho* reports this idle time. If a user hasn't typed to the system for an hour or more, then the user will be omitted from the output of *rwho* unless the **−a** flag is given.

FILES

/usr/spool/rwho/whod.*	information about other machines

SEE ALSO

ruptime(1C), rwhod(1M)

ERRORS

This is unwieldy when the number of machines on the local net is large.

ORIGIN

4.3 BSD

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

sar – system activity reporter

SYNOPSIS

sar [**−ubdycwaqvmprDSA**] [**−o** file] t [n]

sar [**−ubdycwaqvmprDSA**] [**−s** time] [**−e** time] [**−i** sec] [**−f** file]

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **−o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, *sar* extracts data from a previously recorded *file,* either the one specified by **−f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd.* The starting and ending times of the report can be bounded via the **−s** and **−e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **−i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

−u Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **−D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

−b Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, i. e., (1−bread/lread) as a percentage;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.

−d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

−c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls. When used with **−D**, the system calls are split into incoming, outgoing,

and strictly local calls.

−w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s – process switches.

−a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

−q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

−v Report status of process, i-node, file tables:
text-sz, proc-sz, inod-sz, file-sz, lock-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

−m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

−p Report paging activities:
vflt/s – address translation page faults (valid page not in memory);
pflt/s – page faults from protection errors (illegal access to page) or "copy-on-writes";
pgfil/s – vflt/s satisfied by page-in from file system;
rclm/s – valid pages reclaimed for free list.

−r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for process swapping.

−D Report Remote File Sharing activity:
When used in combination with **−u** or **−c**, it causes **sar** to produce the remote file sharing version of the corresponding report. **−u** is assumed when neither **−u** or **−c** is specified.

−S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

−A Report all data. Equivalent to **−udqbwcayvmprSD**.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar −o temp 60 10
```

To later review disk and tape activity from that period:

```
        sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd*	daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G), sar(1M) in the *System Administrator's Reference Manual*.

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff **−r**SID1 **−r**SID2 [**−p**] [**−**sn] files

DESCRIPTION

sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

−r*SID?*	*SID1* and *SID0* 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

get(1).
bdiff(1), help(1), pr(1) in the *User's Reference Manual*.

DIAGNOSTICS

"*file*: No differences"   If the two versions are the same.
Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [**−a**] [file]

DESCRIPTION

script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **−a** option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

ERRORS

script places **everything** in the log file. This is not what the naive user expects.

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [options ...] file1 file2

DESCRIPTION

sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x   |   y
a       a
b   <
c   <
d       d
    >   c
```

OPTIONS

The following options exist:

−w *n*	Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
−l	Only print the left side of any lines that are identical.
−s	Do not print identical lines.
−o *output*	Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a **%** and waits for one of the following user-typed commands:
l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed – stream editor

SYNOPSIS

sed [**−n**] [**−e** script] [**−f** sfile] [files]

DESCRIPTION

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **−f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **−e** option and no **−f** options, the flag **−e** may be omitted. The **−n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **−n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a **/***regular expression***/** in the style of *ed*(1) modified thus:

In a context address, the construction \?*regular expression*?, where ? is any character, is identical to **/***regular expression***/**. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a new-line *embedded* in the pattern space.

A period **.** matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with **** to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) **a**
text　　Append. Place *text* on the output before reading the next input line.

(2) **b** *label*　　Branch to the **:** command bearing the *label*. If *label* is empty, branch to

the end of the script.

(2) **c**
text — Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) **d** — Delete the pattern space. Start the next cycle.

(2) **D** — Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) **g** — Replace the contents of the pattern space by the contents of the hold space.

(2) **G** — Append the contents of the hold space to the pattern space.

(2) **h** — Replace the contents of the hold space by the contents of the pattern space.

(2) **H** — Append the contents of the pattern space to the hold space.

(1) **i**
text — Insert. Place *text* on the standard output.

(2) **l** — List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) **n** — Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) **N** — Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) **p** — Print. Copy the pattern space to the standard output.

(2) **P** — Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) **q** — Quit. Branch to the end of the script. Do not start a new cycle.

(2) **r** *rfile* — Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) **s**/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of **/**. For a fuller description see *ed*(1). *Flags* is zero or more of:

n	n= 1 - 512. Substitute for just the n th occurrence of the *regular expression.*
g	Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
p	Print the pattern space if a replacement was made.

	w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.
(2) **t** *label*	Test. Branch to the **:** command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.
(2) **w** *wfile*	Write. Append the pattern space to *wfile*.
(2) **x**	Exchange the contents of the pattern and hold spaces.
(2) **y**/*string1*/*string2*/	Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.
(2)**!** *function*	Don't. Apply the *function* (or group, if *function* is **{**) only to lines *not* selected by the address(es).
(0) **:** *label*	This command does nothing; it bears a *label* for **b** and **t** commands to branch to.
(1) **=**	Place the current line number on the standard output as a line.
(2) **{**	Execute the following commands through a matching **}** only when the pattern space is selected.
(0)	An empty command is ignored.
(0) **#**	If a **#** appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the **#** is an 'n', then the default output will be suppressed. The rest of the line after **#**n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

setup – initialize system for first user

SYNOPSIS

setup

DESCRIPTION

The *setup* command, which is also accessible as a login by the same name, allows the first user to be established as the "owner" of the machine.

The user is permitted to add the first logins to the system, usually starting with his or her own.

The user can then protect the system from unauthorized modification of the machine configuration and software by giving passwords to the administrative and maintenance functions. Normally, the first user of the machine enters this command through the setup login, which initially has no password, and then gives passwords to the various functions in the system. Any that the user leaves without password protection can be exercised by anyone.

The user can then give passwords to system logins such as "root", "bin", etc. (*provided they do not already have passwords*). Once given a password, each login can only be changed by that login or "root".

The user can then set the date, time and time zone of the machine.

The user can then set the node name of the machine.

SEE ALSO

passwd(1).

DIAGNOSTICS

The *passwd*(1) command complains if the password provided does not meet its standards.

WARNING

If the setup login is not under password control, anyone can put passwords on the other functions.

NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

sh [**−acefhiknrstuvx**] [args]
rsh [**−acefhiknrstuvx**] [args]

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. *rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, −, $, and ! .

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by **;**, **&**, **&&**, or **||**, and optionally terminated by **;** or **&**. Of these four symbols, **;** and **&** have equal precedence, which is lower than that of **&&** and **||**. The symbols **&&** and **||** also have equal precedence. A semicolon (**;**) causes sequential execution of the preceding pipeline; an ampersand (**&**) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol **&&** (**||**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**
: Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. **NOTE:** You can substitute { and } for **do** and **done in for** statements, but not in **while** statements. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [|*pattern*] ...) *list* **;;**] ... **esac**
: A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**
: The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is

executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**
A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)
Execute *list* in a sub-shell.

{*list*;}
list is executed in the current (that is, parent) shell.

name () {*list*;}
Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments
A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution
The shell reads commands from the string between two grave accents (\`\`) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed. No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (\`) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...\` ...\` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact. If a backslash is used to escape a new-line character (**\new-line**), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (**\$**) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, \`, ", **new-line**, and $ are left intact when the command string is read.

Parameter Substitution
The character $ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] . . .

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter*}
The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or **@**, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero

when the shell is invoked.

${*parameter*:**−***word*}
: If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter*:**=***word*}
: If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter*:**?***word*}
: If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter*:**+***word*}
: If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

```
echo ${d:−`pwd`}
```

If the colon (**:**) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

#	The number of positional parameters in decimal.
−	Flags supplied to the shell on invocation or by the **set** command.
?	The decimal value returned by the last synchronously executed command.
$	The process number of this shell.
!	The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.

CDPATH
: The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
: This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
: A colon (**:**) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by **%** and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "**$** ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **new-line**.

SHACCT
: If this parameter is set to the name of a file writable by the user, the shell will

write an accounting record in the file for each shell procedure executed.

SHELL	When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or ´´) are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's 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 as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word　Use file *word* as standard input (file descriptor 0).

>word　Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word　Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[−]word　After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, − is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),
2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and
3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,
2) (escaped) **\new-line** is ignored, and
3) \ must be used to quote the characters \, **$**, and `.

The resulting document becomes the standard input.

<&digit　Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.

<&−　The standard input is closed. Similarly for the standard output using **>&−**.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
    ... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

. . . 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under "Commands," if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by **&** the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation
Before a command is executed, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character **.** at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched explicitly.

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the enclosed characters. A pair of characters separated by **–** matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "!" any character not enclosed is matched.

Quoting
The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a backslash (****) or inserting it between a pair of quote marks (′′ or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\new-line** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (′′), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, "′").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces (**"$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces (**"$1" "$2"** ...). **** quotes the characters ****, **`**, **"**, and **$**. The pair **\new-line** is removed before parameter and command substitution. If a backslash precedes characters other than ****, **`**, **"**, **$**, and new-line, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                              and
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the **−k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set −k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by **&**; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (**:**). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a **/** the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission, but sh fails when it tries to exec(2) it, the file is assumed to contain shell commmands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

Even though the shell will read and execute files containing shell commands, it is better to put "#! /bin/sh" as the very first line of the shell script to allow the system to execute the program automatically. See the exec(2) man page for more details.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

:　No effect; the command does nothing. A zero exit code is returned.

. *file*　Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
: Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
: Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
: Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (**:**). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a **/** the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rsh*.

echo [*arg* ...]
: Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]
: The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
: The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
: Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
: The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

getopts
: Use in shell scripts to support command syntax standards (see *intro*(1)); it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

hash [**−r**] [*name* ...]

For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]

Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]

One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **\new-line**. Characters other than **new-line** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]

The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]

Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [**−−aefhkntuvx** [*arg* ...]]

−a	Mark variables which are modified or created for export.
−e	Exit immediately if a command exits with a non-zero exit status.
−f	Disable file name generation
−h	Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
−k	All keyword arguments are placed in the environment for a command, not just those that precede the command name.
−n	Read commands but do not execute them.
−t	Exit after reading and executing one command.
−u	Treat unset variables as an error when substituting.
−v	Print shell input lines as they are read.
−x	Print commands and their arguments as they are executed.
−−	Do not change any of the flags; useful in setting **$1** to **−**.

Using **+** rather than **−** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$−**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]

The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test
: Evaluate conditional expressions. See *test*(1) for usage and description.

times
: Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
: The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
: For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
: Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user (see *su*(1M)) can raise a ulimit.

umask [*nnn*]
: The user file-creation mask is set to *nnn* (see *umask*(1)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
: For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
: Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is **−**, commands are initially read from */etc/profile* and from *$HOME/.profile*, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the **−c** or **−s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c *string* If the **−c** flag is present commands are read from *string*.

−s If the **−s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

−i If the **−i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

−r If the **−r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rsh Only

rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH,**
specifying path or command names containing **/**,
redirecting output (> and >>).

The restrictions above are enforced after *.profile* is interpreted.

A restricted shell can be invoked in one of the following ways: (1) *rsh* is the file name part of the last entry in the */etc/passwd* file (see *passwd*(4)); (2) the environment variable **SHELL** exists and *rsh* is the file name part of its value; (3) the shell is invoked and *rsh* is the file name part of argument 0; (4) the shell is invoke with the **−r** option.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the *.profile* (see *profile*(4)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), echo(1), env(1), getopts(1), intro(1), login(1), pwd(1), test(1), umask(1), wait(1).
dup(2), exec(2), fork(2), pipe(2), profile(4), signal(2), ulimit(2) in the *Programmer's Reference Manual*.

CAVEATS

Words used for filenames in input/output redirection are not interpreted for filename generation (see "File Name Generation," above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For *wait n*, if *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

size – prints the section size of an object file

SYNOPSIS

size [-o -d -x -A -B -V] [file1 ... fileN]

DESCRIPTION

The **size** command prints information about the *text, rdata, data, sdata, bss* and *sbss* sections of each file. The file can be an object or an archive. If you don't specify a file, **size** uses *a.out* as the default.

The **−o, −x,** and **−d** options print the size in octal, hexadecimal, and decimal, respectively.

The **−A** and **−B** options specify AT&T System V style output or Berkeley (4.3BSD) style output, respectively. The version of UNIX running at your site determines the default. System V style, which is more verbose than Berkeley, dumps the headers of each section. The Berkeley version prints size information for each section, regardless of whether the file exists, and prints the total in hexadecimal and decimal.

The **−V** option prints the version of **size** that you're using.

SEE ALSO

MIPS Languages Programmer Guide

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(3C) in the *Programmer's Reference Manual*.

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**−cmu**] [**−o**output] [**−y**kmem] [**−z**recsz] [**−dfiMnr**] [**−btx**] [**+**pos1 [**−**pos2]] [files]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **−** is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

−c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

−m Merge only, the input files are already sorted.

−u Unique: suppress all but one in each set of lines having equal keys.

−o*output* The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **−o** and *output.*

−y*kmem* The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding minimum or maximum will be used. Thus, **−y**0 is guaranteed to start with minimum memory. By convention, **−y** (with no argument) starts with maximum memory.

−z*recsz* The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **−c** or **−m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

−d "Dictionary" order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

−f Fold lower case letters into upper case.

−i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

−M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that "JAN" < "FEB" < . . . < "DEC". Invalid fields compare low to "JAN". The **−M** option implies the **−b** option (see below).

−n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **−n** option implies the **−b** option (see below). Note that the **−b** option is only effective when restricted sort key specifications are in effect.

−r Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation **+***pos1* **−***pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing **−***pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

−b Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **−b** option is specified before the first **+***pos1* argument, it will be applied to all **+***pos1* arguments. Otherwise, the **b** flag may be attached independently to each **+***pos1* or **−***pos2* argument (see below).

−t*x* Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m***.***n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by **+***m***.***n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing **.***n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; **+***m***.0b** refers to the first non-blank character in the *m*+1st field.

A last position specified by **−***m***.***n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing **.***n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; **−***m***.1b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 −2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort −r −o outfile +1.0 −1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort −r +1.0b −1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort −t: +2n −3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort −um +2 −3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the **−c** option. When the last line of an input file is missing a **new-line** character, *sort* appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−l**] [**+**local_file] [files]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with **=** for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **−l** option, *spell* will follow the chains of *all* included files.

Under the **+***local_file* option, words found in *local_file* are removed from *spell*'s output. *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see *FILES*). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy−y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake	Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.
spellin	Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), sed(1), sort(1), tee(1).
eqn(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH Software 2.0 Technical Discussion and Reference Manual.*

ERRORS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

NAME

split – split a file into pieces

SYNOPSIS

split [**−***n*] [file [name]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if **−** is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

starter – information about the UNIX system for beginning users

SYNOPSIS

[**help**] **starter**

DESCRIPTION

The UNIX system Help Facility command *starter* provides five categories of information about the UNIX system to assist new users.

The five categories are:

- commands a new user should learn first
- UNIX system documents important for beginners
- education centers offering UNIX system courses
- local environment information
- on-line teaching aids installed on the UNIX system

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit"). When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), usage(1).
term(5) in the *Programmer's Reference Manual.*

WARNINGS

If the environment variable TERM is not set, if defaults to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

strings – find the printable strings in an object or other binary file

SYNOPSIS

strings [**–**] [**–a**] [**–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 **–** or **–a** 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, core files, and many other things.

SEE ALSO

od(1)

ERRORS

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*.

FILES

/tmp/stm?	temporary file

SEE ALSO

ld(1)

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**−a**] [**−g**] [options]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., "**^h**" is **CTRL-h** ; in this case, recall that **CTRL-h** is the same as the "back-space" key.) The sequence "^" means that an option has a null value. For example, normally **stty −a** will report that the value of **swtch** is "^".

−a	reports all of the option settings;
−g	reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (−parenb)	enable (disable) parity generation and detection.
parodd (−parodd)	select odd (even) parity.
cs5 cs6 cs7 cs8	select character size (see *termio*(7)).
0	hang up phone line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
hupcl (−hupcl)	hang up (do not hang up) Dataphone connection on last close.
hup (−hup)	same as **hupcl (−hupcl)**.
cstopb (−cstopb)	use two (one) stop bits per character.
cread (−cread)	enable (disable) the receiver.
clocal (−clocal)	n assume a line without (with) modem control.
loblk (−loblk)	block (do not block) output from a non-current layer.

Input Modes

ignbrk (−ignbrk)	ignore (do not ignore) break on input.
brkint (−brkint)	signal (do not signal) INTR on break.
ignpar (−ignpar)	ignore (do not ignore) parity errors.
parmrk (−parmrk)	mark (do not mark) parity errors (see *termio*(7)).
inpck (−inpck)	enable (disable) input parity checking.
istrip (−istrip)	strip (do not strip) input characters to seven bits.
inlcr (−inlcr)	map (do not map) NL to CR on input.
igncr (−igncr)	ignore (do not ignore) CR on input.
icrnl (−icrnl)	map (do not map) CR to NL on input.
iuclc (−iuclc)	map (do not map) upper-case alphabetics to lower case on input.
ixon (−ixon)	enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
ixany (−ixany)	allow any character (only DC1) to restart output.
ixoff (−ixoff)	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (−opost)	post-process output (do not post-process output; ignore all other output modes).
olcuc (−olcuc)	map (do not map) lower-case alphabetics to upper case on output.
onlcr (−onlcr)	map (do not map) NL to CR-NL on output.
ocrnl (−ocrnl)	map (do not map) CR to NL on output.
onocr (−onocr)	do not (do) output CRs at column zero.
onlret (−onlret)	on the terminal NL performs (does not perform) the CR function.
ofill (−ofill)	use fill characters (use timing) for delays.
ofdel (−ofdel)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns (see *termio*(7)).
nl0 nl1	select style of delay for line-feeds (see *termio*(7)).
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs (see *termio*(7)).
bs0 bs1	select style of delay for backspaces (see *termio*(7)).
ff0 ff1	select style of delay for form-feeds (see *termio*(7)).
vt0 vt1	select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (−isig)	enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (−icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (−xcase)	canonical (unprocessed) upper/lower-case presentation.
echo (−echo)	echo back (do not echo back) every character typed.
echoe (−echoe)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (−echok)	echo (do not echo) NL after KILL character.
lfkc (−lfkc)	the same as **echok (−echok)**; obsolete.
echonl (−echonl)	echo (do not echo) NL.
noflsh (−noflsh)	disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (−stwrap)	disable (enable) truncation of lines longer than 79 characters on a synchronous line. (Does not apply to the 3B2.)
stflush (−stflush)	enable (disable) flush on a synchronous line after every *write*(2).*(Doesnotapply*to
stappl (−stappl)	use application mode (use line mode) on a synchronous line. (Does not apply to the 3B2.)

Control Assignments

control-character c	set *control-character* to *c*, where *control-character* is **erase, kill, intr, quit, swtch, eof, ctab, min,** or **time** (**ctab** is used with **−stappl; min** and **time** are used with **−icanon**; see *termio*(7)). If *c* is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "^**d**" is a **CTRL-d**); "^?" is interpreted as DEL and "^−" is interpreted as undefined.
line *i*	set line discipline to *i* ($0 < i < 127$).

Combination Modes

evenp or **parity**	enable **parenb** and **cs7**.
oddp	enable **parenb, cs7**, and **parodd.**
−parity, −evenp, or **−oddp**	disable **parenb**, and set **cs8**.

raw (**−raw** or **cooked**)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (**−nl**)	unset (set) **icrnl**, **onlcr**. In addition **−nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (**−lcase**)	set (unset) **xcase**, **iuclc**, and **olcuc**.
LCASE (**−LCASE**)	same as **lcase** (**−lcase**).
tabs (**−tabs** or **tab3**)	preserve (expand to spaces) tabs when printing.
ek	reset ERASE and KILL characters back to normal **#** and **@**.
sane	resets all modes to some reasonable values.
term	set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

ioctl(2) in the *Programmer's Reference Manual.*
termio(7) in the *System Administrator's Reference Manual.*

NAME

su, ssu – substitute user id temporarily

SYNOPSIS

su [**−f**] [**−**] [**−e**] [**−c**] [*userid* [*command* [*args...*]]]

DESCRIPTION

su demands the password of the specified *userid,* and if it is given, changes to that *userid* and invokes the shell (unless **−c** is given, see below) without changing the current directory. Unless the **−e** option is given, the SHELL for the substituted user is used instead of invoking the user's shell. The new user ID stays in force until the Shell exits.

If no *userid* is specified, "root" is assumed. Only users in the "root" group (group 0) or in the file */etc/su_people* (described below) can *su* to "root", even with the root password (this can be overridden by changing *su* to have group root and turning on the set-group-id permission). To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

The command *ssu* is a link to *su*. Executing *ssu* is the same as executing the command 'su -c -e root'.

If the user tries to su to root and the root account has a password (as is the preferable case), the file */etc/su_people* is read to see if that username is allowed to become root without a password. Since this can be dangerous, the file must have owner 0 (root), group root (0), and mode 0600 (read and write by owner only), or it will be silently ignored. See the manual page for *su_people(5)* for details on this file.

OPTIONS

−f	Prevents *csh*(1) from executing the .cshrc file; thus making *su* start up faster.
−	Simulates a full login by executing the shell with name '-sh'.
−e	Do not overwrite any of the environment. This means that the variables HOME and SHELL are retained from the original user and that shell is executed. For *csh(1)* users, this means that the aliases are taken from the original user's .cshrc file, which is very convenient.
−c	If any arguments are given after the username, they are executed as a command instead of the shell. For example, 'su -c root ls' will execute the command *ls(1)* as root, whereas 'su root ls' will execute the command 'csh ls' as root (this is not the same thing, as it says to execute with the file as input).

FILES

/etc/su_people	Special permission database

SEE ALSO

sh(1), csh(1), su_people(5)

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**−r**] *file*

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

The option **−r** causes an alternate algorithm to be used in computing the checksum. This alternate algorithm is compatable with BSD-based systems.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

sysadm – menu interface to do system administration

SYNOPSIS

sysadm [*sub-command*]

DESCRIPTION

This command, when invoked without an argument, presents a menu of system administration sub-commands, from which the user selects. If the optional argument is presented, the named sub-command is run or the named sub-menu is presented.

The *sysadm* command may be given a password. See **admpasswd** in the SUBCOMMANDS section.

SUB-COMMANDS

The following menus of sub-commands are available. (The number of bullets (●) in front of each item indicates the level of the menu or subcommand.)

● diagnostics　　system diagnostics menu

These subcommands look for and sometimes repair problems in the system. Those subcommands that issue reports allow you to determine if there are detectable problems. Commands that attempt repair are for repair people only. You must know what you are doing!

● ● diskrepair

advice on repair of built-in disk errors

This subcommand advises you on how to go about repairing errors that occur on built-in disks.

WARNING: Because this is a repair function, it should only be performed by qualified service personnel.
NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

● ● diskreport

report on built-in disk errors

This subcommand shows you if the system has collected any information indicating that there have been errors while reading the built-in disks. You can request either summary or full reports. The summary report provides sufficient information about disk errors to determine if repair should be attempted. If the message no errors logged is part of the report, then there is probably no damage. If a number of errors is reported, there is damage and you should call for service. The full report gives additional detail for the expert repair person trouble shooting complicated problems.

NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

● diskmgmt
disk management menu

The subcommands in this menu provide functions for using removable disks. The subcommands include the ability to format disks, copy disks, and to use disks as mountable file systems. It also contains a menu of subcommands for handling non-removable media.

● ● checkfsys
check a removable disk file system for errors

Checkfsys checks a file system on a removable disk for errors. If there are errors, this procedure attempts to repair them.

● ● cpdisk
make exact copies of a removable disk

This procedure copies the contents of a removable disk into the machine and then allows the user to make exact copies of it. These copies are identical to the original in every way. The copies are made by first reading the original removable disk entirely into the machine and then writing it out onto duplicate disks. The procedure will fail if there is not enough space in the system to hold the original disk.

● ● erase
erase data from removable disk

This procedure erases a removable disk by overwriting it with null bytes. The main purpose is to remove data that the user does not want seen. Once performed, this operation is irreversible.

● ● format
format new removable disks

Format prepares new removable disks for use. Once formatted, programs and data can be written on the disks.

● ● harddisk
hard disk management menu

The subcommands in this menu provide functions for using hard disks. For each hard disk, the disk can be partitioned with default partitioning or the current disk partitioning can be displayed.

● ● ● display
display hard disk partitioning

Display will allow the user to display the hard disk partitioning. This will inform the user of current disk partitioning information.

● ● ● partitioning
partition a hard disk

Partitioning configures hard disks. This will allow you to partition a hard disk according to the default partitioning.

● ● ● rmdisk
remove a hard disk

Removes a hard disk from the system configuration. It may then be physically disconnected (once the machine has been turned off) or freshly partitioned (after the machine has been restarted).

● ● makefsys
create a new file system on a removable disk

Makefsys creates a new file system on a removable disk which can then store data which the user does not wish to keep on the hard disk. When "mounted", the file system has all the properties of a file kept on the hard disk, except that it is smaller.

● ● mountfsys
mount a removable disk file system

Mountfsys mounts a file system, found on a removable disk, making it available to the user. The file system is unmounted with the "umountfsys" command. THE DISK MUST NOT BE REMOVED WHILE THE FILE SYSTEM IS STILL MOUNTED.
IF THE FILE SYSTEM HAS BEEN MOUNTED WITH THE **mountfsys** COMMAND, IT MUST BE UNMOUNTED WITH **umountfsys**.

● ● umountfsys
unmount a removable disk file system

Umountfsys unmounts a file system, allowing the user to remove the disk. THE DISK MUST NOT BE REMOVED UNTIL THE FILE SYSTEM IS UNMOUNTED.
umountfsys MAY ONLY BE USED TO UNMOUNT FILE SYSTEMS MOUNTED WITH THE **mountfsys** COMMAND.

● filemgmt
file management menu

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them onto diskettes and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine which files might be best kept on diskette based on age or size.

● ● backup
backup files from integral hard disk to removable disk or tape

Backup saves copies of files from the integral hard disk file systems to removable disk or tape. There are two kinds of backups:

COMPLETE – copies all files (useful in case of serious file system damage)

INCREMENTAL – copies files changed since the last backup

The normal usage is to do a complete backup of each file system and then periodically do incremental backups. Two cycles are recommended (one set of complete backups and several incrementals to each cycle). Files backed up with "backup" are restored using "restore".

● ● bupsched
backup reminder scheduling menu

Backup scheduling is used to schedule backup reminder messages and backup reminder checks. Backup reminder messages are sent to the console to remind the administrator to backup particular file systems when the machine is shutdown or a reminder check has been run during the specified time period.

Backup reminder checks specify particular times at which the system will check to see if any backup reminder messages have been scheduled.

● ● ● schedcheck
schedule backup reminder checks

Backup reminder checks are run at specific times to check to see if any reminders are scheduled. The user specifies the times at which the check is to be run. Checks are run for the reminder messages scheduled by *schedmsg*.

● ● ● schedmsg
schedule backup reminder message

Backup reminder messages are sent to the console if the machine is shutdown or a reminder check has been scheduled. The user specifies the times at which it is appropriate to send a message and the file systems to be included in the message.

● ● diskuse
display how much of the hard disk is being used

Diskuse lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

● ● fileage
list files older than a particular date

Fileage prints the names of all files older than the date specified by the user. If no date is entered, all files older than 90 days will be listed. If no directory is specified to look in, the **/usr/admin** directory will be used.

● ● filesize
list the largest files in a particular directory

Filesize prints the names of the largest files in a specific directory. If no directory is specified, the **/usr/admin** directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

● ● restore
restore files from "backup" and "store" media to integral hard disk

Restore copies files from disks and tapes made by "backup" and "store" back onto the hard disk. You can restore individual files, directories of files, or the entire contents of a disk or tape. The user can restore from both "incremental" and "complete" media. The user can also list the names of files stored on the disk or tape.

● ● store
store files and directories of files onto disk or tape

Store copies files from the integral hard disk to disk or tape and allows the user to optionally verify that they worked and to optionally remove them when done. Typically, these would be files that the user wants to archive or restrict access to. The user can store single files and directories of files. Use the "restore" command to put stored files back on the integral hard disk and to list the files stored.

● machinemgmt
machine management menu

Machine management functions are tools used to operate the machine, e.g., turn it off, reboot, or go to the firmware monitor.

● ● powerdown
stop all running programs, then turn off the machine

Powerdown will stop all running programs, close any open files, write out information to disk (such as directory information), then turn the machine power off.

● ● reboot
stop all running programs then reboot the machine

Reboot will stop all running programs, close any open files, write out information to disk (such as directory information), then reboot the machine. This can be used to get out of some types of system trouble, such as when a process cannot be killed.

● ● whoson
print list of users currently logged onto the system

Whoson prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

● syssetup
system setup menu

System setup routines allow the user to tell the computer what its environment looks like: what the date, time, and time zone is, what administration and system capabilities are to be under password control, what the machine's name is, etc. The first-time setup sequence is also here.

● ● admpasswd
assign or change administrative passwords

Admpasswd lets you set or make changes to passwords for administrative commands and logins such as setup and sysadm.

● ● datetime
set the date, time, time zone, and daylight savings time

Datetime tells the computer the date, time, time zone, and whether you observe Daylight Savings Time (DST). It is normally run once when the machine is first set up. If you observe DST, the computer will automatically start to observe it in the spring and return to Standard Time in the fall. The machine has to be turned off and turned back on again to guarantee that ALL times will be reported correctly. Most are correct the next time the user logs in.

● ● nodename
set the node name of this machine

This allows you to change the node name of this machine. The node name is used by various communications networks to identify this machine.

● ● setup
set up your machine the very first time

Setup allows the user to define the first login, to set the passwords on the user-definable administration logins and to set the time zone for your location.

● ● syspasswd
assign system passwords

Syspasswd lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the "root" login.

● ttymgmt
terminal management

This procedure allows the user to manage the computer's terminal functions.

● ● lineset
show tty line settings and hunt sequences

The tty line settings are often hunt sequences where, if the first line setting does not work, the line "hunts" to the next line setting until one that does work comes by. This subcommand shows the various sequences with only specific line settings in them. It also shows each line setting in detail.

● ● mklineset
create new tyy line settings and hunt sequences

This subcommand helps you to create tty line setting entries. You might want to add line settings that are not in the current set or create hunt sequences with only specific line settings in them. The created hunt sequences are circular; stepping past the last setting puts you on the first.

● ● modtty
show and optionally modify characteristics of tty lines

This subcommand reports and allows you to change the characteristics of tty lines (also called "ports").

● usermgmt
user management menu

These subcommands allow you to add, modify and delete the list of users that have access to your machine. You can also place them in separate groups so that they can share access to files within the group but protect themselves from other groups.

● ● addgroup
add a group to the system

Addgroup adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

● ● adduser
add a user to the system

Adduser installs a new login ID on the machine. You are asked a series of questions about the user and then the new entry is made. You can enter more than one user at a time. Once this procedure is finished, the new login ID is available.

● ● delgroup
delete a group from the system

Delgroup allows you to remove groups from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

● ● deluser
delete a user from the system

Deluser allows you to remove users from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the **/etc/passwd** file.

● ● lsgroup
list groups in the system

Lsgroup will list all the groups that have been entered into the computer. This list is updated automatically by "addgroup" and "delgroup"

● ● lsuser
list users in the system

Lsuser will list all the users that have been entered into the computer. This list is updated automatically by "adduser" and "deluser".

● ● modadduser
modify defaults used by adduser

Modadduser allows the user to change some of the defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

● ● modgroup
make changes to a group on the system

Modgroup allows the user to change the name of a group that the user enters when "addgroup" is run to set up new groups.

● ● moduser
menu of commands to modify a user's login

This menu contains commands that modify the various aspects of a user's login.

● ● ● chgloginid
change a user's login ID

This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

● ● ● chgpasswd
change a user's passwd

This proceudure allows removal or change of a suer's password. Administrative and system login passwords channot be changed. To change administrative and system login passwords, see the system setup menu: sysadm syssetup.

● ● ● chgshell
change a user's login shell

This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

EXAMPLES

sysadm adduser

FILES

The files that support *sysadm* are found in **/usr/admin**.

The menu starts in directory **/usr/admin/menu**.

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [±[number][**lbc**[**f**]]] [file]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance **+***number* from the beginning, or **−***number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **−f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail −f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail −15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1M).

ERRORS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 4096 bytes of a file regardless of its line count.

NAME

tar – tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

tar saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). *tar*'s 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 *tar* are file or directory names specifying which files to dump or restore. 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. (**NOTE:** the **r** function will not write files on the end of most cartridge tapes and some half-inch tapes. The nature of some devices makes it nearly impossible to seek back over individual records, so there is no way to find the end of the *tar* archive.)
d	This option causes directory entries to be placed on the tape, which means that directory premissions and ownerships are preserved. The resulting tar archive can cause warning messages if extracted on standard System III or System V systems, but will still work.
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.

o	Ownership. This causes extracted files to take on the user and group identifier of the user running the program, rather than those on tape. This is only valid with the x key.
p	This modifier says to restore files to their original modes, ignoring the present *umask*(2). Setuid and sticky information will also be restored to the super-user.
0, ..., 9	This modifier selects an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally /dev/rmt/m0.
v	Normally *tar* does its work silently. The **v** (verbose) option makes *tar* print the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about

the tape entries than just their names.

w　*tar* prints 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 done. Any other input means don't do it.

n　*tar* uses the next argument as the name of a file which contains the list of file or directory names. If the name of the file is '–', tar reads from standard input. This option works both for creation (option **c**) and for extraction (option **x**).

f　*tar* uses the next argument as the name of the archive instead of /dev/rmt/m?. 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　*tar* uses the next argument as the blocking factor for tape records. The default is 20. This option should only be used with raw magnetic tape archives (See **f** above). The block size may be determined automatically when reading tapes (key letters 'x' and 't'), but only if the block size on the tape is no more than the block size specified with **b** (or the default).

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* not to restore the modification times. The modification time will be the time of extraction.

h　Force *tar* to follow symbolic links as if they were normal files or directories. Normally, *tar* does not follow symbolic links.

B　Forces input and output blocking to 20 blocks per record. This option was added so that *tar* can work across a communications channel where the blocking may not be maintained.

C　If a file name is preceded by **−C**, then *tar* will perform a *chdir*(2) to that file name. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from /usr/include and from /etc, one might use

```
tar c -C /usr include -C / etc
```

i　Ignore directory checksum errors. The errors are reported, but are not fatal.

F　If one **F** is given, all directories named "SCCS" and "RCS", and files named "core" and "errs" are ignored. If more than one **F** is given, files named "a.out" and all files with the suffix ".o" are also ignored. This is useful for creating archives of source trees.

Previous restrictions dealing with *tar*'s inability to properly handle blocked archives have been lifted.

FILES

/dev/rmt/m?
/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.

ERRORS

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 current limit on file name length is 100 characters.
There is no way selectively to follow symbolic links.
When extracting tapes created with the **r** or **u** options, directory modification times may not be set correctly.

NAME

tee – pipe fitting

SYNOPSIS

tee [**−i**] [**−a**] [file] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*. The

−i ignore interrupts;

−a causes the output to be appended to the *files* rather than overwriting them.

NAME

telnet – user interface to the TELNET protocol

SYNOPSIS

telnet [host [port]]

DESCRIPTION

telnet is used to communicate with another host using the **TELNET** protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, *telnet* enters an input mode. The input mode entered will be either "character at a time" or "line by line" depending on what the remote system supports.

In "character at a time" mode, most text typed is immediately sent to the remote host for processing.

In "line by line" mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The "local echo character" (initially "^E") may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the *localchars* toggle is TRUE (the default in line mode; see below), the user's *quit*, *intr*, and *flush* characters are trapped locally, and sent as **TELNET** protocol sequences to the remote side. There are options (see **toggle** *autoflush* and **toggle** *autosynch* below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the **TELNET** sequence) and flush previous terminal input (in the case of *quit* and *intr*).

While connected to a remote host, *telnet* command mode may be entered by typing the *telnet* "escape character" (initially "^]"). When in command mode, the normal terminal editing conventions are available.

COMMANDS

The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the **mode**, **set**, **toggle**, and **display** commands).

open *host* [*port*]

Open a connection to the named host. If no port number is specified, *telnet* will attempt to contact a **TELNET** server at the default port. The host specification may be either a host name (see *hosts*(4)) or an Internet address specified in the "dot notation" (see *inet*(3N)).

close

Close a **TELNET** session and return to command mode.

quit

Close any open **TELNET** session and exit *telnet*. An end of file (in command mode) will also close a session and exit.

z

Suspend *telnet*. This command only works when the user is using the *csh*(1).

mode *type*

type is either *line* (for "line by line" mode) or *character* (for "character at a time" mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be entered.

status

Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode.

display [*argument...*]

Displays all, or some, of the **set** and **toggle** values (see below).

? [*command*]

Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just that command.

send *arguments*

Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

escape

Sends the current *telnet* escape character (initially "^]").

synch

Sends the **TELNET SYNCH** sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case "r" may be echoed on the terminal).

brk

Sends the **TELNET BRK** (Break) sequence, which may have significance to the remote system.

ip

Sends the **TELNET IP** (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

ao

Sends the **TELNET AO** (Abort Output) sequence, which should cause the remote system to flush all output **from** the remote system **to** the user's terminal.

ayt

Sends the **TELNET AYT** (Are You There) sequence, to which the remote system may or may not choose to respond.

ec

Sends the **TELNET EC** (Erase Character) sequence, which should cause the remote system to erase the last character entered.

el

Sends the **TELNET EL** (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

ga
: Sends the **TELNET GA** (Go Ahead) sequence, which likely has no significance to the remote system.

nop
: Sends the **TELNET NOP** (No OPeration) sequence.

?
: Prints out help information for the **send** command.

set *argument value*

Set any one of a number of *telnet* variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the **display** command. The variables which may be specified are:

echo
: This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape
: This is the *telnet* escape character (initially "^[") which causes entry into *telnet* command mode (when connected to a remote system).

interrupt
: If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *interrupt* character is typed, a **TELNET IP** sequence (see **send** *ip* above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's **intr** character.

quit
: If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *quit* character is typed, a **TELNET BRK** sequence (see **send** *brk* Above) Is sent to the remote host. The initial value for the quit character is taken to be the terminal's **quit** character.

flushoutput
: If *telnet* is in *localchars* mode (see **toggle** *localchars* below) and the *flushoutput* character is typed, a **TELNET AO** sequence (see **send** *ao* above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's **flush** character.

erase
: If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in "character at a time" mode, then when this character is typed, a **TELNET EC** sequence (see **send** *ec* above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's **erase** character.

kill
: If *telnet* is in *localchars* mode (see **toggle** *localchars* below), **and** if *telnet* is operating in "character at a time"

mode, then when this character is typed, a **TELNET EL** sequence (see **send** *el* above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's **kill** character.

eof

If *telnet* is operating in "line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's **eof** character.

toggle *arguments...*

Toggle (between TRUE and FALSE) various flags that control how *telnet* responds to events. More than one argument may be specified. The state of these flags may be interrogated with the **display** command. Valid arguments are:

localchars

If this is TRUE, then the *flush*, *interrupt*, *quit*, *erase*, and *kill* characters (see **set** above) are recognized locally, and transformed into (hopefully) appropriate **TELNET** control sequences (respectively *ao*, *ip*, *brk*, *ec*, and *el*; see **send** above). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

autoflush

If *autoflush* and *localchars* are both TRUE , then when the *ao*, *intr*, or *quit* characters are recognized (and transformed into **TELNET** sequences; see **set** above for details), *telnet* refuses to display any data on the user's terminal until the remote system acknowledges (via a **TELNET** *timing Mark* option) that it has processed those **TELNET** sequences. The initial value for this toggle is TRUE if the terminal user had not done an "stty noflsh", otherwise FALSE (see *stty*(1)).

autosynch

If *autosynch* and *localchars* are both TRUE , then when either the *intr* or *quit* characters is typed (see **set** above for descriptions of the *intr* and *quit* characters), the resulting **TELNET** sequence sent is followed by the **TELNET SYNCH** sequence. This procedure **should** cause the remote system to begin throwing away all previously typed input until both of the **TELNET** sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod

Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug
: Toggles socket level debugging (useful only to the *super*user). The initial value for this toggle is FALSE.

options
: Toggles the display of some internal *telnet* protocol processing (having to do with **TELNET** options). The initial value for this toggle is FALSE.

netdata
: Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

?
: Displays the legal **toggle** commands.

ERRORS

There is no adequate way for dealing with flow control.

On some remote systems, echo has to be turned off manually when in "line by line" mode.

There is enough settable state to justify a *.telnetrc* file.

No capability for a *.telnetrc* file is provided.

In "line by line" mode, the terminal's *eof* character is only recognized (and sent to the remote system) when it is the first character on a line.

ORIGIN

4.3 BSD

NAME

test – condition evaluation command

SYNOPSIS

test expr
[expr]

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, a non-zero (false) exit status is set; *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS line) must be separate arguments to the *test* command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

−r *file*	true if *file* exists and is readable.
−w *file*	true if *file* exists and is writable.
−x *file*	true if *file* exists and is executable.
−f *file*	true if *file* exists and is a regular file.
−d *file*	true if *file* exists and is a directory.
−c *file*	true if *file* exists and is a character special file.
−b *file*	true if *file* exists and is a block special file.
−p *file*	true if *file* exists and is a named pipe (fifo).
−u *file*	true if *file* exists and its set-user-ID bit is set.
−g *file*	true if *file* exists and its set-group-ID bit is set.
−k *file*	true if *file* exists and its sticky bit is set.
−s *file*	true if *file* exists and has a size greater than zero.
−t [*fildes*]	true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
file1 **-nt** *file2*	true if *file1* is newer (has a greater modification time) than *file2*.
file1 **-ot** *file2*	true if *file1* is older (has a smaller modification time) than *file2*.
−z *s1*	true if the length of string *s1* is zero.
−n *s1*	true if the length of the string *s1* is non-zero.
s1 **=** *s2*	true if strings *s1* and *s2* are identical.
s1 **!=** *s2*	true if strings *s1* and *s2* are *not* identical.
s1	true if *s1* is *not* the null string.
n1 **−eq** *n2*	true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!	unary negation operator.
−a	binary *and* operator.
−o	binary *or* operator (**−a** has higher precedence than **−o**).
(expr)	parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the *-r*, *-w*, or *-x* tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are super-user.

The **=** and **!=** operators have a higher precedence than the **−r** through **−n** operators, and **=** and **!=** always expect arguments; therefore, **=** and **!=** cannot be used with the **−r** through **−n** operators.

If more than one argument follows the **−r** through **−n** operators, only the first argument is examined; the others are ignored, unless a **−a** or a **−o** is the second argument.

NAME

tftp – trivial file transfer program

SYNOPSIS

tftp [host]

DESCRIPTION

tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote *host* may be specified on the command line, in which case *tftp* uses *host* as the default host for future transfers (see the **connect** command below).

An account or password on the remote machine is not required. Due to lack of authentication information, *tftp* is only able to access publicly readable files. Search permissions of directories leading to accessed files are not checked.

COMMANDS

Once *tftp* is running, it issues the prompt **tftp>** and recognizes the following commands:

connect *host-name* [*port*]	Set the *host* (and optionally *port*) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections betweeen transfers; thus, the *connect* command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the *connect* command; the remote host can be specified as part of the *get* or *put* commands.
mode *transfer-mode*	Set the mode for transfers; *transfer-mode* may be one of *ascii*, *binary*, or *mail*. The default is *ascii*.
put *file ... destination*	Put a file or set of files to the specified *destination*. *destination* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host:filename* to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers.
get *source ... file*	Get a file or set of files from the specified *sources*. *source* can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form *host : filename* to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.
quit	Exit *tftp*.
verbose	Toggle verbose mode. Has no effect.
trace	Toggle packet tracing.
status	Show current status.
rexmt *retransmission-timeout*	Set the per-packet retransmission timeout, in seconds.
timeout *total-transmission-timeout*	Set the total transmission timeout, in seconds.
? [*command-name* ...]	Print help information.

ERRORS

Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

The *verbose* command has no effect.

NAME

time − time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

times(2) in the *Programmer's Reference Manual.*

NAME

timex − time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported. The output of *timex* is written on standard error. *Options* are:

−p List process accounting records for *command* and all its children. Suboptions **f, h, k, m, r,** and **t** modify the data items reported. The options are as follows:

- **−f** Print the *fork/exec* flag and system exit status columns in the output.
- **−h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

  (total CPU time)/(elapsed time).
- **−k** Instead of memory size, show total kcore-minutes.
- **−m** Show mean core size (the default).
- **−r** Show CPU factor (user time/(system-time + user-time).
- **−t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

−o Report the total number of blocks read or written and total characters transferred by *command* and all its children.

−s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

sar(1).

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
        timex −ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
    timex −opskmt sh
        session commands
    EOT
```

NAME

touch – update access and modification times of a file

SYNOPSIS

touch [**−amc**] [mmddhhmm[yy]] files

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **−a** and **−m** options cause touch to update only the access or modification times respectively (default is **−am**). The **−c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).
utime(2) in the *Programmer's Reference Manual*.

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

tput [**−T**type] capname [parms ...]

tput [**−T**type] **init**

tput [**−T**type] **reset**

tput [**−T**type] **longname**

DESCRIPTION

tput uses the *terminfo*(4) database to make the values of terminal-dependent capabilities and information available to the shell (see *sh*(1)), to initialize or reset the terminal, or return the long name of the requested terminal type. *tput* outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see *sh*(1)) to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS** below.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

−T*type*	indicates the *type* of terminal. Normally this option is unnecessary, because the default is taken from the environment variable **TERM.** If **−T** is specified, then the shell variables **LINES** and **COLUMNS** and the layer size will not be referenced.
capname	indicates the attribute from the *terminfo*(4) database.
parms	If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.
init	If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **−T***type,* above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.
reset	Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname　If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **−T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) database (see *term*(5)).

EXAMPLES

tput init　Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command should be included in everyone's .profile after the environmental variable **TERM** has been exported, as illustrated on the *profile*(4) manual page.

tput −Twyse60 reset
　Reset a Wyse 60 terminal, overriding the type of terminal in the environmental variable **TERM**.

tput cup 0 0　Send the sequence to move the cursor to row **0**, column **0** (the upper left corner of the screen, usually known as the "home" cursor position).

tput clear　Echo the clear-screen sequence for the current terminal.

tput cols　Print the number of columns for the current terminal.

tput -T450 cols　Print the number of columns for the 450 terminal.

bold=‘tput smso‘

offbold=‘tput rmso‘
　Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt:
　echo "${bold}Please type in your name: ${offbold}\c"

tput hc　Set exit code to indicate if the current terminal is a hardcopy terminal.

tput cup 23 4　Send the sequence to move the cursor to row 23, column 4.

tput longname　Print the long name from the *terminfo*(4) database for the type of terminal specified in the environmental variable **TERM**.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/curses.h	*curses*(3X) header file
/usr/include/term.h	*terminfo*(4) header file
/usr/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of *terminfo*(4)

SEE ALSO

stty (1)
profile(4), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 10 of the *Programmer's Guide.*

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE. If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a null value is returned on standard output).

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **−1** means that *capname* is not defined for this terminal *type*. Any other exit code indicates an error; see **DIAGNOSTICS**, below.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

exit code	error message
0	−1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) database for this terminal type, e.g. **tput −T450 lines** and **tput −T2621 xmc**)
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) database
4	unknown *terminfo*(4) capability *capname*

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*.

OPTIONS

Any combination of the options **−cds** may be used:

−c	Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
−d	Deletes all input characters in *string1*.
−s	Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a−z]	Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
[a∗*n*]	Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr −cs "[A−Z][a−z]" "[\012∗]" <file1 >file2
```

SEE ALSO

ed(1), sh(1).
ascii(5) in the *Programmer's Reference Manual*.

ERRORS

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

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 - BSD-derived terminal setup program

SYNOPSIS

tset [*options*]

DESCRIPTION

This command is intended as a mechanism for setting the TERM and TERMCAP environment variables, and initializing the terminal state. Because of various bugs and incompatabilities between the ideas of termcap and terminfo, it does the job rather poorly.

The UMIPS-V 4.0 release should have a version of **tset** that works properly. New users are requested to use the command *tput(1)* instead of attempting to use **tset.**

SEE ALSO

tput(1).

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

The *tsort* command 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.

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [**−l**] [**−s**]

DESCRIPTION

tty prints the path name of the user's terminal.

−l prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.

−s inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

2 if invalid otions were specified,

0 if standard input is a terminal,

1 otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and **−l** is specified.

"not a tty" if the standard input is not a terminal and **−s** is not specified.

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)

ERRORS

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

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user's **.profile** (see *profile*(4)) and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), sh(1).
chmod(2), creat(2), umask(2), profile(4) in the *Programmer's Reference Manual.*

NAME

uname − obtain current system information

SYNOPSIS

name [**−a**] [**−A**] [**−s**] [**−n**] [**−r**] [**−v**] [**−m**] [**−t**] [**−b**]

DESCRIPTION

uname prints the requested pieces of the current system information on the standard output, separated by spaces. The information is obtained from the system call *uname(2)*. See that manual page for the specific field descriptions. By default, the **sysname** field value is printed. The values are printed in the following order: **sysname**, **nodename**, **release**, **version**, **machine**, **m_type**, and **base_rel**.

Note that the fields **sysname**, **nodename**, **release**, **version**, and **machine** are considered to be standard fields. The others are MIPS -specific fields.

The valid options are:

−a	Print all of the standard values available.
−A	Print all of the values, including the Mips-specific values.
−m	Print the value of the **machine** field.
−n	Print the value of the **nodename** field.
−r	Print the value of the **release** field.
−s	Print the value of the **sysname** field.
−v	Print the value of the **version** field.
−t	(Mips-specific) Print the value of the **m_type** field.
−b	(Mips-specific) Print the value of the **base_rel** field.

SEE ALSO

uname(2)

ERRORS

The field values in this version of *uname* do not contain spaces or tabs. This is not guaranteed to be the case in every system that has a *uname* command.

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [**−r**SID] [**−s**] [**−n**] files

DESCRIPTION

unget undoes the effect of a **get −e** done prior to creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of **−** is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. Keyletter arguments apply independently to each named file.

−r*SID* Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

−s Suppresses the printout, on the standard output, of the intended delta's *SID*.

−n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), sact(1).
help(1) in the *User's Reference Manual*.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**−udc** [**+**n] [**−**n]] [input [output]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have:  inch
You want:  cm
        * 2.540000e+00
        / 3.937008e–01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have:  15 lbs force/in2
You want:  atm
        * 1.020689e+00
        / 9.797299e–01
```

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

Pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed thus: **brgallon.** For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

uptime - display system up time and load statistics

SYNOPSIS

uptime [**−l**] [**−s**] [**−n**] [**−f** *format*]

DESCRIPTION

The *uptime* command displays statistics on how long the system has been up, how many users are on the system, and three types of load averages: standard (exponentially decaying for 1, 5, and 15 minutes), extended (exponentially decaying for 1, 5, and 20 seconds plus the standard load average), and rolling average (for 1, 5, and 15 minutes).

The default format of output looks similar to the BSD UNIX command of the same name. The following shows how this might look:

```
10:33am  up 16:50,  17 users,  load average: 1.03, 0.50, 0.00
```

Command-line options provide similar formats using the various types of available load averages. In addition, the format can be defined by the user by using the **−f** option or the UPTIME_FORMAT environment variable.

The format string can contain any text. The sequences \n, \t, \f, \r, \b, and \\ are handled just like C escapes. All other escaped characters are printed as-is. Statistics are printed by using %-specifiers, as in the *date(1)* command. The available sequences are:

t	Current time as HH:MM on a 12 hour clock
T	Current time as HH:MM on a 24 hour clock
r	"am" or "pm"
R	"AM" or "PM"
u	Number of users with idle time less than an hour
U	Number of users on the system
d	Number of days the system has been up
h	Number of hours the system has been up not including days
H	Number of hours the system has been up including days
m	Number of minutes the system has been up not including days and hours
M	Number of minutes the system has been up including days and hours
D	Date sentence (hh:mm or d day(s), hh:mm)
p	Plurality – 's' if last %-specifier used was not exactly 1
a	Exponentially decaying load average as: x.xx, x.xx, x.xx
A	Extended load average as: x.xx, x.xx, x.xx, x.xx, x.xx, x.xx
1a	Exponentially decaying load average over last 1 second
2a	Exponentially decaying load average over last 5 seconds
3a	Exponentially decaying load average over last 20 seconds
4a	Exponentially decaying load average over last 60 seconds
5a	Exponentially decaying load average over last 300 seconds
6a	Exponentially decaying load average over last 900 seconds
s	Rolling average as: x.xx, x.xx, x.xx
1s	Rolling average over last 60 seconds
2s	Rolling average over last 300 seconds
3s	Rolling average over last 900 seconds
%	The character %

The specifiers %[456]s are interpreted as %s. The sequences %[1-6][^as] cause nothing to be printed.

The default format is given by the following string:

```
%t%r up %D, %U user%p, load average: %a
```

Any format option (**−l**, **−s**, **−n**, or **−f**,) given overrides the UPTIME_FORMAT variable.

OPTIONS

−l	Use the standard format, but display all exponentially decaying load averages (%A) instead of just the oldest 3 (%a).
−s	Use the standard format, but display rolling average (%s) instead of exponentially decaying averages (%a).
−n	Use the default format (see above). This should be used in all shell script applications that need to use the default format, since it overrides the UPTIME_FORMAT environment variable.
−f *format*	Use the format given by *format* to print the statistics. The format string should be quoted in the shell to avoid interpretation by the shell.

FILES

/unix	Operating system executable
/dev/kmem	Operating system memory image

SEE ALSO

ruptime(1c).

NAME

usage – retrieve a command description and usage examples

SYNOPSIS

[**help**] **usage** [**−d**] [**−e**] [**−o**] [command_name]

DESCRIPTION

The UNIX system Help Facility command *usage* retrieves information about UNIX system commands. With no argument, *usage* displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by *usage*. The user may also exit to the shell by typing q (for "quit).

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to *usage*. To receive information on the command's description, examples, or options, the user may use the **−d**, **−e**, or **−o** options respectively. (The default option is **−d**.)

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see *profile*(4)): "export SCROLL ; SCROLL=no". If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (*starter*, *locate*, *usage*, *glossary*, and *help*) is located on their respective manual pages.

SEE ALSO

glossary(1), help(1), locate(1), sh(1), starter(1).
term(5) in the *Programmer's Reference Manual*.

WARNINGS

If the shell variable **TERM** (see *sh*(1)) is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [options] source-files destination-file
uulog [options] **−s**system
uulog [options] system
uulog [options] **−f**system
uuname [**−l**] [**−c**]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names that *uucp* knows about. The *system-name* may also be a list of names such as

system-name!system-name!...!system-name!path-name

in which case an attempt is made to send the file via the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to foward information (see WARNINGS below for restrictions).

The shell metacharacters ?, ∗ and [...] appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

(1) a full path name;

(2) a path name preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory;

(3) a path name preceded by *~/destination* where *destination* is appended to **/usr/spool/uucppublic**; (NOTE: This destination will be treated as a file name unless more than one file is being transfered by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory).

(4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

−c	Do not copy local file to the spool directory for transfer to the remote machine (default).
−C	Force the copy of local files to the spool directory for transfer.
−d	Make all necessary directories for the file copy (default).
−f	Do not make intermediate directories for the file copy.
−g*grade*	*Grade* is a single letter/number; lower ascii sequence characters will cause the job

to be transmitted earlier during a particular conversation.

−j Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

−m Send mail to the requester when the copy is completed.

−n*user* Notify *user* on the remote system that a file was sent.

−r Do not start the file transfer, just queue the job.

−s*file* Report status of the transfer to *file*. Note that the *file* must be a full path name.

−x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog

uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/***system*, or **/usr/spool/uucp/.Log/uuxqt/***system*.

The options cause *uulog* to print logging information:

−s*sys* Print information about file transfer work involving system *sys*.

−f*system* Does a "tail **−f**" of the file transfer log for *system*. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:

−x Look in the *uuxqt* log file for the given system.

−*number* Indicates that a "tail" command of *number* lines should be executed.

uuname

uuname lists the names of systems known to *uucp*. The **−c** option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different *Systems* files for *cu* and *uucp*. See the *Sysfiles* file.) The **−l** option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending **(/usr/spool/uucppublic)**
/usr/lib/uucp/*	other data and program files

SEE ALSO

mail(1), uustat(1C), uux(1C), uuxqt(1M).
chmod(2) in the *Programmer's Reference Manual*.

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to ~/).

All files received by *uucp* will be owned by *uucp*.
The **−m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the **−m** option. The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

ERRORS

Protected files and files that are in protected directories that are owned by the requestor can be sent by *uucp*. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uuencode,uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [*source_file*] *dest_file*
uudecode [*filename*]

DESCRIPTION

uuencode and *uudecode* are used to send a binary file via *uucp* (or other) mail. This combination can be used over indirect mail links.

uuencode takes the named source file (or reads from standard input if none is given) and produces an encoded version on the standard output. The encoding uses only printable ASCII characters, and includes the mode of the file and the name of the destination file for recreation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The encoded file can be edited with any ordinary text editor to change the mode or destination filename. See *uuencode(4)* for details of the format.

The intended use of *uuencode* is in the pipeline:

uuencode *source_file dest_file* | **mail** *system*!...!**decode**

The user *decode* is expected to be set up on the remote end such that the mail is piped through *uudecode*. For sites using *sendmail(1)*, this can be done in the system mail aliases file.

In any event, an encoded file can be sent directly to a user to be decoded manually.

SEE ALSO

mail(1), uucp(1), uuencode(4), sendmail(1M).

ERRORS

The file is expanded by 35% (3 bytes become 4 plus control information), causing it to take longer to transmit.

The user invoking *uudecode* must have write permission for the destination file.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [**−a**]
uustat [**−m**]
uustat [**−p**]
uustat [**−q**]
uustat [**−k***jobid*]
uustat [**−r***jobid*]
uustat [**−s***system*] [**−u***user*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

−a Output all jobs in queue.

−m Report the status of accessibility of all machines.

−p Execute a "ps −flp" for all the process-ids that are in the lock files.

−q List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. NOTE: for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **−q** option:

```
eagle     3C    04/07-11:07   NO DEVICES AVAILABLE
mh3bs3    2C    07/07-10:42   SUCCESSFUL
```

The above output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

−k*jobid*
Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

−r*jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat*:

−s*sys*	Report the status of all *uucp* requests for remote system *sys*.
−u*user*	Report the status of all *uucp* requests issued by *user*.

Output for both the **−s** and **−u** options has the following format:

```
eaglen0000  4/07-11:01:03     (POLL)
eagleN1bd7 4/07-11:07         Seagledan522 /usr/dan/A
eagleC1bd8 4/07-11:07         Seagledan59 D.3b2al2ce4924
                  4/07-11:07  Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/*	spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick – public UNIX-to-UNIX system 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*). *User* is the login name of someone on the specified system.

Two *options* are available:

−p	Copy the source file into the spool directory before transmission.
−m	Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

PUBDIR/receive/*user*/*mysystem*/files.

The destined recipient is notified by *mail*(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*. If *dir* is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
a [*dir*]	Same as **m** except moving all the files sent from *system*.
p	Print the content of the file.
q	Stop.
EOT (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), uustat(1C), uux(1C).
uucleanup(1M) in the *System Administrator's Reference Manual.*

WARNINGS

In order to send files that begin with a dot (e.g., .profile) the files must by qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [options] command-string

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permiting only the receipt of mail (see *mail*(1)). (Remote execution permissions are defined in **/usr/lib/uucp/Permissions**.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

(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.

As an example, the command

```
uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 > !~/dan/file.diff"
```

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a *diff*(1) command and put the results in *file.diff* in the local PUBDIR/dan/ directory.

Any special shell characters such as **<>;|** should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!cut -fR b!/usr/file \(c!/usr/file\)
```

gets /usr/file from system "b" and sends it to system "a", performs a *cut* command on that file and sends the result of the *cut* command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **−n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

−	The standard input to *uux* is made the standard input to the *command-string*.
−a*name*	Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)
−b	Return whatever standard input was provided to the *uux* command if the exit status is non-zero.
−c	Do not copy local file to the spool directory for transfer to the remote machine (default).
−C	Force the copy of local files to the spool directory for transfer.
−g*grade*	*Grade* is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

−j	Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.
−n	Do not notify the user if the command fails.
−p	Same as −: The standard input to *uux* is made the standard input to the *command-string*.
−r	Do not start the file transfer, just queue the job.
−s*file*	Report status of the transfer in *file*.
−x*debug_level*	Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.
−z	Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will NOT work:

```
uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If *diff* is a permitted command.)

ERRORS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using *uux*. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.

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., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.
–mname	The argument value *name* is compared with the s-1SCCS %M% keyword in *file*.
–ytype	The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **–y** mismatch;
bit 7 = %M%, **–m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Use *help*(1) for explanations.

ERRORS

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.
−cchar	Specifies a control character to be used in place of **:**.
−s	Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition
.
.
.
:end

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.
The syntax of a condition is:

```
<cond>        ::= [ "not" ] <or>
<or>          ::= <and> | <and> "|" <or>
<and>         ::= <exp> | <exp> "&" <and>
<exp>         ::= "(" <or> ")" | <value> <op> <value>
<op>          ::= "=" | "!=" | "<" | ">"
<value>       ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <     all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text
: Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the **−a** keyletter.

:on

:off
: Turn on or off keyword replacement on all lines.

:ctl char
: Change the control character to char.

:msg message
: Prints the given message on the diagnostic output.

:err message
: Prints the given message followed by:

  ERROR: err statement on line **...** (915)

  on the diagnostic output. *vc* halts execution, and returns an exit code of 1.

SEE ALSO

ed(1), help(1) in the *User's Reference Manual.*

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 [**−t** *tag*] [**−r** *file*] [**−w***n*] [**−R**] [**−x**] [**+***command*] *name* ...
view [**-t** *tag*] [**−r** *file*] [**−w***n*] [**−R**] [**−x**] [**+***command*] *name*
vedit [**−t** *tag*] [**−r** *file*] [**−w***n*] [**−R**] [**−x**] [**+***command*] *name*

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by *vi*:

−t *tag*	Edit the file containing the *tag* and position the editor at its definition.
−r*file*	Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
−w*n*	Set the default window size to *n*. This is useful when using the editor over a slow speed line.
−R	Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
+*command*	The specified *ex* command is interpreted before editing begins.
−x	Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see *crypt*(1)). Also, see the **WARNING** section at the end of this manual page.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command	Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
Input	Entered by the following options **a i A I o O c C s S R**. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for **: / ?** or **!**; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text*ESC	insert text *abc*
cw*new*ESC	change word to *new*
ea*s*ESC	pluralize word
x	delete a character
dw	delete a word
dd	delete a line
3dd	... 3 lines
u	undo previous change
ZZ	exit vi, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*ex cmd***CR**	any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
DEL	(delete or rubout) interrupts
^L	reprint screen if DEL scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
	synonym for **:e #**

:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:!*cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	to tag file entry *tag*
^]	**:ta**, following word is *tag*

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a **CR**.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/pat	next line matching *pat*
?pat	prev line matching *pat*
n	repeat last **/** or **?**
N	reverse last **/** or **?**
/pat/+n	nth line after *pat*
?pat?−n	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) **{** or **}**

Adjusting the screen

^L	clear and redraw
^R	retype, eliminate @ lines
zCR	redraw, current at window top
z−CR	... at bottom
z .CR	... at center
*/pat/*z−**CR**	*pat* line at bottom
z*n* **.CR**	use *n* line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

\`\`	move cursor to previous context
′′	... at first non-white in line
m*x*	mark current position with letter *x*
\`*x*	move cursor to mark *x*
′*x*	... at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white
0	beginning of line
$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
f*x*	find *x* forward
F*x*	**f** backward
t*x*	upto *x* forward
T*x*	back upto *x*
;	repeat last **f F t** or **T**
,	inverse of ;
|	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(	back sentence
{	back paragraph
W	blank delimited word
B	back **W**
E	to end of **W**

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as **^H**
kill	your kill, erase input this line
\	quotes **^H**, your erase and kill
ESC	ends insertion, back to command
DEL	interrupt, terminates insert
^D	backtab over *autoindent*
↑^D	kill *autoindent*, save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
=	indent for LISP

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	... before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"*x*p	put from buffer *x*
"*x*y	yank to buffer *x*
"*x*d	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d* p	retrieve *d*'th last delete

AUTHOR

vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

/usr/lib/.COREterm/?/* subset of compiled terminal description database, supplied on hard disk d

SEE ALSO

ed(1), edit(1), ex(1).
User's Guide.
Editing Guide.

WARNING

The **−x** option is provided with the Security Administration Utilities, which is available only in the United States.

Tampering with entries in */usr/lib/.COREterm/?/** or */usr/lib/terminfo/?/** (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

ERRORS

Software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

vmsbackup – read a VMS backup tape

SYNOPSIS

vmsbackup −{tx}[cdevw][s setnumber][f tapefile] [name ...]

DESCRIPTION

vmsbackup reads a VMS generated backup tape, converting the files to UNIX format and writing the files to disc. The default operation of the program is to go through an entire tape, extracting every file and writing it to disc. This may be modified by the following options.

c Use complete filenames, including the version number. A colon and the octal version number will be appended to all filenames. A colon, rather than a semicolon, is used since the UNIX Shell uses the semicolon as the line separator. Using a colon prevents the user from having to escape the semicolon when referencing the filename. This option is useful only when multiple versions of the same file are on a single tape or when a file of the same name already exists in the destination directory. The default is to ignore version numbers.

d use the directory structure from VMS, the default value is off.

e Process all filename extensions. Since this program is mainly intended to move source code and possibly data from a DEC system to a UNIX system, the default is to ignore all files whose filename extension specifies system dependent data. The file types which will be ignored, unless the **e** option is specified, are

exe	VMS executable file
lib	VMS object library file
obj	RSX object file
odl	RSX overlay description file
olb	RSX object library file
pmd	RSX post mortem dump file
stb	RSX task symbol table file
sys	RSX bootable system file
tsk	RSX executable task file

f Use the next argument in the command line as the tape device to be used, rather than the default.

The optional *user* portion of the pathname specifies the login name to use on the remote system. If it is not supplied, the current user's login name will be used. In all the cases, the user must have the appropriate permissions on the remote machine, in order to use this facility. The default is */dev/rmt8* (drive 0, raw mode, 1600 bpi). This must be a raw mode tape device.

s saveset Process only the given saveset number.

t Produce a table of contents (a directory listing) on the standard output of the files on tape.

v Verbose output. Normally *vmsbackup* does its work silently. The verbose option will cause the filenames of the files being read from tape to disk to be output on the standard output.

w *vmsbackup* prints 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 done. Any other input means don't do it.

x	extract the named files from the tape.
The optional	*name* argument specifies one or more filenames to be searched for specifically on the tape and only those files are to be processed. The name may contain the usal sh(1) meta-characters *?![] 0n.

FILES

/dev/rmt*x*

ERRORS

The filename match uses the complete VMS file names.

NAME

vsar – visual system activity reporter

SYNOPSIS

vsar [**−ubdycwaqvmprA**] [**−i** *sec*] [**−D** *num*] [**−V**] [**−S**]

DESCRIPTION

vsar samples cumulative activity counters in the operating system at intervals of *t* seconds. The default value of *t* is 1. The **−i** option selects records at *sec* second intervals. The **−D** option limits the number of block devices that will be displayed. For example it is possible that more devices could be configured into the kernel than are actually attached to the machine. In order to avoid displaying information on devices not actually present, this option should be used. The **−V** option displays a running average for any value being monitored. The **−S** option displays a single page containing all the options listed below.

In a multi-page display typeing the keys **j,k,n,p** will cause pagnation. The key **q** will always terminate the program.

Subsets of data to be printed are specified by option:

−u Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

−b Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, i. e., (1−bread/lread) as a percentage;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.

−d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dkip-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

−y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

−c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls.

−w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);

	pswch/s – process switches.
−a	Report use of file access system routines: *iget/s, namei/s, dirblk/s.*
−q	Report average queue length while occupied, and % of time occupied: *runq-sz, %runocc* – run queue of processes in memory and runnable; *swpq-sz, %swpocc* – swap queue of processes swapped out but ready to run.
−v	Report status of process, i-node, file tables: *text-sz, proc-sz, inod-sz, file-sz, lock-sz* – entries/size for each table, evaluated once at sampling point; *ov* – overflows that occur between sampling points for each table.
−m	Report message and semaphore activities: *msg/s, sema/s* – primitives per second.
−p	Report paging activities: *vflt/s* – address translation page faults (valid page not in memory); *pflt/s* – page faults from protection errors (illegal access to page) or "copy-on-writes"; *pgfil/s* – *vflt/s* satisfied by page-in from file system; *rclm/s* – valid pages reclaimed for free list.
−r	Report unused memory pages and disk blocks: *freemem* – average pages available to user processes; *freeswap* – disk blocks available for process swapping.
−A	Report all data. Equivalent to **−udqbwcayvmpr**.

SEE ALSO

sar(1), sar(1M) in the *System Administrator's Reference Manual.*

NAME

wait – await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

ERRORS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wait – wait for a process to terminate

SYNOPSIS

integer function wait (status)
integer status

DESCRIPTION

Wait causes its caller to be suspended until a signal is received or one of its child processes terminates. If any child has terminated since the last **wait,** return is immediate; if there are no children, return is immediate with an error code.

If the returned value is positive, it is the process ID of the child and **status** is its termination status (see *wait(2)*). If the returned value is negative, it is the negation of a system error code.

FILES

/usr/lib/libU77.a

SEE ALSO

wait(2), signal(3F), kill(3F), perror(3F)

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 to all currently logged-in users preceded by:

Broadcast Message from **. . .**

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's tty file fails.

NAME

wc – word count

SYNOPSIS

wc [**−lwc**] [names]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **−lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what – identify SCCS files

SYNOPSIS

what [−s] files

DESCRIPTION

what searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ˜, >, new-line, \, or null character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command

```
what f.c f.o a.out
```

will print

```
f.c:
                identification information
f.o:
                identification information
a.out:
                identification information
```

what is intended to be used in conjunction with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

−s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1).
help(1) in the *User's Reference Manual.*

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

ERRORS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

NAME

who – who is on the system

SYNOPSIS

who [**−uTlHqpdbrtas**] [file]

who am i

who am I

DESCRIPTION

who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the **/etc/utmp** file at login time to obtain its information. If *file* is given, that file (which must be in *utmp*[4] format) is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [state] line time [idle] [pid] [comment] [exit]

The *name*, *line*, and *time* information is produced by all options except **−q**; the *state* information is produced only by **−T**; the *idle* and *pid* information is produced only by **−u** and **−l**; and the *comment* and *exit* information is produced only by **−a**. The information produced for **−p**, **−d**, and **−r** is explained during the discussion of each option, below.

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process.

OPTIONS

−u　This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (**.**) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*[4]). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

−T　This option is the same as the **−s** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A **+** appears if the terminal is writable by anyone; a **−** appears if it is not. **root** can write to all lines having a **+** or a **−** in the *state* field. If a bad line is encountered, a ? is printed.

−l　This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

−H　This option will print column headings above the regular output.

−q　This is a quick *who,* displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

−p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

−d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*[2]), of the dead process. This can be useful in determining why a process terminated.

−b This option indicates the time and date of the last reboot.

−r This option indicates the current *run-level* of the *init* process. In addition, it produces the process termination status, process id, and process exit status (see *utmp*(4)) under the *idle*, *pid*, and *comment* headings, respectively.

−t This option indicates the last change to the system clock (via the *date*[1] command) by **root**. See *su*(1).

−a This option processes **/etc/utmp** or the named *file* with all options turned on.

−s This option is the default and lists only the *name*, *line*, and *time* fields.

Note to the super-user: after a shutdown to the single-user state, *who* returns a prompt; the reason is that since **/etc/utmp** is updated at login time and there is no login in single-user state, *who* cannot report accurately on this state. *who am i*, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), su(1M).
init(1M) in the *System Administrator's Reference Manual*.
wait(2), inittab(4), utmp(4) in the *Programmer's Reference Manual*.

NAME

winsize – set/print window size

SYNOPSIS

winsize [**−c**] [**−r**] [*value*...]

DESCRIPTION

With no arguments, *winsize* prints the current number of rows and columns associated with the terminal. With two *values*, the number of rows and columns is set.

Giving the **−c** option without a *value* prints the number of columns. With a *value*, the number of columns is set.

Giving the **−r** option without a *value* prints the number of rows. With a *value*, the number of rows is set.

A *value* may either be a number or the word "same", which indicates that the item should not change. Note that a value of 0 for either rows or columns indicates that programs will use the value found in the *terminfo(4)* entry, not that that dimension has a value of 0.

Giving both **−c** and **−r** is the same as giving neither.

Changing the number of rows and columns does not necessarily change the size of the associated window, and should not be done frivolously.

EXAMPLES

SEE ALSO

ioctl(2), terminfo(4), termio(7).

NAME

write – write to another user

SYNOPSIS

write user [line]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

> **Message from** *yourname* **(tty??)** [*date*]**...**

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

> *user* is logged on more than one place.
> You are connected to "*terminal*".
> Other locations are:
> *terminal*

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, such as *pr*(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute **!**

SEE ALSO

mail(1), mesg(1), pr(1), sh(1), who(1).

DIAGNOSTICS

"*user is not logged on*" if the person you are trying to *write* to is not logged on.
"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).
"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.
"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **−i** flag). Flags **−i**, **−l**, and **−n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **−l** vs. **−n**), the last flag has precedence. *Flag* values are:

−l*number* *command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **−x** is forced.

−i*replstr* Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **−x** is also forced. **{ }** is assumed for *replstr* if not specified.

−n*number* Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **−x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

−t Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

−p Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (**−t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If **−s** is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

−e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1.  ls | xargs -p -l ar r arch
2.  ls | xargs -p -l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(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*	Temp file when 'xstr name' doesn't touch *strings*

SEE ALSO

mkstr(1)

ERRORS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**−vdlt**] grammar

DESCRIPTION

The *yacc* command converts a context-free grammar into a set of tables for a simple automaton which executes a 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.

If the **−l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **−t** option is used, this debugging code will be compiled by default. Independent of whether the **−t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	
yacc.debug, yacc.acts	temporary files
/usr/lib/yaccparparser	prototype for C programs

SEE ALSO

lex(1).
Programmer's Guide.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

CAVEAT

Because file names are fixed, at most one *yacc* process can be active in a given directory at a given time.