RISC/os (UMIPS) User's Reference Manual Volume II (BSD)

Order Number 3204DOC

mips

The power of RISC is in the system.

RISC/os (UMIPS)
User's Reference Manual
Volume II (BSD)

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

PERMUTED INDEX

terminal worms	: animate worms on a display	worms(6)
hangman hangman	: Computer version of the game	hangman(6)
worm	: Play the growing worm game	worm(6)
game hunt	: a multi-player multi-terminal	hunt(6)
battlestar	: a tropical adventure game	battlestar(6)
adventure	: an exploration game	adventure(6)
error messages error	: analyze and disperse compiler	error(1)
messages a.error	: analyze and disperse error	a.error(1)
rain	: animated raindrops display	rain(6)
arguments apply	: apply a command to a set of	apply(1)
yes	: be repetitively affirmative	yes(1)
bibliography, indxbib, lookbib	: build inverted index for a	lookbib(1)
cat	: catenate and print	cat(1)
chgrp	: change group	chgrp(1)
chmod	: change mode	chmod(1)
information chfn, chsh, passwd	: change password file	passwd(1)
checknr	: check nroff troff files	checknr(1)
compress, uncompress, zcat	: compress and expand data	compress(1)
tip, cu	: connect to a remote system	tip(1c)
units	: conversion program	units(1)
English number	: convert Arabic numerals to	number(6)
dd	: convert and copy a file	dd(1)
bcd	: convert to antique media	bcd(6)
cp	: copy	cp(1)
tcopy	: copy a mag tape	tcopy(1)
a.tags	: create a tags file	a.tags(1)
database addbib	: create or extend bibliographic	raddbib(1)
whatis	: describe what a command is	whatis(1)
contains a unit a.which	: determine which project library	a.which(1)
and create a default a.vadsrc	: display available VADS versions	a.vadsrc(1)
snake, snscore	: display chase game	snake(6)
on the target board cross a.run	: download and execute a program	a.run(1)
graph	: draw a graph	graph(1g)
input soelim	: eliminate .so's from nroff	soelim(1)
crypt	: encode decode	crypt(1)
and history mechanism for a.view	: establish command abbreviations	a.view(1)
time at	: execute commands at a later	at(1)
versa expand, unexpand	: expand tabs to spaces, and vice	expand(1)
robots	: fight off villainous robots	robots(6)
previewing colcrt	: filter nroff output for CRT	colcrt(1)
col	: filter reverse line feeds	col(1)
references in documents refer	: find and insert literature	refer(1)
find	: find files	find(1)
look	: find lines in a sorted list	look(1)
keywords; print out the man	: find manual information by	man(1)
spell, spellin, spellout	: find spelling errors	spell(1)
a.pr	: format source code	a.pr(1)
troff tbl	: format tables for nroff or	tbl(1)
subsystem sccs	: front end for the SCCS	sccs(1)
tty	: get terminal name	tty(1)
head	: give first few lines	head(1)
source indent	: indent and format C program	indent(1)
and teletypes last	: indicate last logins of users	last(1)
a.help	: interactive help utility	a.help(1)
spline	: interpolate smooth curve	spline(1g)
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)
lookup apropos	: locate commands by keyword	apropos(1)
manual for program whereis	: locate source, binary, and or	whereis(1)
regenerate groups make,makerules	: maintain, update, and	make(1)
a.mklib	: make library directory	a.mklib(1)
ln	: make links	ln(1)
monop	: monopoly game	monop(6)
iron men sail	: multi-user wooden ships and	sail(6)
od	: octal, decimal, hex, ascii dump	od(1)
4014 tk	: paginator for the Tektronix	tk(1)
language awk	: pattern scanning and processing	awk(1)
mesg	: permit or deny messages	mesg(1)
ptx	: permuted index	ptx(1)
tc	: photoypesetter simulator	tc(1)

	chgrp : change group	chgrp(1)
	chmod : change mode	chmod(1)
file information chfn,	chsh, passwd : change password	passwd(1)
a.pr : format source	code	a.pr(1)
	col : filter reverse line feeds	col(1)
CRT previewing	colcrt : filter nroff output for	colcrt(1)
file	colrm : remove columns from a	colrm(1)
colrm : remove	columns from a file	colrm(1)
mechanism for a.view : establish	command abbreviations and history	a.view(1)
whatis : describe what a	command is	whatis(1)
apply : apply a	command to a set of arguments	apply(1)
at : execute	commands at a later time	at(1)
apropos : locate	commands by keyword lookup	apropos(1)
order lastcomm : show last	commands executed in reverse	lastcomm(1)
a.rmlib : remove	compilation library	a.rmlib(1)
a.ls : list	compiled programs	a.ls(1)
error : analyze and disperse	compiler error messages	error(1)
program on the target board cross	compilers download and execute a	a.run(1)
compress, uncompress, zcat :	compress and expand data	compress(1)
compress and expand data	compress, uncompress, zcat :	compress(1)
tip, cu :	connect to a remote system	tip(1c)
remove nroff, troff, tbl and eqn	constructs deroff :	deroff(1)
: determine which project library	contains a unit a.which	a.which(1)
ls : list	contents of directory	ls(1)
units :	conversion program	units(1)
English number :	convert Arabic numerals to	number(6)
dd :	convert and copy a file	dd(1)
bcd :	convert to antique media	bcd(6)
cp :	copy	cp(1)
dd : convert and	copy a file	dd(1)
tcopy :	copy a mag tape	tcopy(1)
	cp : copy	cp(1)
available VADS versions and	create a default library display	a.vadsrc(1)
a.tags :	create a tags file	a.tags(1)
database addbib :	create or extend bibliographic	raddbib(1)
cribbage : the card game	cribbage	cribbage(6)
	cribbage : the card game cribbage	cribbage(6)
a program on the target board	cross compilers and execute	a.run(1)
	crypt : encode decode	crypt(1)
tip,	cu : connect to a remote system	tip(1c)
spline : interpolate smooth	curve	spline(1g)
zcat : compress and expand	data compress, uncompress,	compress(1)
: create or extend bibliographic	database addbib	raddbib(1)
roffbib : run off bibliographic	database	roffbib(1)
sortbib : sort bibliographic	database	sortbib(1)
date : print and set the	date	date(1)
	date : print and set the date	date(1)
touch : update	date last modified of a file	touch(1)
	dd : convert and copy a file	dd(1)
a.db : source level	debugger	a.db(1)
od : octal,	decimal, hex, ascii dump	od(1)
crypt : encode	decode	crypt(1)
VADS versions and create a	default library available	a.vadsrc(1)
mesg : permit or	deny messages	mesg(1)
: recompile source files in	dependency order a.make	a.make(1)
and eqn constructs	deroff : remove nroff, troff, tbl	deroff(1)
whatis :	describe what a command is	whatis(1)
contains a unit a.which :	determine which project library	a.which(1)
ratfor : rational Fortran	dialect	ratfor(1)
wordy sentences; thesaurus for	diction diction, explain : print	diction(1)
sentences; thesaurus for diction	diction, explain : print wordy	diction(1)
rmdir : remove (unlink) files or	directories rm,	rm(1)
a.cleanlib : reinitialize library	directory	a.cleanlib(1)
a.mklib : make library	directory	a.mklib(1)
ls : list contents of	directory	ls(1)
a.du : summarize library	disk usage	a.du(1)
du : summarize	disk usage	du(1)
error : analyze and	disperse compiler error messages	error(1)
a.error : analyze and	disperse error messages	a.error(1)
rain : animated raindrops	display	rain(6)
and create a default a.vadsrc :	display available VADS versions	a.vadsrc(1)
snake, snscore :	display chase game	snake(6)
worms : animate worms on a	display terminal	worms(6)
insert literature references in	documents refer : find and	refer(1)
w : who is on and what they are	doing	w(1)
the target board cross a.run :	download and execute a program on	a.run(1)

graph :	draw a graph	graph(1g)
arithmetic : provide	drill in number facts	arithmetic(6)
	du : summarize disk usage	du(1)
od : octal, decimal, hex, ascii	dump	od(1)
zork : the game of	dungeon	zork(6)
input soelim :	eliminate .so's from nroff	soelim(1)
crypt :	encode decode	crypt(1)
sccs : front	end for the SCCS subsystem	sccs(1)
xsend, xget,	enroll : secret mail	send(1)
window : window	environment	window(1)
: remove nroff, troff, tbl and	eqn constructs deroff	deroff(1)
mathematics	eqn, neqn, checkeq : typeset	eqn(1)
compiler error messages	error : analyze and disperse	error(1)
a.error : analyze and disperse	error messages	a.error(1)
: analyze and disperse compiler	error messages error	error(1)
spellin, spellout : find spelling	errors spell,	spell(1)
and history mechanism a.view :	establish command abbreviations	a.view(1)
lpq : spool queue	examination program	lpq(1)
board cross a.run : download and	execute a program on the target	a.run(1)
at :	execute commands at a later time	at(1)
lastcomm : show last commands	executed in reverse order	lastcomm(1)
uncompress, zcat : compress and	expand data compress,	compress(1)
versa expand, unexpand :	expand tabs to spaces, and vice	expand(1)
spaces, and vice versa	expand, unexpand : expand tabs to	expand(1)
thesaurus for diction diction,	explain : print wordy sentences;	diction(1)
adventure : an	exploration game	adventure(6)
addbib : create or	extend bibliographic database	raddbib(1)
kill : terminate a process with	extreme prejudice	kill(1)
: provide drill in number	facts arithmetic	arithmetic(6)
col : filter reverse line	feeds	col(1)
head : give first	few lines	head(1)
robots :	fight off villainous robots	robots(6)
a.tags : create a tags	file	a.tags(1)
colrm : remove columns from a	file	colrm(1)
dd : convert and copy a	file	dd(1)
pr : print	file	pr(1)
rev : reverse lines of a	file	rev(1)
: update date last modified of a	file touch	touch(1)
uniq : report repeated lines in a	file	uniq(1)
chsh, passwd : change password	file information chfn,	passwd(1)
: split a multi-routine Fortran	file into individual files	fsplit(1)
checknr : check nroff troff	files	checknr(1)
find : find	files	find(1)
Fortran file into individual	files : split a multi-routine	fsplit(1)
sort : sort or merge	files	sort(1)
a.make : recompile source	files in dependency order	a.make(1)
rm, rmdir : remove (unlink)	files or directories	rm(1)
previewing colcrt :	filter nroff output for CRT	colcrt(1)
col :	filter reverse line feeds	col(1)
	find : find files	find(1)
references in documents refer :	find and insert literature	refer(1)
find :	find files	find(1)
look :	find lines in a sorted list	look(1)
keywords; print out the man :	find manual information by	man(1)
index for a bibliography,	find references in a bibliography	lookbib(1)
spell, spellin, spellout :	find spelling errors	spell(1)
fish : play "Go	fish"	fish(6)
	fish : play "Go fish"	fish(6)
	fmt : simple text formatter	fmt(1)
indent : indent and	format C program source	indent(1)
a.pr :	format source code	a.pr(1)
tbl :	format tables for nroff or troff	tbl(1)
fmt : simple text	formatter	fmt(1)
nroff : text	formatting	nroff(1)
troff, nroff : text	formatting and typesetting	troff(1)
hopefully interesting, adage	fortune : print a random,	fortune(6)
from : who is my mail	from?	from(1)
sccs :	front end for the SCCS subsystem	sccs(1)
Fortran file into individual	fsplit : split a multi-routine	fsplit(1)
adventure : an exploration	game	adventure(6)
backgammon : the	game	backgammon(6)
battlestar : a tropical adventure	game	battlestar(6)
: a multi-player multi-terminal	game hunt	hunt(6)
monop : monopoly	game	monop(6)
snake, snscore : display chase	game	snake(6)
trek : trekkie	game	trek(6)

worm : Play the growing worm	game	worm(6)
cfscores : the solitaire card	game canfield canfield,	canfield(6)
cribbage : the card	game cribbage	cribbage(6)
hangman : Computer version of the	game hangman	hangman(6)
boggle : play the	game of boggle	boggle(6)
zork : the	game of dungeon	zork(6)
wump : the	game of hunt-the-wumpus	wump(6)
head :	give first few lines	head(1)
graph : draw a	graph	graph(1g)
	graph : draw a graph	graph(1g)
chgrp : change	group	chgrp(1)
groups : show	group memberships	groups(1)
	groups : show group memberships	groups(1)
maintain, update, and regenerate	groups of programs :	make(1)
worm : Play the	growing worm game	worm(6)
: Computer version of the game	hangman hangman	hangman(6)
game hangman	hangman : Computer version of the	hangman(6)
	head : give first few lines	head(1)
a.help : interactive	help utility	a.help(1)
od : octal, decimal,	hex, ascii dump	od(1)
command abbreviations and	history mechanism for C shell	a.view(1)
fortune : print a random,	hopefully interesting, adage	fortune(6)
multi-terminal game	hunt : a multi-player	hunt(6)
wump : the game of	hunt-the-wumpus	wump(6)
su, ssu : substitute user	id temporarily	su(1)
program source	indent : indent and format C	indent(1)
source indent :	indent and format C program	indent(1)
ptx : permuted	index	ptx(1)
indxbib, lookbib : build inverted	index for a bibliography, find	lookbib(1)
teletypes last :	indicate last logins of users and	last(1)
a multi-routine Fortran file into	individual files fsplit : split	fsplit(1)
index for a bibliography, find	indxbib, lookbib : build inverted	lookbib(1)
: remove source unit and library	information a.rm	a.rm(1)
passwd : change password file	information chfn, chsh,	passwd(1)
out the manual man : find manual	information by keywords; print	man(1)
: eliminate .so's from nroff	input soelim	soelim(1)
documents refer : find and	insert literature references in	refer(1)
a.help :	interactive help utility	a.help(1)
: print a random, hopefully	interesting, adage fortune	fortune(6)
spline :	interpolate smooth curve	spline(1g)
indxbib, lookbib : build	inverted index for a	lookbib(1)
: multi-user wooden ships and	iron men sail	sail(6)
spooling queue lprm : remove	jobs from the line printer	lprm(1)
atrm : remove	jobs spooled by at	atrm(1)
atq : print the queue of	jobs waiting to be run	atq(1)
apropos : locate commands by	keyword lookup	apropos(1)
man : find manual information by	keywords; print out the manual	man(1)
extreme prejudice	kill : terminate a process with	kill(1)
quiz : test your	knowledge	quiz(6)
: pattern scanning and processing	language awk	awk(1)
banner : print	large banner on printer	banner(6)
executed in reverse order	lastcomm : show last commands	lastcomm(1)
at : execute commands at a	later time	at(1)
: remind you when you have to	leave leave	leave(1)
to leave	leave : remind you when you have	leave(1)
a.db : source	level debugger	a.db(1)
a.rmlib : remove compilation	library	a.rmlib(1)
versions and create a default	library : display available VADS	a.vadsrc(1)
a.which : determine which project	library contains a unit	a.which(1)
a.cleanlib : reinitialize	library directory	a.cleanlib(1)
a.mklib : make	library directory	a.mklib(1)
a.du : summarize	library disk usage	a.du(1)
a.rm : remove source unit and	library information	a.rm(1)
a.info : list or change VADS	library options	a.info(1)
a.pathxs : report or change VADS	library search list	a.path(1)
col : filter reverse	line feeds	col(1)
: produce program listing with	line numbers a.list	a.list(1)
lprm : remove jobs from the	line printer spooling queue	lprm(1)
head : give first few	lines	head(1)
uniq : report repeated	lines in a file	uniq(1)
look : find	lines in a sorted list	look(1)
rev : reverse	lines of a file	rev(1)
ln : make	links	ln(1)
or change VADS library search	list a.pathxs : report	a.path(1)
look : find lines in a sorted	list	look(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)
a.list : produce program	listing with line numbers	a.list(1)
refer : find and insert	literature references in	refer(1)
	ln : make links	ln(1)
apropos :	locate commands by keyword lookup	apropos(1)
manual for program whereis :	locate source, binary, and or	whereis(1)
last : indicate last	logins of users and teletypes	last(1)
list	look : find lines in a sorted	look(1)
for a bibliography, indxbib,	lookbib : build inverted index	lookbib(1)
: locate commands by keyword	lookup apropos	apropos(1)
program	lpq : spool queue examination	lpq(1)
printer spooling queue	lprm : remove jobs from the line	lprm(1)
	ls : list contents of directory	ls(1)
tcopy : copy a	mag tape	tcopy(1)
xsend, xget, enroll : secret	mail	send(1)
from : who is my	mail from?	from(1)
groups of make,makerules :	maintain, update, and regenerate	make(1)
update, and regenerate groups of	make,makerules : maintain,	make(1)
by keywords; print out the	manual : find manual information	man(1)
: locate source, binary, and or	manual for program whereis	whereis(1)
print out the manual man : find	manual information by keywords;	man(1)
eqn, neqn, checkeq : typeset	mathematics	eqn(1)
command abbreviations and history	mechanism for C shell establish	a.view(1)
bcd : convert to antique	media	bcd(6)
groups : show group	memberships	groups(1)
sort : sort or	merge files	sort(1)
	mesg : permit or deny messages	mesg(1)
: analyze and disperse error	messages a.error	a.error(1)
and disperse compiler error	messages error : analyze	error(1)
mesg : permit or deny	messages	mesg(1)
	mille : play mille Bournes	mille(6)
mille : play	mille Bournes	mille(6)
chmod : change	mode	chmod(1)
touch : update date last	modified of a file	touch(1)
	monop : monopoly game	monop(6)
monop :	monopoly game	monop(6)
hunt : a	multi-player multi-terminal game	hunt(6)
individual fsplit : split a	multi-routine Fortran file into	fsplit(1)
hunt : a multi-player	multi-terminal game	hunt(6)
men sail :	multi-user wooden ships and iron	sail(6)
tty : get terminal	name	tty(1)
mathematics eqn,	neqn, checkeq : typeset	eqn(1)
	nroff : text formatting	nroff(1)
typesetting troff,	nroff : text formatting and	troff(1)
soelim : eliminate .so's from	nroff input	soelim(1)
tbl : format tables for	nroff or troff	tbl(1)
colcrt : filter	nroff output for CRT previewing	colcrt(1)
checknr : check	nroff troff files	checknr(1)
constructs deroff : remove	nroff, troff, tbl and eqn	deroff(1)
to English	number : convert Arabic numerals	number(6)
arithmetic : provide drill in	number facts	arithmetic(6)
produce program listing with line	numbers a.list :	a.list(1)
number : convert Arabic	numerals to English	number(6)
od :	octal, decimal, hex, ascii dump	od(1)
dump	od : octal, decimal, hex, ascii	od(1)
: list or change VADS library	options a.info	a.info(1)
stty : set terminal	options	stty(1)
source files in dependency	order a.make : recompile	a.make(1)
last commands executed in reverse	order lastcomm : show	lastcomm(1)
colcrt : filter nroff	output for CRT previewing	colcrt(1)
pagesize : print system	page size	pagesize(1)
	pagesize : print system page size	pagesize(1)
tk :	paginator for the Tektronix 4014	tk(1)
information chfn, chsh,	passwd : change password file	passwd(1)
chfn, chsh, passwd : change	password file information	passwd(1)
language awk :	pattern scanning and processing	awk(1)
mesg :	permit or deny messages	mesg(1)
ptx :	permuted index	ptx(1)
tc :	photoypesetter simulator	tc(1)
fish :	play "Go fish"	fish(6)
mille :	play mille Bournes	mille(6)
boggle :	play the game of boggle	boggle(6)
	pr : print file	pr(1)
terminate a process with extreme	prejudice kill :	kill(1)
a.ld :	prelinker	a.ld(1)

: filter nroff output for CRT	previewing colcrt	colcrt(1)
cat : catenate and	print	cat(1)
interesting, adage fortune :	print a random, hopefully	fortune(6)
date :	print and set the date	date(1)
pr :	print file	pr(1)
banner :	print large banner on printer	banner(6)
manual information by keywords;	print out the manual man : find	man(1)
pagesize :	print system page size	pagesize(1)
to be run atq :	print the queue of jobs waiting	atq(1)
for diction diction, explain :	print wordy sentences; thesaurus	diction(1)
banner : print large banner on	printer	banner(6)
lprm : remove jobs from the line	printer spooling queue	lprm(1)
kill : terminate a	process with extreme prejudice	kill(1)
awk : pattern scanning and	processing language	awk(1)
numbers a.list :	produce program listing with line	a.list(1)
lpq : spool queue examination	program	lpq(1)
units : conversion	program	units(1)
source, binary, and or manual for	program whereis : locate	whereis(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)
indent : indent and format C	program source	indent(1)
a.ls : list compiled	programs	a.ls(1)
update, and regenerate groups of	programs : maintain,	make(1)
a.which : determine which	project library contains a unit	a.which(1)
arithmetic :	provide drill in number facts	arithmetic(6)
	ptx : permuted index	ptx(1)
from the line printer spooling	queue lprm : remove jobs	lprm(1)
lpq : spool	queue examination program	lpq(1)
atq : print the	queue of jobs waiting to be run	atq(1)
	quiz : test your knowledge	quiz(6)
	rain : animated raindrops display	rain(6)
rain : animated	raindrops display	rain(6)
adage fortune : print a	random, hopefully interesting,	fortune(6)
	ratfor : rational Fortran dialect	ratfor(1)
ratfor :	rational Fortran dialect	ratfor(1)
dependency order a.make :	recompile source files in	a.make(1)
literature references in	refer : find and insert	refer(1)
index for a bibliography, find	references in a bibliography	lookbib(1)
: find and insert literature	references in documents refer	refer(1)
: maintain, update, and	regenerate groups of programs	make(1)
a.cleanlib :	reinitialize library directory	a.cleanlib(1)
leave :	remind you when you have to leave	leave(1)
tip, cu : connect to a	remote system	tip(1c)
colrm :	remove columns from a file	colrm(1)
a.rmlib :	remove compilation library	a.rmlib(1)
spooling queue lprm :	remove jobs from the line printer	lprm(1)
atrm :	remove jobs spooled by at	atrm(1)
constructs deroff :	remove nroff, troff, tbl and eqn	deroff(1)
information a.rm :	remove source unit and library	a.rm(1)
directories rm, rmdir :	remove (unlink) files or	rm(1)
uniq : report	repeated lines in a file	uniq(1)
yes : be	repetitively affirmative	yes(1)
search list a.pathxs :	report or change VADS library	a.path(1)
uniq :	report repeated lines in a file	uniq(1)
	rev : reverse lines of a file	rev(1)
col : filter	reverse line feeds	col(1)
rev :	reverse lines of a file	rev(1)
: show last commands executed in	reverse order lastcomm	lastcomm(1)
files or directories	rm, rmdir : remove (unlink)	rm(1)
directories rm,	rmdir : remove (unlink) files or	rm(1)
robots : fight off villainous	robots	robots(6)
robots	robots : fight off villainous	robots(6)
database	roffbib : run off bibliographic	roffbib(1)
the queue of jobs waiting to be	run atq : print	atq(1)
roffbib :	run off bibliographic database	roffbib(1)
and iron men	sail : multi-user wooden ships	sail(6)
awk : pattern	scanning and processing language	awk(1)
subsystem	sccs : front end for the SCCS	sccs(1)
: report or change VADS library	search list a.pathxs	a.path(1)
xsend, xget, enroll :	secret mail	send(1)
diction, explain : print wordy	sentences; thesaurus for diction	diction(1)
apply : apply a command to a	set of arguments	apply(1)
stty :	set terminal options	stty(1)
date : print and	set the date	date(1)
and history mechanism for C	shell command abbreviations	a.view(1)
sail : multi-user wooden	ships and iron men	sail(6)

groups :	show group memberships	groups(1)
reverse order lastcomm :	show last commands executed in	lastcomm(1)
fmt :	simple text formatter	fmt(1)
tc : photoypesetter	simulator	tc(1)
pagesize : print system page	size	pagesize(1)
spline : interpolate	smooth curve	spline(1g)
game	snake, snscore : display chase	snake(6)
snake,	snscore : display chase game	snake(6)
nroff input	soelim : eliminate .so's from	soelim(1)
canfield, cfscores : the	solitaire card game canfield	canfield(6)
tsort : topological	sort	tsort(1)
	sort : sort or merge files	sort(1)
sortbib :	sort bibliographic database	sortbib(1)
sort :	sort or merge files	sort(1)
database	sortbib : sort bibliographic	sortbib(1)
look : find lines in a	sorted list	look(1)
soelim : eliminate	.so's from nroff input	soelim(1)
: indent and format C program	source indent	indent(1)
program whereis : locate	source, binary, and or manual for	whereis(1)
a.pr : format	source code	a.pr(1)
a.make : recompile	source files in dependency order	a.make(1)
a.db :	source level debugger	a.db(1)
information a.rm : remove	source unit and library	a.rm(1)
expand, unexpand : expand tabs to	spaces, and vice versa	expand(1)
spelling errors	spell, spellin, spellout : find	spell(1)
errors spell,	spellin, spellout : find spelling	spell(1)
spell, spellin, spellout : find	spelling errors	spell(1)
spell, spellin,	spellout : find spelling errors	spell(1)
	spline : interpolate smooth curve	spline(1g)
file into individual fsplit :	split a multi-routine Fortran	fsplit(1)
lpq :	spool queue examination program	lpq(1)
atrm : remove jobs	spooled by at	atrm(1)
remove jobs from the line printer	spooling queue lprm :	lprm(1)
temporarily su,	ssu : substitute user id	su(1)
	stty : set terminal options	stty(1)
temporarily	su, ssu : substitute user id	su(1)
su, ssu :	substitute user id temporarily	su(1)
sccs : front end for the SCCS	subsystem	sccs(1)
du :	summarize disk usage	du(1)
a.du :	summarize library disk usage	a.du(1)
tip, cu : connect to a remote	system	tip(1c)
pagesize : print	system page size	pagesize(1)
tbl : format	tables for nroff or troff	tbl(1)
expand, unexpand : expand	tabs to spaces, and vice versa	expand(1)
a.tags : create a	tags file	a.tags(1)
	talk : talk to another user	talk(1)
talk :	talk to another user	talk(1)
tcopy : copy a mag	tape	tcopy(1)
and execute a program on the	target board cross compilers	a.run(1)
troff	tbl : format tables for nroff or	tbl(1)
deroff : remove nroff, troff,	tbl and eqn constructs	deroff(1)
	tc : photoypesetter simulator	tc(1)
	tcopy : copy a mag tape	tcopy(1)
indicate last logins of users and	teletypes last :	last(1)
su, ssu : substitute user id	temporarily	su(1)
: animate worms on a display	terminal worms	worms(6)
tty : get	terminal name	tty(1)
stty : set	terminal options	stty(1)
prejudice kill :	terminate a process with extreme	kill(1)
quiz :	test your knowledge	quiz(6)
fmt : simple	text formatter	fmt(1)
nroff :	text formatting	nroff(1)
troff, nroff :	text formatting and typesetting	troff(1)
explain : print wordy sentences;	thesaurus for diction diction,	diction(1)
system	tip, cu : connect to a remote	tip(1c)
4014	tk : paginator for the Tektronix	tk(1)
tsort :	topological sort	tsort(1)
of a file	touch : update date last modified	touch(1)
	tr : translate characters	tr(1)
tr :	translate characters	tr(1)
	trek : trekkie game	trek(6)
trek :	trekkie game	trek(6)
tbl : format tables for nroff or	troff	tbl(1)
checknr : check nroff	troff files	checknr(1)
and typesetting	troff, nroff : text formatting	troff(1)
deroff : remove nroff,	troff, tbl and eqn constructs	deroff(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" **−g2**	Have the compiler produce additional symbol table information for full symbolic debugging and not do optimizations that limit full symbolic debugging. **−g2** 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 optimixed 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

addbib – create or extend bibliographic database

SYNOPSIS

addbib [**−p** promptfile] [**−a**] database

DESCRIPTION

When this program starts up, answering "y" to the initial "Instructions?" prompt yields directions; typing "n" or RETURN skips them. *addbib* then prompts for various bibliographic fields, reads responses from the terminal, and sends output records to a *database*. A null response (just RETURN) means to leave out that field. A minus sign (–) means to go back to the previous field. A trailing backslash allows a field to be continued on the next line. The repeating "Continue?" prompt allows the user either to resume by typing "y" or RETURN, to quit the current session by typing "n" or "q", or to edit the *database* with any system editor *(vi, ex, edit, ed)*.

The **−a** option suppresses prompting for an abstract; asking for an abstract is the default. Abstracts are ended with a CTRL-d. The **−p** option causes *addbib* to use a new prompting skeleton, defined in *promptfile*. This file should contain prompt strings, a tab, and the key-letters to be written to the *database*.

The most common key-letters and their meanings are given below. *addbib* insulates you from these key-letters, since it gives you prompts in English, but if you edit the bibliography file later on, you will need to know this information.

%A	Author's name
%B	Book containing article referenced
%C	City (place of publication)
%D	Date of publication
%E	Editor of book containing article referenced
%F	Footnote number or label (supplied by *refer*)
%G	Government order number
%H	Header commentary, printed before reference
%I	Issuer (publisher)
%J	Journal containing article
%K	Keywords to use in locating reference
%L	Label field used by **−k** option of *refer*
%M	Bell Labs Memorandum (undefined)
%N	Number within volume
%O	Other commentary, printed at end of reference
%P	Page number(s)
%Q	Corporate or Foreign Author (unreversed)
%R	Report, paper, or thesis (unpublished)
%S	Series title
%T	Title of article or book
%V	Volume number
%X	Abstract – used by *roffbib*, not by *refer*
%Y,Z	ignored by *refer*

Except for 'A', each field should be given just once. Only relevant fields should be supplied. An example is:

```
%A    Bill Tuthill
%T    Refer – A Bibliography System
%I    Computing Services
```

%C	Berkeley
%D	1982
%O	UNX 4.3.5.

FILES

promptfile	optional file to define prompting

SEE ALSO

refer(1), sortbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHORS

Al Stangenberger, Bill Tuthill

NAME

apply – apply a command to a set of arguments

SYNOPSIS

apply [**−a***c*] [**−***n*] command args ...

DESCRIPTION

apply runs the named *command* on each argument *arg* in turn. Normally arguments are chosen singly; the optional number *n* specifies the number of arguments to be passed to *command.* If *n* is zero, *command* is run without arguments once for each *arg.* Character sequences of the form %*d* in *command,* where *d* is a digit from 1 to 9, are replaced by the *d*'th following unused *arg.* If any such sequences occur, *n* is ignored, and the number of arguments passed to *command* is the maximum value of *d* in *command.* The character '%' may be changed by the **−a** option.

Examples:

```
        apply echo *
```
is similar to ls(1);
```
        apply −2 cmp a1 b1 a2 b2 ...
```
compares the 'a' files to the 'b' files;
```
        apply −0 who 1 2 3 4 5
```
runs who(1) 5 times; and
```
        apply ′ln %1 /usr/joe′ *
```
links all files in the current directory to the directory /usr/joe.

SEE ALSO

sh(1)

AUTHOR

Rob Pike

BUGS

Shell metacharacters in *command* may have bizarre effects; it is best to enclose complicated commands in single quotes ′ ′.

There is no way to pass a literal '%2' if '%' is the argument expansion character.

NAME

apropos – locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered; thus, when looking for compile, *apropos* will find all instances of 'compiler' also. Try

```
apropos password
```

and

```
apropos editor
```

If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'apropos format' and then 'man 3s printf' to get the manual on the subroutine *printf*.

apropos is actually just the **−k** option to the *man*(1) command.

FILES

/usr/man/whatis	data base

SEE ALSO

man(1), whatis(1), catman(8)

AUTHOR

William Joy

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 ASCII-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 – execute commands at a later time

SYNOPSIS

at [**-c**] [**-s**] [**-m**] time [day] [file]

DESCRIPTION

at spools away a copy of the named *file* to be used as input to *sh*(1) or *csh*(1). If the **−c** flag (for *(csh*(1))) or the **−s** flag (for *(sh*(1))) is specified, then that shell will be used to execute the job; if no shell is specified, the current environment shell is used. If no file name is specified, *at* prompts for commands from standard input until a ^D is typed.

If the **−m** flag is specified, mail will be sent to the user after the job has been run. If errors occur during execution of the job, then a copy of the error diagnostics will be sent to the user. If no errors occur, then a short message is sent informing the user that no errors occurred.

The format of the spool file is as follows: A four line header that includes the owner of the job, the name of the job, the shell used to run the job, and whether mail will be set after the job is executed. The header is followed by a *cd* command to the current directory and a *umask* command to set the modes on any files created by the job. Then *at* copies all relevant environment variables to the spool file. When the script is run, it uses the user and group ID of the creator of the spool file.

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

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

```
at 8am jan 24
at -c -m 1530 fr week
at -s -m 1200n week
```

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

Error output is lost unless redirected or the *−m* flag is requested, in which case a copy of the errors is sent to the user via *mail*(1).

FILES

/usr/spool/at	spooling area
/usr/spool/at/yy.ddd.hhhh.*	job file
/usr/spool/at/past	directory where jobs are executed from
/usr/spool/at/lasttimedone	last time atrun was run
/usr/lib/atrun	executor (run by cron(8))

SEE ALSO

atl(1), atq(1), atrm(1), calendar(1), sleep(1), cron(8).

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

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

If the system crashes, mail is not sent to the user informing them that the job was not completed.

Sometimes old spool files are not removed from the directory /usr/spool/at/past. This is usually due to a system crash, and requires that they be removed by hand.

NAME

atl - list a job waiting to be run

SYNOPSIS

atl *job#...*

DESCRIPTION

atl lists on stdout the contents of the *job#* which is waiting to be run at a later date. These jobs were created with the *at*(1) command. To obtain the needed *job#,* the user should use the *atq*(1) command.

Only the job's owner (or root) may list the contents of the job.

FILES

/usr/spool/at	spool area

AUTHOR

Roger Southwick (rogers@dadla.TEK.COM)

SEE ALSO

at(1), atq(1), atrm(1), cron(8).

NAME

atq − print the queue of jobs waiting to be run

SYNOPSIS

atq [-c] [-n] [name ...]

DESCRIPTION

atq prints the queue of jobs that are waiting to be run at a later date. These jobs were created with the *at*(1) command. With no flags, the queue is sorted in the order that the jobs will be executed.

If the **−c** flag is used, the queue is sorted by the time that the *at* command was given.

The **−n** flag prints only the total number of files that are currently in the queue.

If a name(s) is provided, only those files belonging to that user(s) are displayed.

FILES

/usr/spool/at	spool area

SEE ALSO

at(1), atl(1), atrm(1), cron(8).

NAME

atrm – remove jobs spooled by at

SYNOPSIS

atrm [-f] [-i] [-] [[job #] [name]...]

DESCRIPTION

atrm removes jobs that were created with the *at*(1) command. With the **–** flag, all jobs belonging to the person invoking *atrm* are removed. If a job number(s) is specified, *atrm* attempts to remove only that job number(s).

If the **–f** flag is used, all information regarding the removal of the specified jobs is suppressed. If the **–i** flag is used, *atrm* asks if a job should be removed; a response of 'y' causes the job to be removed.

If a user(s) name is specified, all jobs belonging to that user(s) are removed. This form of invoking *atrm* is useful only to the super-user.

FILES

/usr/spool/at	spool area

SEE ALSO

at(1), atl(1), atq(1), cron(8).

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [**−F***c*] [prog] [file] ...

DESCRIPTION

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

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

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

A pattern-action statement has the form

```
pattern { action }
```

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

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

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

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

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

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

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

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

A relational expression is one of the following:

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

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

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

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

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

or by using the **−F***c* option.

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

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

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

Add up first column, print sum and average:

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

Print fields in reverse order:

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

Print all lines between start/stop pairs:

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

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

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

SEE ALSO

lex(1), sed(1)
A. V. Aho, B. W. Kernighan, P. J. Weinberger, *awk − a pattern scanning and processing language*

BUGS

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

NAME

cat – catenate and print

SYNOPSIS

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

DESCRIPTION

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

```
cat file
```

displays the file on the standard output, 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. Output is buffered in the block size recommended by *stat*(2) unless the standard output is a terminal, when it is line buffered. The **−u** option makes the output completely unbuffered.

The **−n** option displays the output lines preceded by lines numbers, numbered sequentially from 1. The **−b** option numbers lines like **−n**, but omits the line numbers from blank lines.

The **−s** option crushes out multiple adjacent empty lines so that the output is displayed single spaced.

The **−v** option displays non-printing characters so that they are visible. Control characters print like ^X for control-x; the delete character (octal 0177) prints as ^?. Non-ascii characters (with the high bit set) are printed as M- (for meta) followed by the character of the low 7 bits. The **−e** is the same as the **−v** option, but in addition displays a '$' character at the end of each line. The **−t** is the same as the **−v** option, but in addition displays tab characters as ^I.

SEE ALSO

cp(1), ex(1), more(1), pr(1), tail(1)

BUGS

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

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

checknr – check nroff/troff files

SYNOPSIS

checknr [**−s**] [**−f**] [**−a**.x1.y1.x2.y2. ... .xn.yn] [**−c**.x1.x2.x3 ... .xn] [*file* ...]

DESCRIPTION

checknr checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, *checknr* checks the standard input. Delimeters checked are:

(1) Font changes using \fx ... \fP.

(2) Size changes using \sx ... \s0.

(3) Macros that come in open ... close forms, for example, the .TS and .TE macros which must always come in pairs.

checknr knows about the *ms*(7) and *me*(7) macro packages.

Additional pairs of macros can be added to the list using the **−a** option. This must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair .BS and .ES, use −**a**.BS.ES

The **−c** option defines commands which would otherwise be complained about as undefined.

The **−f** option requests *checknr* to ignore \f font changes.

The **−s** option requests *checknr* to ignore \s size changes.

checknr is intended to be used on documents that are prepared with *checknr* in mind, much the same as *lint*. It expects a certain document writing style for \f and \s commands, in that each \fx must be terminated with \fP and each \sx must be terminated with \s0. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr*. Since it is probably better to use the \fP and \s0 forms anyway, you should think of this as a contribution to your document preparation style.

SEE ALSO

nroff(1), troff(1), checkeq(1), ms(7), me(7)

DIAGNOSTICS

Complaints about unmatched delimiters.
Complaints about unrecognized commands.
Various complaints about the syntax of commands.

BUGS

There is no way to define a 1 character macro name using **−a**.
Does not correctly recognize certain reasonable constructs, such as conditionals.

NAME

chgrp − change group

SYNOPSIS

chgrp [-f -R] group file ...

DESCRIPTION

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

The user invoking *chgrp* must belong to the specified group and be the owner of the file, or be the super-user.

No errors are reported when the **−f** (force) option is given.

When the **−R** option is given, *chgrp* recursively descends its directory arguments setting the specified group-ID. When symbolic links are encountered, their group is changed, but they are not traversed.

FILES

/etc/group

SEE ALSO

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

NAME

chmod – change mode

SYNOPSIS

chmod [**−Rf**] mode file ...

DESCRIPTION

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

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

A symbolic *mode* has the form:

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

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

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

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

When the **−R** option is given, *chmod* recursively descends its directory arguments setting the mode for each file as described above. When symbolic links are encountered, their mode is not changed and they are not traversed.

If the **−f** option is given, *chmod* will not complain if it fails to change the mode on a file.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
chmod o−w file
chmod +X file
```

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

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

SEE ALSO

ls(1), chmod(2), stat(2), umask(2), chown(8)

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*(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.

−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 [**−bfh**]

DESCRIPTION

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

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

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

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

If the **−h** option is given, *col* converts white space to tabs to shorten printing time.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 7, 8, 9, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

SEE ALSO

troff(1), tbl(1)

BUGS

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

NAME

colcrt – filter nroff output for CRT previewing

SYNOPSIS

colcrt [**–**] [**–2**] [**–b**] [*file* ...]

DESCRIPTION

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

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

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

The option **–r** prevents underscores under blanks from being split across lines. As an example, take the text "hello_there_folks". Without **–r**, the text is printed as

```
hello there folks
     –     –
```

The **–r** option causes this to be printed as it was originally given. Note that this may not be the correct thing to do when multiple words are underlined.

A typical use of *colcrt* would be

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

SEE ALSO

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

BUGS

Can't back up more than 102 lines.

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

Lines are trimmed to 132 characters.

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

NAME

colrm – remove columns from a file

SYNOPSIS

colrm [startcol [endcol]]

DESCRIPTION

colrm removes selected columns from a file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

expand(1)

NAME

compress, uncompress, zcat – compress and expand data

SYNOPSIS

compress [**−f**] [**−v**] [**−c**] [**−b** *bits*] [*name* ...]
uncompress [**−f**] [**−v**] [**−c**] [*name* ...]
zcat [*name* ...]

DESCRIPTION

compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension **.Z,** while keeping the same ownership modes, access and modification times. If no files are specified, the standard input is compressed to the standard output. compressed files can be restored to their original form using *uncompress* or *zcat.*

The **−f** option will force compression of *name*, even if it does not actually shrink or the corresponding *name*.Z file already exists. Except when run in the background under */bin/sh*, if **−f** is not given the user is prompted as to whether an existing *name*.Z file should be overwritten.

The **−c** ("cat") option makes *compress/uncompress* write to the standard output; no files are changed. The nondestructive behavior of *zcat* is identical to that of *uncompress* **−c.**

compress uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data compression", Terry A. Welch, *IEEE Computer,* vol. 17, no. 6 (June 1984), pp. 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the **−b** flag is reached (default 16). *Bits* must be between 9 and 16. The default can be changed in the source to allow *compress* to be run on a smaller machine.

After the *bits* limit is attained, *compress* periodically checks the compression ratio. If it is increasing, *compress* continues to use the existing code dictionary. However, if the compression ratio decreases, *compress* discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the **−b** flag is omitted for *uncompress,* since the *bits* parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50–60%. compression is generally much better than that achieved by Huffman coding (as used in *pack*), or adaptive Huffman coding (*compact*), and takes less time to compute.

The **−v** option causes the printing of the percentage reduction of each file.

If an error occurs, exit status is 1, else if the last file was not compressed because it became larger, the status is 2; else the status is 0.

DIAGNOSTICS

Usage: compress [−fvc] [−b maxbits] [file ...]
    Invalid options were specified on the command line.

Missing maxbits
    Maxbits must follow **−b**.

file: not in compressed format
    The file specified to *uncompress* has not been compressed.

file: compressed with *xx* bits, can only handle *yy* bits
    File was compressed by a program that could deal with more *bits* than the

compress code on this machine. Recompress the file with smaller *bits*.

file: already has .Z suffix -- no change
: The file is assumed to be already compressed. Rename the file and try again.

file: filename too long to tack on .Z
: The file cannot be compressed because its name is longer than 12 characters. Rename and try again. This message does not occur on BSD systems.

file already exists; do you wish to overwrite (y or n)?
: Respond "y" if you want the output file to be replaced; "n" if not.

uncompress: corrupt input
: A SIGSEGV violation was detected which usually means that the input file is corrupted.

compression: *xx.xx%*
: Percentage of the input saved by compression. (Relevant only for **−v**.)

– not a regular file: unchanged
: When the input file is not a regular file, (e.g. a directory), it is left unaltered.

– has *xx* other links: unchanged
: The input file has links; it is left unchanged. See *ln*(1) for more information.

– file unchanged
: No savings is achieved by compression. The input remains virgin.

BUGS

Although compressed files are compatible between machines with large memory, **−b**12 should be used for file transfer to architectures with a small process data space (64KB or less, as exhibited by the DEC PDP series, the Intel 80286, etc.)

compress should be more flexible about the existence of the '.Z' suffix.

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 specifIed, *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 – copy

SYNOPSIS

cp [**−ip**] file1 file2

cp [**−ipr**] file ... directory

DESCRIPTION

File1 is copied onto *file2*. By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the current *umask*(2) is used. The **−p** option causes *cp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

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

cp refuses to copy a file onto itself.

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

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

SEE ALSO

cat(1), mv(1), rcp(1C)

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

crypt – encode/decode

SYNOPSIS

crypt [password]

DESCRIPTION

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

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

will print the clear.

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

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

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

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

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

FILES

/dev/tty for typed key

SEE ALSO

ed(1), makekey(8)

BUGS

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

NAME

date – print and set the date

SYNOPSIS

date [-nu] [-d dst] [-t timezone] [yymmddhhmm [.ss]]

DESCRIPTION

If no arguments are given, the current date and time are printed. Providing an argument will set the desired date; only the superuser can set the date.

The *-d* and *-t* flags set the kernel's values for daylight savings time and minutes west of GMT. If *dst* is non-zero, future calls to *gettimeofday*(2) will return a non-zero *tz_dsttime*. *Timezone* provides the number of minutes returned by future calls to *gettimeofday*(2) in *tz_minuteswest*. You should also set the default timezone by using the command *zic(8)*. Details are given in that manual page.

The *-u* flag is used to display or set the date in GMT (universal) time. *yy* represents the last two digits of the year; the first *mm* is the month number; *dd* is the day number; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *.ss* is optional and represents the seconds. For example:

```
date 8506131627
```

sets the date to June 13 1985, 4:27 PM. The year, month and day may be omitted; the default values will be the current ones. The system operates in GMT. *date* takes care of the conversion to and from local standard and daylight-saving time.

If *timed(8)* is running to synchronize the clocks of machines in a local area network, *date* sets the time globally on all those machines unless the **–n** option is given.

FILES

/usr/adm/wtmp to record time-setting. In /usr/adm/messages, *date* records the name of the user setting the time.

SEE ALSO

zic(1), gettimeofday(2), utmp(5), timed(8),
TSP: The Time Synchronization Protocol for UNIX 4.3BSD, R. Gusella and S. Zatti

DIAGNOSTICS

Exit status is 0 on success, 1 on complete failure to set the date, and 2 on successfully setting the local date but failing globally.

Occasionally, when *timed* synchronizes the time on many hosts, the setting of a new time value may require more than a few seconds. On these occasions, *date* prints: 'Network time being set'. The message 'Communication error with timed' occurs when the communication between *date* and *timed* fails.

BUGS

The system attempts to keep the date in a format closely compatible with VMS. VMS, however, uses local time (rather than GMT) and does not understand daylight-saving time. Thus, if you use both UNIX and VMS, VMS will be running on GMT.

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

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

option	*values*
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
files=*n*	copy *n* input files before terminating (makes sense only where input is a magtape or similar device).
seek=*n*	seek *n* records from beginning of output file before copying
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
block	convert variable length records to fixed length
unblock	convert fixed length records to variable length
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

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

Cbs is used only if *ascii, unblock, ebcdic, ibm,* or *block* conversion is specified. In the first two cases, *cbs* characters are placed into the conversion buffer, any specified character mapping is done, trailing blanks trimmed and new-line added before sending the line to the output. In the latter three cases, characters are read into the conversion buffer, and blanks added to make up an output record of size *cbs*.

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

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

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

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

SEE ALSO

cp(1), tr(1)

DIAGNOSTICS

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

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The 'ibm' conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.
One must specify "conv=noerror,sync" when copying raw disks with bad sectors to insure *dd* stays synchronized.

Certain combinations of arguments to *conv=* are permitted. However, the *block* or *unblock* option cannot be combined with *ascii*, *ebcdic* or *ibm*. Invalid combinations *silently ignore* all but the last mutually-exclusive keyword.

NAME

deroff – remove nroff, troff, tbl and eqn constructs

SYNOPSIS

deroff [**−w**] file ...

DESCRIPTION

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

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

SEE ALSO

troff(1), eqn(1), tbl(1)

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output. slkfjsldfjsdlslk.

NAME

diction, explain – print wordy sentences; thesaurus for diction

SYNOPSIS

diction [**−ml**] [**−mm**] [**−n**] [**−f** pfile] file ...
explain

DESCRIPTION

diction finds all sentences in a document that contain phrases from a data base of bad or wordy diction. Each phrase is bracketed with [ ]. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package **−ms** may be overridden with the flag **−mm.** The flag **−ml** which causes **deroff** to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with **−f** *pfile.* If the flag **−n** is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by diction.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks. In particular, *diction* doesn't grok **−me.**

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

du – summarize disk usage

SYNOPSIS

du [**−a**] [**−f**] [−s] [name ...]

DESCRIPTION

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

The option **−s** causes only the grand total to be given.

The option **−a** causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

The option **−f** causes the size calculation of directories to ignore sizes of subdirectories. This is useful for finding "fat" directories (those that have lots of files at the top level, as opposed to trees that have lots of files).

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

SEE ALSO

df(1), quot(8)

BUGS

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

In previous versions of *du* , if there are too many distinct linked files, *du* counts the excess files multiply.

Previous versions of *du* required options to be in the order **−s** followed by **−a** if both are given. This version uses *getopt(3),* and thus any order is allowed.

NAME

eqn, neqn, checkeq – typeset mathematics

SYNOPSIS

eqn [**−d**xy] [**−p**n] [**−s**n] [**−f**n] [file] ...
checkeq [file] ...

DESCRIPTION

eqn is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, *neqn* on terminals. Usage is almost always

```
eqn file ... | troff
neqn file ... | nroff
```

If no files are specified, these programs read from the standard input. A line beginning with '.EQ' marks the start of an equation; the end of an equation is marked by a line beginning with '.EN'. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as 'delimiters'; subsequent text between delimiters is also treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument **−d***xy* or (more commonly) with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by 'delim off'. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within *eqn* are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup.** Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things: $\lim\limits_{n \to \infty} \sum\limits_0^n x_i$ is made with *lim from {n−> inf } sum from 0 to n x sub i.*

Left and right brackets, braces, etc., of the right height are made with **left** and **right:** *left [x sup 2 + y sup 2 over alpha right] ~=~1* produces $\left[x^2+\frac{y^2}{\alpha} \right] = 1$. The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

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

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x} = \overline{f(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with **size** *n* or **size** $\pm n$, **roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **−s***n* and **−f***n*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **−p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like *sum*($\sum$) *int* ($\int$) *inf* (∞) and shorthands like >= ($\geq$) -> ($\rightarrow$), and != ($\neq$) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *Troff*(1) four-character escapes like \(bs (∅) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

SEE ALSO

troff(1), tbl(1), ms(7), eqnchar(7)
B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics–User's Guide*
J. F. Ossanna, *NROFF/TROFF User's Manual*

BUGS

To embolden digits, parens, etc., it is necessary to quote them, as in 'bold "12.3"'.

NAME

error – analyze and disperse compiler error messages

SYNOPSIS

error [**−n**] [**−s**] [**−q**] [**−v**] [**−t** suffixlist] [**−I** ignorefile] [name]

DESCRIPTION

error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding to which the line the error message refers. error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *error* touches source files only after all input has been read. By specifying the **−q** query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise *error* proceeds on its merry business. If the **−t** touch option and associated suffix list is given, *error* will restrict itself to touch only those files with suffices in the suffix list. error also can be asked (by specifying **−v**) to invoke *vi*(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *csh* syntax,

```
make −s lint |& error −q −v
```

will analyze all the error messages produced by whatever programs *make* runs when making lint.

error knows about the error messages produced by: *make, cc, cpp, ccom, as, ld, lint, pi, pc, f77,* and *DEC Western Research Modula-2. error* knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except *Pascal,* error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; *error* will duplicate the error message and insert it at all of the places referenced.

error will do one of six things with error messages.

synchronize
: Some language processors produce short errors describing which file it is processing. *error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error.*

discard
: error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/llib-lc* and */usr/lib/llib-port* are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify
: error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The

names of functions to ignore are taken from either the file named *.errorrc* in the users's home directory, or from the file named by the **−I** option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific
: error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They will not be inserted into any source file.

file specific
: error message that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other error messages are consumed entirely by *error* or are written to the standard output. *error* inserts the error messages into the source file on the line preceding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error, and "%%%" at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, programs with comments and source on the same line should be formatted so that language statements appear before comments.

Options available with *error* are:

−n Do *not* touch any files; all error messages are sent to the standard output.

−q The user is *queried* whether s/he wants to touch the file. A "y" or "n" to the question is necessary to continue. Absence of the **−q** option implies that all referenced files (except those referring to discarded error messages) are to be touched.

−v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.

−t Take the following argument as a suffix list. Files whose suffixes do not appear in the suffix list are not touched. The suffix list is dot separated, and "*" wildcards work. Thus the suffix list:

```
".c.y.foo*.h"
```

allows *error* to touch files ending with ".c", ".y", ".foo*" and ".y".

−s Print out *statistics* regarding the error categorization. Not too useful.

error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

AUTHOR

Robert Henry

FILES

~/.errorrc	function names to ignore for *lint* error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the ' |' marking the point of error is also disturbed by *error.*

error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [−tabstop] [−tab1,tab2,...,tabn] [file ...]
unexpand [**−a**] [file ...]

DESCRIPTION

expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given, then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default, only leading blanks and tabs are reconverted to maximal strings of tabs. If the **−a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

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 ɔ 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(

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

find − find files

SYNOPSIS

find pathname-list expression
find pattern

DESCRIPTION

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

The second form rapidly searches a database for all pathnames which match *pattern*. Usually the database is recomputed weekly and contains the pathnames of all files which are publicly accessible. If escaped, normal shell "globbing" characters ('*', '?', '[', and ']') may be used in *pattern*, but the matching differs in that no characters (*e.g.* '/') have to be matched explicitly. As a special case, a simple *pattern* containing no globbing characters is matched as though it were **pattern**; if any globbing character appears there are no implicit globbing characters.

−name filename
: True if the *filename* argument matches the current file name. Normal shell argument syntax may be used if escaped (watch out for '[', '?' and '*').

−perm onum
: True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared: *(flags&onum)==onum*.

−fstype *type*
: True if the filesystem to which the file belongs is of type *type*, where *type* is typically **4.3** or **nfs**.

−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.

−links n
: True if the file has *n* links.

−prune
: Always true. Has the side effect of pruning the search tree at the file. That is, if the current pathname is a directory, *find* will not descend into that directory.

−depth
: Always true. Has the side effect of causing a depth-first search. That is, the children of the directory are processed before the directory itself. The options **−cpio** and **−ncpio** cause **−depth** to be turned on by default.

−user uname
: True if the file belongs to the user *uname* (login name or numeric user ID).

−nouser
: True if the file belongs to a user *not* in the /etc/passwd database.

−group gname
: True if the file belongs to group *gname* (group name or numeric group ID).

−nogroup
: True if the file belongs to a group *not* in the /etc/group database.

−size n
: True if the file is *n* blocks long (512 bytes per block).

−inum n
: True if the file has inode number *n*.

−atime n
: True if the file has been accessed in *n* days.

−ctime n
: True if the inode has been changed in *n* days (see *ls(1)* for more information).

−mtime n True if the file has been modified in *n* days.

−exec command
True if the executed command returns a zero value as exit status. The end of the command must be punctuated by an escaped semicolon. A command argument '{}' is replaced by the current pathname.

−ok command
Like **−exec** except that the generated command is written on the standard output, then the standard input is read and the command executed only upon response **y**.

−print Always true; causes the current pathname to be printed.

−ls Always true; causes current pathname to be printed together with its associated statistics. These include (respectively) inode number, size in kilobytes (1024 bytes), protection mode, number of hard links, user, group, size in bytes, and modification time. If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "->". The format is identical to that of "ls -gilds" (note however that formatting is done internally, without executing the ls program).

−newer file
True if the current file has been modified more recently than the argument *file*.

−cpio file Write the current file on the argument *file* in standard *cpio* format. Implies **−depth**.

−ncpio file
Write the current file on the argument *file* in ASCII (**−C**) *cpio* format. Implies **−depth**.

−xdev Always true; causes find *not* to traverse down into a file system different from the one on which current *argument* pathname resides.

The primaries may be combined using the following operators (in order of decreasing precedence):

1) A parenthesized group of primaries and operators (parentheses are special to the Shell and must be escaped).

2) The negation of a primary ('!' is the unary *not* operator).

3) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

4) Alternation of primaries ('**−o**' is the *or* operator).

EXAMPLES

To find all accessible files whose pathname contains 'find':

```
find find
```

To typeset all variants of manual pages for 'ls':

```
vtroff -man 'find '*man*/ls.?"
```

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 {} \;
```

The following will print *ls*-like information for all files in the current directory and subdirectories except for files in RCS directories:

```
find . -name RCS -prune -o -print
```

FILES

/etc/passwd	
/etc/group	
/usr/lib/find/find.codes	coded pathnames database

SEE ALSO

sh(1), test(1), fs(5)
Relevant paper in February, 1983 issue of *;login:*.

BUGS

The first form's syntax is painful, and the second form's exact semantics is confusing and can vary from site to site.

More than one '-newer' option does not work properly.

NAME

fmt – simple text formatter

SYNOPSIS

fmt [name ...]

DESCRIPTION

fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, within visual mode of the *ex* editor (e.g. *vi*) the command

```
	!}fmt
```

will reformat a paragraph, evening the lines.

SEE ALSO

nroff(1), mail(1)

AUTHOR

Kurt Shoens

BUGS

The program was designed to be simple and fast – for more complex operations, the standard text processors are likely to be more appropriate.

NAME

from – who is my mail from?

SYNOPSIS

from [**−s** sender] [user]

DESCRIPTION

from prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own. If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1)

NAME

fsplit – split a multi-routine Fortran file into individual files

SYNOPSIS

fsplit [**-e** efile] ... [file]

DESCRIPTION

fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form *name.f,* where *name* is the name of the program unit (e.g. function, subroutine, block data or program). The name for unnamed block data subprograms has the form *blkdtaNNN.f* where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form *mainNNN.f.* If there is an error in classifying a program unit, or if *name.f* already exists, the program unit will be put in a file of the form *zzzNNN.f* where *zzzNNN.f* does not already exist.

Normally each subprogram unit is split into a separate file. When the *-e* option is used, only the specified subprogram units are split into separate files. E.g.:

```
fsplit -e readit -e doit prog.f
```

will split readit and doit into separate files.

DIAGNOSTICS

If names specified via the *-e* option are not found, a diagnostic is written to *standard error.*

AUTHOR

Asa Romberger and Jerry Berkman

BUGS

fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse *fsplit.*

It is hard to use *-e* for unnamed main programs and block data subprograms since you must predict the created file name.

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

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

graph − draw a graph

SYNOPSIS

graph [option] ...

DESCRIPTION

graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *plot*(1G) filters.

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

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

−a	Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **−x**).
−b	Break (disconnect) the graph after each label in the input.
−c	Character string given by next argument is default label for each point.
−g	Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
−l	Next argument is label for graph.
−m	Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers.
−s	Save screen, don't erase before plotting.
−x [**l**]	If **l** is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally these quantities are determined automatically.
−y [**l**]	Similarly for *y*.
−h	Next argument is fraction of space for height.
−w	Similarly for width.
−r	Next argument is fraction of space to move right before plotting.
−u	Similarly to move up before plotting.
−t	Transpose horizontal and vertical axes. (Option **−x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **−s** option is present.

If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

spline(1G), plot(1G)

BUGS

graph stores all points internally and drops those for which there isn't room.
Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

groups – show group memberships

SYNOPSIS

groups [user]

DESCRIPTION

The *groups* command shows the groups to which you or the optionally specified user belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but belong to the group which it is owned by then you are granted group access to the file.

When a new file is created it is given the group of the containing directory.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

BUGS

More groups should be allowed.

NAME

head – give first few lines

SYNOPSIS

head [−count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1)

NAME

indent – indent and format C program source

SYNOPSIS

indent [*input-file* [*output-file*]] [**−bad** | **−nbad**] [**−bap** | **−nbap**] [**−bbb** | **−nbbb**] [**−bc** | **−nbc**] [**−bl** | **−br**] [**−c***n*] [**−cd***n*] [**−cdb** | **−ncdb**] [**−ce** | **−nce**] [**−ci***n*] [**−cli***n*] [**−d***n*] [**−di***n*] [**−dj** | **−ndj**] [**−ei** | **−nei**] [**−fc1** | **−nfc1**] [**−i***n*] [**−ip** | **−nip**] [**−l***n*] [**−lc***n*] [**−lp** | **−nlp**] [**−npro**] [**−pcs** | **−npcs**] [**−ps** | **−nps**] [**−psl** | **−npsl**] [**−sc** | **−nsc**] [**−sob** | **−nsob**] [**−st**] [**−troff**] [**−v** | **−nv**]

DESCRIPTION

indent is a **C** program formatter. It reformats the **C** program in the *input-file* according to the switches. The switches which can be specified are described below. They may appear before or after the file names.

NOTE: If you only specify an *input-file,* the formatting is done 'in-place', that is, the formatted file is written back into *input-file* and a backup copy of *input-file* is written in the current directory. If *input-file* is named '/blah/blah/file', the backup file is named file.*BAK.*

If *output-file* is specified, *indent* checks to make sure it is different from *input-file*.

OPTIONS

The options listed below control the formatting style imposed by *indent*.

−bad,−nbad If **−bad** is specified, a blank line is forced after every block of declarations. Default: **−nbad**.

−bap,−nbap If **−bap** is specified, a blank line is forced after every procedure body. Default: **−nbap.**

−bbb,−nbbb If **−bbb** is specified, a blank line is forced before every block comment. Default: **−nbbb.**

−bc,−nbc If **−bc** is specified, then a newline is forced after each comma in a declaration. **−nbc** turns off this option. The default is **−nbc**.

−br,−bl Specifying **−bl** lines up compound statements like this:

```
if (...)
{
    code
}
```

Specifying **−br** (the default) makes them look like this:

```
if (...) {
    code
}
```

−c*n* The column in which comments on code start. The default is 33.

−cd*n* The column in which comments on declarations start. The default is for these comments to start in the same column as those on code.

−cdb, −ncdb Enables (disables) the placement of comment delimiters on blank lines. With this option enabled, comments look like this:

```
/*
 * this is a comment
 */
```

Rather than like this:

```
/* this is a comment */
```

This only affects block comments, not comments to the right of code. The default is **−cdb**.

−ce, −nce Enables (disables) forcing 'else's to cuddle up to the immediately preceding '}'. The default is **−ce**.

−ci*n* Sets the continuation indent to be *n*. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless **−lp** is in effect. **−ci** defaults to the same value as **−i**.

−cli*n* Causes case labels to be indented *n* tab stops to the right of the containing **switch** statement. **−cli0.5** causes case labels to be indented half a tab stop. The default is **−cli0**. (This is the only option that takes a fractional argument.)

−d*n* Controls the placement of comments which are not to the right of code. Specifying **−d1** means that such comments are placed one indentation level to the left of code. The default **−d0** lines up these comments with the code. See the section on comment indentation below.

−di*n* Specifies the indentation, in character positions, from a declaration keyword to the following identifier. The default is **−di16**.

−dj, −ndj **−dj** left justifies declarations. **−ndj** indents declarations the same as code. The default is **−ndj**.

−ei, −nei Enables (disables) special **else-if** processing. If enabled, **if**s following **else**s will have the same indentation as the preceding **if** statement. The default is **−ei**.

−fc1, −nfc1 Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, **−nfc1** should be used. The default is **−fc1**.

−i*n* The number of spaces for one indentation level. The default is 8.

−ip, −nip Enables (disables) the indentation of parameter declarations from the left margin. The default is **−ip**.

−l*n* Maximum length of an output line. The default is 78.

−lp, −nlp Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left paren. For example, here is how a piece of continued code looks with **−nlp** in effect:

```
p1 = first_procedure(second_procedure(p2, p3),
   third_procedure(p4, p5));
```

With **−lp** in effect (the default) the code looks somewhat clearer:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

Inserting two more newlines we get:

```
p1 = first_procedure(second_procedure(p2,
                                      p3),
                     third_procedure(p4,
                                     p5));
```

−npro Causes the profile files, './.indent.pro' and '~/.indent.pro', to be ignored.

−pcs, −npcs If true (**−pcs**) all procedure calls will have a space inserted between the name and the '('. The default is **−npcs**.

−ps, −nps If true (**−ps**) the pointer following operator '−>' will be surrounded by spaces on either side. The default is **−nps**.

−psl,−npsl If true (**−psl**) the names of procedures being defined are placed in column 1 – their types, if any, will be left on the previous lines. The default is **−psl.**

−sc,−nsc Enables (disables) the placement of asterisks ('*'s) at the left edge of all comments. The default is **−sc**.

−sob,−nsob If **−sob** is specified, indent will swallow optional blank lines. You can use this to get rid of blank lines after declarations. Default: **−nsob**.

−st Causes **indent** to take its input from stdin, and put its output to stdout.

−T*typename* Adds *typename* to the list of type keywords. Names accumulate: **−T** can be specified more than once. You need to specify all the typenames that appear in your program that are defined by **typedef**s – nothing will be harmed if you miss a few, but the program won't be formatted as nicely as it should. This sounds like a painful thing to have to do, but it's really a symptom of a problem in C: **typedef** causes a syntactic change in the language and *indent* can't find all **typedef**s.

−troff Causes **indent** to format the program for processing by troff. It will produce a fancy listing in much the same spirit as **vgrind**. If the output file is not specified, the default is standard output, rather than formatting in place.

−v,−nv **−v** turns on 'verbose' mode; **−nv** turns it off. When in verbose mode, *indent* reports when it splits one line of input into two or more lines of output, and gives some size statistics at completion. The default is **−nv**.

FURTHER DESCRIPTION

You may set up your own 'profile' of defaults to *indent* by creating a file called *.indent.pro* in either your login directory and/or the current directory and including whatever switches you like. Switches in '.indent.pro' in the current directory override those in your login directory (with the exception of **-T** type definitions, which just accumulate). If *indent* is run and a profile file exists, then it is read to set up the program's defaults. The switches should be separated by spaces, tabs or newlines. Switches on the command line, however, override profile switches.

Comments

'Box' comments. *indent* assumes that any comment with a dash or star immediately after the start of comment (that is, '/*–' or '/**') is a comment surrounded by a box of stars. Each line of such a comment is left unchanged, except that its indentation may be adjusted to account for the change in indentation of the first line of the comment.

Straight text. All other comments are treated as straight text. *indent* fits as many words (separated by blanks, tabs, or newlines) on a line as possible. Blank lines break paragraphs.

Comment indentation

If a comment is on a line with code it is started in the 'comment column', which is set by the **−c***n* command line parameter. Otherwise, the comment is started at *n* indentation levels less than where code is currently being placed, where *n* is specified by the **−d***n* command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

Preprocessor lines

In general, *indent* leaves preprocessor lines alone. The only reformatting that it will do is to straighten up trailing comments. It leaves embedded comments alone. Conditional compilation (**#ifdef...#endif**) is recognized and *indent* attempts to correctly compensate for the syntactic peculiarities introduced.

C syntax

indent understands a substantial amount about the syntax of C, but it has a 'forgiving' parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

```
    #define forever for(;;)
```

is handled properly.

FILES

./.indent.pro	profile file
~/.indent.pro	profile file

BUGS

indent has even more switches than *ls*.

A common mistake that often causes grief is typing:

```
  indent *.c
```

to the shell in an attempt to indent all the **C** programs in a directory. This is probably a bug, not a feature.

NAME

kill − terminate a process with extreme prejudice

SYNOPSIS

kill [**−**sig] processid ...
kill −l

DESCRIPTION

kill sends the TERM (terminate, 15) signal to the specified processes. If a signal name or number preceded by '−' is given as first argument, that signal is sent instead of terminate (see *sigvec*(2)). The signal names are listed by 'kill −l', and are as given in */usr/include/signal.h,* stripped of the common SIG prefix.

The terminate signal will kill processes that do not catch the signal; 'kill −9 ...' is a sure kill, as the KILL (9) signal cannot be caught. By convention, if process number 0 is specified, all members in the process group (i.e. processes resulting from the current login) are signaled (but beware: this works only if you use *sh*(1); not if you use *csh*(1).) Negative process numbers also have special meanings; see *kill*(2) for details.

The killed processes must belong to the current user unless he is the super-user.

The process number of an asynchronous process started with '&' is reported by the shell. Process numbers can also be found by using *ps*(1). *kill* is a built-in to *csh*(1); it allows job specifiers of the form "%..." as arguments so process id's are not as often used as *kill* arguments. See *csh*(1) for details.

SEE ALSO

csh(1), ps(1), kill(2), sigvec(2)

BUGS

A replacement for "kill 0" for *csh*(1) users should be provided.

NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [**−***N*] [**−a**] [*name...*] [*tty...*]

DESCRIPTION

last will look back in the *wtmp* file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example 'last 0' is the same as 'last tty0'. If multiple arguments are given, the information which applies to any of the arguments is printed. For example 'last root console' would list all of "root's" sessions as well as all sessions on the console terminal. *last* will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, *last* so indicates.

The pseudo-user **reboot** logs in at reboots of the system, thus

```
last reboot
```

will give an indication of mean time between reboot.

If *last* is interrupted, it indicates how far the search has progressed in *wtmp*. If interrupted with a quit signal (generated by a control-\) *last* indicates how far the search has progressed so far, and the search continues.

OPTIONS

−*N* By default, all logins and logouts that match are printed. This option limits the report to *N* lines.

−a Print information from all data base files. By default, the report only includes data from */usr/adm/wtmp,* which only applies to the last current period (usually a month). The **−a** option tells **last** to get data from *wtmp.0, wtmp.1,* and so forth.

FILES

/usr/adm/wtmp*	login data base files
/usr/adm/shutdownlog	which records shutdowns and reasons for same

SEE ALSO

wtmp(5), ac(8), lastcomm(1)

AUTHOR

Howard Katseff

BUGS

The **−a** option causes **last** to look at *wtmp, wtmp.0, wtmp.1,* and so forth, but it stops as soon as it finds a missing file, so if there is a *wtmp.5* but no *wtmp.4,* processing stops with *wtmp.3.*

NAME

lastcomm – show last commands executed in reverse order

SYNOPSIS

lastcomm [command name] ... [user name] ... [terminal name] ...

DESCRIPTION

lastcomm gives information on previously executed commands. With no arguments, *lastcomm* prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. So, for example,

```
lastcomm a.out root ttyd0
```

would produce a listing of all the executions of commands named *a.out* by user *root* on the terminal *ttyd0*.

For each process entry, the following are printed.

The name of the user who ran the process.
Flags, as accumulated by the accounting facilities in the system.
The command name under which the process was called.
The amount of cpu time used by the process (in seconds).
The time the process exited.

The flags are encoded as follows: "S" indicates the command was executed by the super-user, "F" indicates the command ran after a fork, but without a following *exec*, "C" indicates the command was run in PDP-11 compatibility mode (VAX only), "D" indicates the command terminated with the generation of a *core* file, and "X" indicates the command was terminated with a signal.

FILES

/usr/adm/acct

SEE ALSO

last(1), sigvec(2), acct(8), core(5)

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

leave – remind you when you have to leave

SYNOPSIS

leave [[**+**]*hhmm*]

DESCRIPTION

leave waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, *leave* exits just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If the time is preceeded by '+', the alarm will go off in hours and minutes from the current time.

If no argument is given, *leave* prompts with "When do you have to leave?". A reply of newline causes *leave* to exit, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile.*

leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill −9" giving its process id.

If *leave* is executed within the *script(1)* or *window(1)* command, it prints the message "Unregistered login" and aborts. This is because the only way *leave* can tell if you have logged out is to look at the login resgistry information for the port in the file */etc/utmp*, and these commands do not register the port.

SEE ALSO

calendar(1)

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

ln – make links

SYNOPSIS

ln [**–s**] sourcename [targetname]
ln [**–s**] sourcename1 sourcename2 [sourcename3 ...] targetdirectory

DESCRIPTION

A link is a directory entry referring to a file; the same file (together with its size, all its protection information, etc.) may have several links to it. There are two kinds of links: hard links and symbolic links.

By default *ln* makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

The **–s** option causes *ln* to create symbolic links. A symbolic link contains the name of the file to which it is linked. The referenced file is used when an *open*(2) operation is performed on the link. A *stat*(2) on a symbolic link will return the linked-to file; an *lstat*(2) must be done to obtain information about the link. The *readlink*(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

Given one or two arguments, *ln* creates a link to an existing file *sourcename*. If *targetname* is given, the link has that name; *targetname* may also be a directory in which to place the link; otherwise it is placed in the current directory. If only the directory is specified, the link will be made to the last component of *sourcename*.

Given more than two arguments, *ln* makes links in *targetdirectory* to all the named source files. The links made will have the same name as the files being linked to.

SEE ALSO

rm(1), cp(1), mv(1), link(2), readlink(2), stat(2), symlink(2)

NAME

look – find lines in a sorted list

SYNOPSIS

look [**−df**] string [file]

DESCRIPTION

look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options **d** and **f** affect comparisons as in *sort*(1):

d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence **−df.**

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)

NAME

indxbib, lookbib – build inverted index for a bibliography, find references in a bibliography

SYNOPSIS

indxbib database ...
lookbib [**−n**] database

DESCRIPTION

Indxbib makes an inverted index to the named *databases* (or files) for use by *lookbib*(1) and *refer*(1). These files contain bibliographic references (or other kinds of information) separated by blank lines.

A bibliographic reference is a set of lines, constituting fields of bibliographic information. Each field starts on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, which may continue until the next line starting with "%".

Indxbib is a shell script that calls /usr/lib/refer/mkey and /usr/lib/refer/inv. The first program, *mkey,* truncates words to 6 characters, and maps upper case to lower case. It also discards words shorter than 3 characters, words among the 100 most common English words, and numbers (dates) < 1900 or > 2000. These parameters can be changed; see page 4 of the *Refer* document by Mike Lesk. The second program, *inv,* creates an entry file (.ia), a posting file (.ib), and a tag file (.ic), all in the working directory.

lookbib uses an inverted index made by *indxbib* to find sets of bibliographic references. It reads keywords typed after the ">" prompt on the terminal, and retrieves records containing all these keywords. If nothing matches, nothing is returned except another ">" prompt.

lookbib will ask if you need instructions, and will print some brief information if you reply "y". The "−n" flag turns off the prompt for instructions.

It is possible to search multiple databases, as long as they have a common index made by *indxbib.* In that case, only the first argument given to *indxbib* is specified to *lookbib.*

If *lookbib* does not find the index files (the .i[abc] files), it looks for a reference file with the same name as the argument, without the suffixes. It creates a file with a '.ig' suffix, suitable for use with *fgrep.* It then uses this fgrep file to find references. This method is simpler to use, but the .ig file is slower to use than the .i[abc] files, and does not allow the use of multiple reference files.

FILES

x.ia, *x*.ib, *x*.ic, where *x* is the first argument, or if these are not present, then *x*.ig, *x*

SEE ALSO

refer(1), addbib(1), sortbib(1), roffbib(1), lookbib(1)

BUGS

Probably all dates should be indexed, since many disciplines refer to literature written in the 1800s or earlier.

NAME

lpq − spool queue examination program

SYNOPSIS

lpq [+[n]] [-l] [−Pprinter] [job # ...] [user ...]

DESCRIPTION

lpq examines the spooling area used by *lpd*(8) for printing files on the line printer, and reports the status of the specified jobs or all jobs associated with a user. *lpq* invoked without any arguments reports on any jobs currently in the queue. A **−P** flag may be used to specify a particular printer, otherwise the default line printer is used (or the value of the PRINTER variable in the environment). If a + argument is supplied, *lpq* displays the spool queue until it empties. Supplying a number immediately after the + sign indicates that *lpq* should sleep *n* seconds in between scans of the queue. All other arguments supplied are interpreted as user names or job numbers to filter out only those jobs of interest.

For each job submitted (i.e. invocation of *lpr*(1)) *lpq* reports the user's name, current rank in the queue, the names of files comprising the job, the job identifier (a number which may be supplied to *lprm*(1) for removing a specific job), and the total size in bytes. The **−l** option causes information about each of the files comprising the job to be printed. Normally, only as much information as will fit on one line is displayed. Job ordering is dependent on the algorithm used to scan the spooling directory and is supposed to be FIFO (First in First Out). File names comprising a job may be unavailable (when *lpr*(1) is used as a sink in a pipeline) in which case the file is indicated as "(standard input)".

If *lpq* warns that there is no daemon present (i.e. due to some malfunction), the *lpc*(8) command can be used to restart the printer daemon.

FILES

/etc/termcap	for manipulating the screen for repeated display
/etc/printcap	to determine printer characteristics
/usr/spool/*	the spooling directory, as determined from printcap
/usr/spool/*/cf*	control files specifying jobs
/usr/spool/*/lock	the lock file to obtain the currently active job

SEE ALSO

lpr(1), lprm(1), lpc(8), lpd(8)

BUGS

Due to the dynamic nature of the information in the spooling directory lpq may report unreliably. Output formatting is sensitive to the line length of the terminal; this can results in widely spaced columns.

DIAGNOSTICS

Unable to open various files. The lock file being malformed. Garbage files when there is no daemon active, but files in the spooling directory.

NAME

lpr – off line print

SYNOPSIS

lpr [**−P***printer*] [**−#***num*] [**−C** *class*] [**−J** *job*] [**−T** *title*] [**−i** [*numcols*]] [**−1234** *font*] [**−w***num*] [**−pltndgvcfrmhs**] [name ...]

DESCRIPTION

lpr uses a spooling daemon to print the named files when facilities become available. If no names appear, the standard input is assumed. The **−P** option may be used to force output to a specific printer. Normally, the default printer is used (site dependent), or the value of the environment variable PRINTER is used.

The following single letter options are used to notify the line printer spooler that the files are not standard text files. The spooling daemon will use the appropriate filters to print the data accordingly.

−p Use *pr*(1) to format the files (equivalent to *print*).

−l Use a filter which allows control characters to be printed and suppresses page breaks.

−t The files are assumed to contain data from *troff*(1) (cat phototypesetter commands).

−n The files are assumed to contain data from *ditroff* (device independent troff).

−d The files are assumed to contain data from *tex*(l) (DVI format from Stanford).

−g The files are assumed to contain standard plot data as produced by the *plot*(3X) routines (see also *plot*(1G) for the filters used by the printer spooler).

−v The files are assumed to contain a raster image for devices like the Benson Varian.

−c The files are assumed to contain data produced by *cifplot*(l).

−f Use a filter which interprets the first character of each line as a standard FORTRAN carriage control character.

The remaining single letter options have the following meaning.

−r Remove the file upon completion of spooling or upon completion of printing (with the **−s** option).

−m Send mail upon completion.

−h Suppress the printing of the burst page.

−s Use symbolic links. Usually files are copied to the spool directory.

The **−C** option takes the following argument as a job classification for use on the burst page. For example,

```
lpr −C EECS foo.c
```

causes the system name (the name returned by *hostname*(1)) to be replaced on the burst page by EECS, and the file foo.c to be printed.

The **−J** option takes the following argument as the job name to print on the burst page. Normally, the first file's name is used.

The **−T** option uses the next argument as the title used by *pr*(1) instead of the file name.

To get multiple copies of output, use the **−#***num* option, where *num* is the number of copies desired of each file named. For example,

```
lpr −#3 foo.c bar.c more.c
```

would result in 3 copies of the file foo.c, followed by 3 copies of the file bar.c, etc. On the other hand,

```
cat foo.c bar.c more.c | lpr -#3
```

will give three copies of the concatenation of the files.

The **−i** option causes the output to be indented. If the next argument is numeric, it is used as the number of blanks to be printed before each line; otherwise, 8 characters are printed.

The **−w** option takes the immediately following number to be the page width for *pr*.

The **−s** option will use *symlink*(2) to link data files rather than trying to copy them so large files can be printed. This means the files should not be modified or removed until they have been printed.

The option **−1234** Specifies a font to be mounted on font position *i*. The daemon will construct a *.railmag* file referencing */usr/lib/vfont/name.size*.

FILES

/etc/passwd	personal identification
/etc/printcap	printer capabilities data base
/usr/lib/lpd*	line printer daemons
/usr/spool/*	directories used for spooling
/usr/spool/*/cf*	daemon control files
/usr/spool/*/df*	data files specified in "cf" files
/usr/spool/*/tf*	temporary copies of "cf" files

SEE ALSO

lpq(1), lprm(1), pr(1), symlink(2), printcap(5), lpc(8), lpd(8)

DIAGNOSTICS

If you try to spool too large a file, it will be truncated. *lpr* will object to printing binary files. If a user other than root prints a file and spooling is disabled, *lpr* will print a message saying so and will not put jobs in the queue. If a connection to *lpd* on the local machine cannot be made, *lpr* will say that the daemon cannot be started. Diagnostics may be printed in the daemon's log file regarding missing spool files by *lpd*.

BUGS

Fonts for *troff* and *tex* reside on the host with the printer. It is currently not possible to use local font libraries.

NAME

lprm – remove jobs from the line printer spooling queue

SYNOPSIS

lprm [**–P***printer*] [**–**] [job # ...] [user ...]

DESCRIPTION

lprm will remove a job, or jobs, from a printer's spool queue. Since the spooling directory is protected from users, using *lprm* is normally the only method by which a user may remove a job.

lprm without any arguments will delete the currently active job if it is owned by the user who invoked *lprm*.

If the **–** flag is specified, *lprm* will remove all jobs which a user owns. If the super-user employs this flag, the spool queue will be emptied entirely. The owner is determined by the user's login name and host name on the machine where the *lpr* command was invoked.

Specifying a user's name, or list of user names, will cause *lprm* to attempt to remove any jobs queued belonging to that user (or users). This form of invoking *lprm* is useful only to the super-user.

A user may dequeue an individual job by specifying its job number. This number may be obtained from the *lpq*(1) program, e.g.

```
% lpq –l

1st: ken                    [job #013ucbarpa]
      (standard input)      100 bytes
% lprm 13
```

lprm will announce the names of any files it removes and is silent if there are no jobs in the queue which match the request list.

lprm will kill off an active daemon, if necessary, before removing any spooling files. If a daemon is killed, a new one is automatically restarted upon completion of file removals.

The **–P** option may be usd to specify the queue associated with a specific printer (otherwise the default printer, or the value of the PRINTER variable in the environment is used).

FILES

/etc/printcap	printer characteristics file
/usr/spool/*	spooling directories
/usr/spool/*/lock	lock file used to obtain the pid of the current daemon and the job number of the currently active job

SEE ALSO

lpr(1), lpq(1), lpd(8)

DIAGNOSTICS

"Permission denied" if the user tries to remove files other than his own.

BUGS

Since there are race conditions possible in the update of the lock file, the currently active job may be incorrectly identified.

NAME

ls – list contents of directory

SYNOPSIS

ls [**−acdfgilqrstu1ACLFR**] name ...

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. By default, the output is sorted alphabetically. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments are processed before directories and their contents.

There are a large number of options:

−l List in long format, giving mode, number of hard links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers. If the file is a symbolic link the pathname of the linked-to file is printed preceded by "–>".

−g Include the group ownership of the file in a long output.

−t Sort by time modified (latest first) instead of by name.

−a List all entries; in the absence of this option, entries whose names begin with a period (.) are *not* listed.

−s Give size in kilobytes of each file.

−d If argument is a directory, list only its name; often used with **−l** to get the status of a directory.

−L If argument is a symbolic link, list the file or directory link references rather than the link itself.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−u Use time of last access instead of last modification for sorting (with the **−t** option) and/or printing (with the **−l** option).

−c Use file status change time for sorting or printing.

−i For each file, print the i-number in the first column of the report.

−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.

−F cause directories to be marked with a trailing '/', sockets and fifos with a trailing (|), symbolic links with a trailing '@', and executable files with a trailing '*'.

−R recursively list subdirectories encountered.

−1 force one entry per line output format; this is the default when output is not to a terminal.

−C force multi-column output; this is the default when output is to a terminal.

−q force printing of non-graphic characters in file names as the character '?'; this is the default when output is to a terminal.

The mode printed under the **−l** option contains 11 characters which are interpreted as follows: the first character is

d if the entry is a directory;
b if the entry is a block-type special file;

c if the entry is a character-type special file;
l if the entry is a symbolic link;
s if the entry is a socket, or
p if the entry is a fifo (a.k.a. "named pipe") special files;
– if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next refers to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission is interpreted to mean permission to search the directory. The permissions are indicated as follows:

r if the file is readable;
w if the file is writable;
x if the file is executable;
– if the indicated permission is not granted.

The group-execute permission character is given as **s** if the file has the set-group-id bit set; likewise the user-execute permission character is given as **s** if the file has the set-user-id bit set.

The last character of the mode (normally 'x' or '–') is **t** if the 1000 bit of the mode is on. See *chmod*(1) for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

FILES

/etc/passwd to get user id's for 'ls –l'.
/etc/group to get group id's for 'ls –g'.

BUGS

Newline and tab are considered printing characters in file names.

The output device is assumed to be 80 columns wide unless it is a tty with a nonzero window size, in which case the window size is used.

The option setting based on whether the output is a teletype is undesirable as "ls –s" is much different than "ls –s | lpr". On the other hand, not doing this setting would make old shell scripts which used *ls* almost certain losers.

Due to an oversight, many versions of this document incorrectly state that the **–c** option uses the file creation time. Unix does not store file creation times, so this is impossible.

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

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

nroff − text formatting

SYNOPSIS

nroff [option] ... [file] ...

DESCRIPTION

nroff formats text in the named *files* for typewriter-like devices. See also *troff(1)*. The full capabilities of *nroff* are described in the *nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (**−**) is taken to be a file name corresponding to the standard input.

The options, which may appear in any order so long as they appear *before* the files, are:

−o*list*	Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N***−***M* means pages *N* through *M*; an initial −*N* means from the beginning to page *N*; and a final *N*− means from *N* to the end.
−n*N*	Number first generated page *N*.
−s*N*	Stop every *N* pages. *nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline.
−m*name*	Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.
−r*aN*	Set register *a* (one-character) to *N*.
−i	Read standard input after the input files are exhausted.
−q	Invoke the simultaneous input-output mode of the **rd** request.
−T*name*	Prepare output for specified terminal. Known *names* are **37** for the (default) Teletype Corporation Model 37 terminal, **tn300** for the GE TermiNet 300 (or any terminal without half-line capability), **300S** for the DASI-300S, **300** for the DASI-300, and **450** for the DASI-450 (Diablo Hyterm).
−e	Produce equally-spaced words in adjusted lines, using full terminal resolution.
−h	Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.

FILES

/tmp/ta∗	temporary file
/usr/lib/tmac/tmac.∗	standard macro files
/usr/lib/term/∗	terminal driving tables for *nroff*

SEE ALSO

J. F. Ossanna, *nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*
troff(1), eqn(1), tbl(1), ms(7), me(7), man(7), col(1)

NAME

od – octal, decimal, hex, ascii dump

SYNOPSIS

od [–format] [file] [[**+**]offset[**.**][**b**] [label]]

DESCRIPTION

od displays *file*, or it's standard input, in one or more dump formats as selected by the first argument. If the first argument is missing, **–o** is the default. Dumping continues until end-of-file.

The meanings of the format argument characters are:

a	Interpret bytes as characters and display them with their ACSII names. If the **p** character is given also, then bytes with even parity are underlined. The **P** character causes bytes with odd parity to be underlined. Otherwise the parity bit is ignored.
b	Interpret bytes as unsigned octal.
c	Interpret bytes as ASCII characters. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers. Bytes with the parity bit set are displayed in octal.
d	Interpret (short) words as unsigned decimal.
f	Interpret long words as floating point.
h	Interpret (short) words as unsigned hexadecimal.
i	Interpret (short) words as signed decimal.
l	Interpret long words as signed decimal.
o	Interpret (short) words as unsigned octal.
s[n]	Look for strings of ascii graphic characters, terminated with a null byte. *N* specifies the minimum length string to be recognized. By default, the minimum length is 3 characters.
v	Show all data. By default, display lines that are identical to the last line shown are not output, but are indicated with an "*" in column 1.
w[n]	Specifies the number of input bytes to be interpreted and displayed on each output line. If **w** is not specified, 16 bytes are read for each display line. If *n* is not specified, it defaults to 32.
x	Interpret (short) words as hexadecimal.

An upper case format character implies the long or double precision form of the object.

The *offset* argument specifies the byte offset into the file where dumping is to commence. By default this argument is interpreted in octal. A different radix can be specified; If "**.**" is appended to the argument, then *offset* is interpreted in decimal. If *offset* begins with "**x**" or "**0x**", it is interpreted in hexadecimal. If "**b**" ("**B**") is appended, the offset is interpreted as a block count, where a block is 512 (1024) bytes. If the *file* argument is omitted, an *offset* argument must be preceded by "**+**".

The radix of the displayed address will be the same as the radix of the *offset*, if specified; otherwise it will be octal.

Label will be interpreted as a pseudo-address for the first byte displayed. It will be shown in "()" following the file offset. It is intended to be used with core images to indicate the real memory address. The syntax for *label* is identical to that for *offset*.

SEE ALSO

adb(1)

BUGS

A file name argument can't start with "+". A hexadecimal offset can't be a block count. Only one file name argument can be given.

It is an historical botch to require specification of object, radix, and sign representation in a single character argument.

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

pagesize – print system page size

SYNOPSIS

pagesize

DESCRIPTION

pagesize prints the size of a page of memory in bytes, as returned by *getpagesize*(2). This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(2)

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), fto

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.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

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*(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.

−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.

−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.

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 file

SYNOPSIS
pr [option] ... [file] ...

DESCRIPTION
pr produces a printed listing of one or more *files.* The output is separated into pages headed by the date that the file was last modified (or the current date if standard input is used), the name of the file or a specified header, and the page number. If there are no file arguments, *pr* prints its standard input.

Options apply to all following files but may be reset between files:

–*n*	produce *n*-column output.
+*n*	Begin printing with page *n*.
–h	Take the next argument as a page header.
–w*n*	For purposes of multi-column output, take the width of the page to be *n* characters instead of the default 72.
–f	Use formfeeds instead of newlines to separate pages. A formfeed is assumed to use up two blank lines at the top of a page. (Thus this option does not affect the effective page length.)
–l*n*	Take the length of the page to be *n* lines instead of the default 66.
–t	Do not print the 5-line header or the 5-line trailer normally supplied for each page.
–s*c*	Separate columns by the single character *c* instead of by the appropriate amount of white space. A missing *c* is taken to be a tab.
–m	print all *files* simultaneously, each in one column,
–b	Break up lines wider than the page width. This is useful for printing with printers and window systems that do not wrap long lines.

Inter-terminal messages via *write*(1) are forbidden during a *pr*.

FILES
/dev/tty? to suspend messages.

SEE ALSO
cat(1)

DIAGNOSTICS
There are no diagnostics when *pr* is printing on a terminal.

NAME

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

ptx – permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *ptx* produces output in the form:

```
.xx "tail" "before keyword" "keyword and after" "head"
```

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head,* at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

−f Fold upper and lower case letters for sorting.

−t Prepare the output for the phototypesetter; the default line length is 100 characters.

−w *n* Use the next argument, *n,* as the width of the output line. The default line length is 72 characters.

−g *n* Use the next argument, *n,* as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.

−o only
Use as keywords only the words given in the *only* file.

−i ignore
Do not use as keywords any words given in the *ignore* file. If the **−i** and **−o** options are missing, use */usr/lib/eign* as the *ignore* file.

−b break
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.

−r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx.*

FILES

/usr/bin/sort
/usr/lib/eign

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

ratfor – rational Fortran dialect

SYNOPSIS

ratfor [option ...] [filename ...]

DESCRIPTION

ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *ratfor* provides control flow constructs essentially identical to those in C:

```
statement grouping:
        { statement; statement; statement }

decision-making:
        if (condition) statement [ else statement ]
        switch (integer value) {
                case integer:    statement
                ...
                [ default: ]     statement
        }

loops:  while (condition) statement
        for (expression; condition; expression) statement
        do limits statement
        repeat statement [ until (condition) ]
        break
        next
```

and some syntactic sugar to make programs easier to read and write:

```
free form input:
        multiple statements/line; automatic continuation

comments:
        # this is a comment

translation of relationals:
        >, >=, etc., become .GT., .GE., etc.

return (expression)
        returns expression to caller from function

define: define name replacement

include:
        include filename
```

ratfor is best used with *f77*(1).

SEE ALSO

f77(1)
B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

refer – find and insert literature references in documents

SYNOPSIS

refer [**−a**] [**−b**] [**−c**] [**−e**] [**−f***n*] [**−k***x*] [**−l***m,n*] [**−n**] [**−p** bib] [**−s***keys*] [**−B***l.m*] [**−P**] [**−S**] [file ...]

DESCRIPTION

refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references for footnotes or endnotes. It is also the base for a series of programs designed to index, search, sort, and print stand-alone bibliographies, or other data entered in the appropriate form.

Given an incomplete citation with sufficiently precise keywords, *refer* will search a bibliographic database for references containing these keywords anywhere in the title, author, journal, etc. The input file (or standard input) is copied to standard output, except for lines between .[and .] delimiters, which are assumed to contain keywords, and are replaced by information from the bibliographic database. The user may also search different databases, override particular fields, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. By default references are flagged by footnote numbers.

The following options are available:

−a*n* Reverse the first *n* author names (Jones, J. A. instead of J. A. Jones). If *n* is omitted all author names are reversed.

−b Bare mode: do not put any flags in text (neither numbers nor labels).

−c*keys*
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *keys*.

−e Instead of leaving the references where encountered, accumulate them until a sequence of the form

```
.[
$LIST$
.]
```

is encountered, and then write out all references collected so far. Collapse references to same source.

−f*n* Set the footnote number to *n* instead of the default of 1 (one). With labels rather than numbers, this flag is a no-op.

−k*x* Instead of numbering references, use labels as specified in a reference data line beginning *%x;* by default *x* is **L**.

−l*m,n*
Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.

−n Do not search the default file /usr/dict/papers/Ind. If there is a REFER environment variable, the specified file will be searched instead of the default file; in this case the **−n** flag has no effect.

−p *bib*
Take the next argument *bib* as a file of references to be searched. The default file is searched last.

−s*keys*
Sort references by fields whose key-letters are in the *keys* string; permute reference

numbers in text accordingly. Implies **−e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with **+** taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title, use **-sA+T**.

−B*l.m*
Bibliography mode. Take a file composed of records separated by blank lines, and turn them into *troff* input. Label *l* will be turned into the macro *.m* with *l* defaulting to **%X** and *.m* defaulting to **.AP** (annotation paragraph).

−P Place punctuation marks .,:;?! after the reference signal, rather than before. (Periods and commas used to be done with strings.)

−S Produce references in the Natural or Social Science format.

To use your own references, put them in the format described below. They can be searched more rapidly by running *indxbib*(1) on them before using *refer;* failure to index results in a linear search. When *refer* is used with the *eqn, neqn* or *tbl* preprocessors *refer* should be first, to minimize the volume of data passed through pipes.

The *refer* preprocessor and associated programs expect input from a file of references composed of records separated by blank lines. A record is a set of lines (fields), each containing one kind of information. Fields start on a line beginning with a "%", followed by a key-letter, then a blank, and finally the contents of the field, and continue until the next line starting with "%". The output ordering and formatting of fields is controlled by the macros specified for *nroff/troff* (for footnotes and endnotes) or *roffbib* (for stand-alone bibliographies). For a list of the most common key-letters and their corresponding fields, see *addbib*(1). An example of a *refer* entry is given below.

EXAMPLE

```
%A   M. E. Lesk
%T   Some Applications of Inverted Indexes on the UNIX System
%B   UNIX Programmer's Manual
%V   2b
%I   Bell Laboratories
%C   Murray Hill, NJ
%D   1978
```

FILES

/usr/dict/papers	directory of default publication lists
/usr/lib/refer	directory of companion programs

SEE ALSO

addbib(1), sortbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHOR

Mike Lesk

BUGS

Blank spaces at the end of lines in bibliography fields will cause the records to sort and reverse incorrectly. Sorting large numbers of references causes a core dump.

NAME

rev – reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

NAME

rm, rmdir – remove (unlink) files or directories

SYNOPSIS

rm [**−f**] [**−r**] [**−i**] [**−**] file ...

rmdir dir ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with 'y' the file is deleted, otherwise the file remains. No questions are asked and no errors are reported when the **−f** (force) option is given.

If a designated file is a directory, an error comment is printed unless the optional argument **−r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **−i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **−r**, whether to examine each directory.

The null option **−** indicates that all the arguments following it are to be treated as file names. This allows the specification of file names starting with a minus.

rmdir removes entries for the named directories, which must be empty.

SEE ALSO

rm(1), unlink(2), rmdir(2)

NAME

roffbib – run off bibliographic database

SYNOPSIS

roffbib [**−e**] [**−h**] [**−n**] [**−o**] [**−r**] [**−s**] [**−T***term*] [**−x**] [**−m** *mac*] [**−V**] [**−Q**] [file ...]

DESCRIPTION

roffbib prints out all records in a bibliographic database, in bibliography format rather than as footnotes or endnotes. Generally it is used in conjunction with *sortbib*:

```
sortbib  database | roffbib
```

roffbib accepts most of the options understood by *nroff*(1), most importantly the **−T** flag to specify terminal type.

If abstracts or comments are entered following the %X field key, *roffbib* will format them into paragraphs for an annotated bibliography. Several %X fields may be given if several annotation paragraphs are desired. The **−x** flag will suppress the printing of these abstracts.

A user-defined set of macros may be specified after the **−m** option. There should be a space between the **−m** and the macro filename. This set of macros will replace the ones defined in /usr/lib/tmac/tmac.bib. The **−V** flag will send output to the Versatec; the **−Q** flag will queue output for the phototypesetter.

Four command-line registers control formatting style of the bibliography, much like the number registers of *ms*(7). The command-line argument **−r**N1 will number the references starting at one (1). The flag **−r**V2 will double space the bibliography, while **−r**V1 will double space references but single space annotation paragraphs. The line length can be changed from the default 6.5 inches to 6 inches with the **−r**L6i argument, and the page offset can be set from the default of 0 to one inch by specifying **−r**O1i (capital O, not zero). Note: with the **−V** and **−Q** flags the default page offset is already one inch.

FILES

/usr/lib/tmac/tmac.bib   file of macros used by *nroff/troff*

SEE ALSO

refer(1), addbib(1), sortbib(1), indxbib(1), lookbib(1)

BUGS

Users have to rewrite macros to create customized formats.

NAME

sccs – front end for the SCCS subsystem

SYNOPSIS

sccs [**−r**] [**−d***path*] [**−p***path*] command [flags] [args]

DESCRIPTION

sccs is a front end to the SCCS programs that helps them mesh more cleanly with the rest of UNIX. It also includes the capability to run "set user id" to another user to provide additional protection.

Basically, *sccs* runs the *command* with the specified *flags* and *args*. Each argument is normally modified to be prepended with "SCCS/s.".

Flags to be interpreted by the *sccs* program must be before the *command* argument. Flags to be passed to the actual SCCS program must come after the *command* argument. These flags are specific to the command and are discussed in the documentation for that command.

Besides the usual SCCS commands, several "pseudo-commands" can be issued. These are:

edit	Equivalent to "get −e".
delget	Perform a delta on the named files and then get new versions. The new versions will have id keywords expanded, and will not be editable. The −m, −p, −r, −s, and −y flags will be passed to delta, and the −b, −c, −e, −i, −k, −l, −s, and −x flags will be passed to get.
deledit	Equivalent to "delget" except that the "get" phase includes the "−e" flag. This option is useful for making a "checkpoint" of your current editing phase. The same flags will be passed to delta as described above, and all the flags listed for "get" above except −e and −**k** are passed to "edit".
create	Creates an SCCS file, taking the initial contents from the file of the same name. Any flags to "admin" are accepted. If the creation is successful, the files are renamed with a comma on the front. These should be removed when you are convinced that the SCCS files have been created successfully.
fix	Must be followed by a **−r** flag. This command essentially removes the named delta, but leaves you with a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, etc. Since it doesn't leave audit trails, it should be used carefully.
clean	This routine removes everything from the current directory that can be recreated from SCCS files. It will not remove any files being edited. If the **−b** flag is given, branches are ignored in the determination of whether they are being edited; this is dangerous if you are keeping the branches in the same directory.
unedit	This is the opposite of an "edit" or a "get −e". It should be used with extreme caution, since any changes you made since the get will be irretrievably lost.
info	Gives a listing of all files being edited. If the **−b** flag is given, branches (i.e., SID's with two or fewer components) are ignored. If the **−u** flag is given (with an optional argument) then only files being edited by you (or the named user) are listed.
check	Like "info" except that nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. The intent is to have this included in an "install" entry in a makefile to insure that everything is included into the SCCS file before a version is installed.

tell	Gives a newline-separated list of the files being edited on the standard output. Takes the **−b** and **−u** flags like "info" and "check".
diffs	Gives a "diff" listing between the current version of the program(s) you have out for editing and the versions in SCCS format. The **−r, −c, −i, −x,** and **−t** flags are passed to *get*; the **−l, −s, −e, −f, −h,** and **−b** options are passed to *diff*. The **−C** flag is passed to *diff* as **−c.**
print	This command prints out verbose information about the named files.

The **−r** flag runs *sccs* as the real user rather than as whatever effective user *sccs* is "set user id" to. The **−d** flag gives a root directory for the SCCS files. The default is the current directory. The **−p** flag defines the pathname of the directory in which the SCCS files will be found; "SCCS" is the default. The **−p** flag differs from the **−d** flag in that the **−d** argument is prepended to the entire pathname and the **−p** argument is inserted before the final component of the pathname. For example, "sccs −d/x −py get a/b" will convert to "get /x/a/y/s.b". The intent here is to create aliases such as "alias syssccs sccs -d/usr/src" which will be used as "syssccs get cmd/who.c". Also, if the environment variable PROJECT is set, its value is used to determine the **−d flag.** If it begins with a slash, it is taken directly; otherwise, the home directory of a user of that name is examined for a subdirectory "src" or "source". If such a directory is found, it is used.

Certain commands (such as *admin*) cannot be run "set user id" by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

EXAMPLES

To get a file for editing, edit it, and produce a new delta:

```
sccs get −e file.c
ex file.c
sccs delta file.c
```

To get a file from another directory:

```
sccs −p/usr/src/sccs/s. get cc.c
```

or

```
sccs get /usr/src/sccs/s.cc.c
```

To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

To get a list of files being edited that are not on branches:

```
sccs info −b
```

To delta everything being edited by you:

```
sccs delta `sccs tell −u`
```

In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
        sccs get $(REL) $@
```

SEE ALSO

admin(SCCS), chghist(SCCS), comb(SCCS), delta(SCCS), get(SCCS), help(SCCS), prt(SCCS), rmdel(SCCS), sccsdiff(SCCS), what(SCCS)
Eric Allman, *An Introduction to the Source Code Control System*

BUGS

It should be able to take directory arguments on pseudo-commands like the SCCS commands do.

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

soelim – eliminate .so's from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

```
.so somefile
```

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (**–**) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using '´' instead of '.', i.e.

```
´so /usr/lib/tmac.s
```

A sample usage of *soelim* would be

```
soelim exum?.n | tbl | nroff –ms | col | lpr
```

SEE ALSO

colcrt(1), more(1)

BUGS

The format of the source commands must involve no strangeness – exactly one blank must precede and no blanks follow the file name.

NAME

sort – sort or merge files

SYNOPSIS

sort [**−mubdfinrt***x*] [**+***pos1* [**−***pos2*]] ... [**−o** name] [**−T** directory] [name] ...

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The name '−' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

b Ignore leading blanks (spaces and tabs) in field comparisons.

d 'Dictionary' order: only letters, digits and blanks are significant in comparisons.

f Fold upper case letters onto lower case.

i Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.

n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b.**

r Reverse the sense of comparisons.

t*x* 'Tab character' separating fields is *x*.

The notation **+***pos1* **−***pos2* restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr,** where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *.n* means .0; a missing **−***pos2* means the end of the line. Under the **−t***x* option, fields are strings separated by *x*; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

m Merge only, the input files are already sorted.

o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.

T The next argument is the name of a directory in which temporary files should be made.

u Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

EXAMPLES

Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort −u +0f +0 list
```

Print the password file (*passwd*(5)) sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

FILES

/usr/tmp/stm*, /tmp/* first and second tries for temporary files

SEE ALSO

uniq(1), comm(1), rev(1), join(1)

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option **−c**.

BUGS

Very long lines (>1023 characters) are silently truncated.

NAME

sortbib – sort bibliographic database

SYNOPSIS

sortbib [**−s**KEYS] database ...

DESCRIPTION

sortbib sorts files of records containing *refer* key-letters by user-specified keys. Records may be separated by blank lines, or by .[and .] delimiters, but the two styles may not be mixed together. This program reads through each *database* and pulls out key fields, which are sorted separately. The sorted key fields contain the file pointer, byte offset, and length of corresponding records. These records are delivered using disk seeks and reads, so *sortbib* may not be used in a pipeline to read standard input.

By default, *sortbib* alphabetizes by the first %A and the %D fields, which contain the senior author and date. The **−s** option is used to specify new *KEYS*. For instance, **−s**ATD will sort by author, title, and date, while **−s**A+D will sort by all authors, and date. Sort keys past the fourth are not meaningful. No more than 16 databases may be sorted together at one time. Records longer than 4096 characters will be truncated.

sortbib sorts on the last word on the %A line, which is assumed to be the author's last name. A word in the final position, such as "jr." or "ed.", will be ignored if the name beforehand ends with a comma. Authors with two-word last names or unusual constructions can be sorted correctly by using the *nroff* convention "\0" in place of a blank. A %Q field is considered to be the same as %A, except sorting begins with the first, not the last, word. *sortbib* sorts on the last word of the %D line, usually the year. It also ignores leading articles (like "A" or "The") when sorting by titles in the %T or %J fields; it will ignore articles of any modern European language. If a sort-significant field is absent from a record, *sortbib* places that record before other records containing that field.

SEE ALSO

refer(1), addbib(1), roffbib(1), indxbib(1), lookbib(1)

AUTHORS

Greg Shenaut, Bill Tuthill

BUGS

Records with missing author fields should probably be sorted by title.

NAME

spell, spellin, spellout – find spelling errors

SYNOPSIS

spell [**−v**] [**−b**] [**−x**] [**−d** hlist] [**−s** hstop] [**−h** spellhist] [file] ...

spellin [list]

spellout [**−d**] list

DESCRIPTION

spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

spell ignores most *troff, tbl* and *eqn*(1) constructions.

Under the **−v** option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the **−b** option, British spelling is checked. Besides preferring *centre, colour, speciality, travelled,* etc., this option insists upon *-ise* in words like *standardise,* Fowler and the OED to the contrary notwithstanding.

Under the **−x** option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources. While it is more haphazard than an ordinary dictionary, it is also more effective with proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

The auxiliary files used for the spelling list, stop list, and history file may be specified by arguments following the **−d**, **−s**, and **−h** options. The default files are indicated below. Copies of all output may be accumulated in the history file. The stop list filters out misspellings (e.g. thier=thy−y+ier) that would otherwise pass.

Two routines help maintain the hash lists used by *spell.* Both expect a set of words, one per line, from the standard input. *spellin* combines the words from the standard input and the preexisting *list* file and places a new list on the standard output. If no *list* file is specified, the new list is created from scratch. *spellout* looks up each word from the standard input and prints on the standard output those that are missing from (or present on, with option **−d**) the hashed *list* file. For example, to verify that *hookey* is not on the default spelling list, add it to your own private list, and then use it with *spell*,

```
echo  hookey  |  spellout  /usr/dict/hlista
echo  hookey  |  spellin  /usr/dict/hlista  >  myhlist
spell  −d  myhlist  huckfinn
```

FILES

/usr/dict/hlist[ab]	hashed spelling lists, American & British, default for **−d**
/usr/dict/hstop	hashed stop list, default for **−s**
/dev/null	history file, default for **−h**
/tmp/spell.$$*	temporary files
/usr/lib/spell	

SEE ALSO

deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. Spellout is limited to a maximum of 30 characters per word - words greater than 30 characters are broken and treated as two words. British spelling was done by an American.

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 terminal options

SYNOPSIS

stty [option ...]

DESCRIPTION

stty sets certain I/O options on the current output terminal, placing its output on the diagnostic output. With no argument, it reports the speed of the terminal and the settings of the options which are different from their defaults. Use of one of the following options modifies the output as described:

all	All normally used option settings are reported.
everything	Everything *stty* knows about is printed.
speed	The terminal speed alone is printed on the standard output.
size	The terminal (window) sizes are printed on the standard output, first rows and then columns.

The option strings are selected from the following set:

even	allow even parity input
−even	disallow even parity input
odd	allow odd parity input
−odd	disallow odd parity input
raw	raw mode input (**no input processing** (erase, kill, interrupt, ...); parity bit passed back)
−raw	negate raw mode
cooked	same as '−raw'
cbreak	make each character available to *read*(2) as received; no erase and kill processing, but all other processing (interrupt, suspend, ...) is performed
−cbreak	make characters available to *read* only when newline is received
−nl	allow carriage return for new-line, and output CR-LF for carriage return or new-line
nl	accept only new-line to end lines
echo	echo back every character typed
−echo	do not echo characters
lcase	map upper case to lower case
−lcase	do not map case
tandem	enable flow control, so that the system sends out the stop character when its internal queue is in danger of overflowing on input, and sends the start character when it is ready to accept further input
−tandem	disable flow control
−tabs	replace tabs by spaces when printing
tabs	preserve tabs
ek	set erase and kill characters to # and @

For the following commands which take a character argument *c*, you may also specify *c* as the "u" or "undef", to set the value to be undefined. A value of "^x", a 2 character sequence, is also interpreted as a control character, with "^?" representing delete.

erase *c*	set erase character to *c* (default '#', but often reset to ^H.)
kill *c*	set kill character to *c* (default '@', but often reset to ^U.)
intr *c*	set interrupt character to *c* (default DEL or ^? (delete), but often reset to ^C.)
quit *c*	set quit character to *c* (default control \\.)
start *c*	set start character to *c* (default control Q.)

stop *c*	set stop character to *c* (default control S.)
eof *c*	set end of file character to *c* (default control D.)
brk *c*	set break character to *c* (default undefined.) This character is an additional character causing wakeup.
cr0 cr1 cr2 cr3	select style of delay for carriage return (see *ioctl*(2))
nl0 nl1 nl2 nl3	select style of delay for linefeed
tab0 tab1 tab2 tab3	select style of delay for tab
ff0 ff1	select style of delay for form feed
bs0 bs1	select style of delay for backspace
tty33	set all modes suitable for the Teletype Corporation Model 33 terminal.
tty37	set all modes suitable for the Teletype Corporation Model 37 terminal.
vt05	set all modes suitable for Digital Equipment Corp. VT05 terminal
dec	set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decctlq and "newcrt".)
tn300	set all modes suitable for a General Electric TermiNet 300
ti700	set all modes suitable for Texas Instruments 700 series terminal
tek	set all modes suitable for Tektronix 4014 terminal
0	hang up phone line immediately
50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb	Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).
rows *n*	The terminal size is recorded as having *n* rows.
columns *n*	The terminal size is recorded as having *n* columns.
cols *n*	is an alias for *columns*.

A teletype driver which supports the job control processing of *csh*(1) and more functionality than the basic driver is fully described in *tty*(4). The following options apply only to it.

new	Use new driver (switching flushes typeahead).
crt	Set options for a CRT (crtbs, ctlecho and, if >= 1200 baud, crterase and crtkill.)
crtbs	Echo backspaces on erase characters.
prterase	For printing terminal echo erased characters backwards within "\" and "/".
crterase	Wipe out erased characters with "backspace-space-backspace."
−crterase	Leave erased characters visible; just backspace.
crtkill	Wipe out input on like kill ala **crterase.**
−crtkill	Just echo line kill character and a newline on line kill.
ctlecho	Echo control characters as "^*x*" (and delete as "^?".) Print two backspaces following the EOT character (control D).
−ctlecho	Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.
decctlq	After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.
−decctlq	After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop	Background jobs stop if they attempt terminal output.
−tostop	Output from background jobs to the terminal is allowed.
tilde	Convert "~" to "`" on output (for Hazeltine terminals).

−tilde	Leave poor "~" alone.
flusho	Output is being discarded usually because user hit control O (internal state bit).
−flusho	Output is not being discarded.
pendin	Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
−pendin	Input is not pending.
pass8	Passes all 8 bits through on input, in any mode.
−pass8	Strips the 0200 bit on input except in raw mode.
mdmbuf	Start/stop output on carrier transitions (not implemented).
−mdmbuf	Return error if write attempted after carrier drops.
litout	Send output characters without any processing.
−litout	Do normal output processing, inserting delays, etc.
nohang	Don't send hangup signal if carrier drops.
−nohang	Send hangup signal to control process group when carrier drops.
etxack	Diablo style etx/ack handshaking (not implemented).

The following special characters are applicable only to the new teletype driver and are not normally changed.

susp *c*	set suspend process character to *c* (default control Z).
dsusp *c*	set delayed suspend process character to *c* (default control Y).
rprnt *c*	set reprint line character to *c* (default control R).
flush *c*	set flush output character to *c* (default control O).
werase *c*	set word erase character to *c* (default control W).
lnext *c*	set literal next character to *c* (default control V).

SEE ALSO

ioctl(2), tabs(1), tset(1), tty(4)

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 (see below), the user environment is unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(7)). The new user ID stays in force until the Shell exits.

If no *userid* is specified, "root" is assumed. Only users in the "wheel" 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 wheel 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 wheel (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).

FILES

/etc/su_people	Special permission database

SEE ALSO

sh(1), csh(1), su_people(5)

NAME

talk – talk to another user

SYNOPSIS

talk person [ttyname]

DESCRIPTION

talk is a visual communication program which copies lines from your terminal to that of another user.

If you wish to talk to someone on you own machine, then *person* is just the person's login name. If you wish to talk to a user on another host, then *person* is of the form :

host!user or
host.user or
host:user or
user@host

though *host@user* is perhaps preferred.

If you want to talk to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

When first called, it sends the message

```
Message from talk.Daemon@his_machine...
talk: connection requested by your_name@your_machine.
talk: respond with: talk your_name@your_machine
```

to the user you wish to talk to. At this point, the recipient of the message should reply by typing

```
talk  your_name@your_machine
```

It doesn't matter from which machine the recipient replies, as long as his login-name is the same. Once communication is established, the two parties may type simultaneously, with their output appearing in separate windows. Typing control L will cause the screen to be reprinted, while your erase, kill, and word kill characters will work in talk as normal. To exit, just type your interrupt character; *talk* then moves the cursor to the bottom of the screen and restores the terminal.

Permission to talk may be denied or granted by use of the *mesg* command. At the outset talking is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

FILES

/etc/hosts	to find the recipient's machine
/etc/utmp	to find the recipient's tty

SEE ALSO

mesg(1), who(1), mail(1), write(1)

BUGS

The version of *talk*(1) released with 4.3BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD.

NAME

tbl – format tables for nroff or troff

SYNOPSIS

tbl [files] ...

DESCRIPTION

tbl is a preprocessor for formatting tables for *nroff* or *troff*(1). The input files are copied to the standard output, except for lines between and are reformatted. Details are given in the *tbl*(1) reference manual.

EXAMPLE

As an example, letting \t represent a tab (which should be typed as a genuine tab) the input

```
.TS
c s s
c c s
c c c
l n n.
Household Population
Town\tHouseholds
\tNumber\tSize
Bedminster\t789\t3.26
Bernards Twp.\t3087\t3.74
Bernardsville\t2018\t3.30
Bound Brook\t3425\t3.04
Branchburg\t1644\t3.49
Bridgewater\t7897\t3.81
Far Hills\t240\t3.19
.TE
```

yields

```
   Household Population
   Town        Households
             Number   Size
Bedminster      789   3.26
Bernards Twp.  3087   3.74
Bernardsville  2018   3.30
Bound Brook    3425   3.04
Branchburg     1644   3.49
Bridgewater    7897   3.81
Far Hills       240   3.19
```

If no arguments are given, *tbl* reads the standard input, so it may be used as a filter. When *tbl* is used with *eqn* or *neqn* the *tbl* command should be first, to minimize the volume of data passed through pipes.

SEE ALSO

troff(1), eqn(1)
M. E. Lesk, *TBL.*

NAME

tc – photoypesetter simulator

SYNOPSIS

tc [**−t**] [**−s**N] [**−p**L] [file]

DESCRIPTION

tc interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of *tc* is intended for a Tektronix 4015 (a 4014 terminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

```
troff −t file | tc
```

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; sN will cause the next N pages to be skipped; and !line will send line to the shell.

The command line options are:

−t Don't wait between pages; for directing output into a file.

−sN Skip the first N pages.

−pL Set page length to L. L may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.

'−*l w*' Multiply the default aspect ratio, 1.5, of a displayed page by *l/w*.

SEE ALSO

troff(1), plot(1G)

BUGS

Font distinctions are lost.
tc's character set is limited to ASCII in just one size.
The aspect ratio option is unbelievable.

NAME
tcopy – copy a mag tape

SYNOPSIS
tcopy src [**dest**]

DESCRIPTION
tcopy is designed to copy magnetic tapes. The only assumption made about the tape is that there are two tape marks at the end. *tcopy* with only a source tape specified will print information about the sizes of records and tape files. If a destination is specified, then, a copy will be made of the source tape. The blocking on the destination tape will be identical to that used on the source tape. Copying a tape will yield the same output as if just printing the sizes.

SEE ALSO
mtio(4)

NAME

tip, cu – connect to a remote system

SYNOPSIS

tip [**−v**] [**−***speed*] system-name
tip [**−v**] [**−***speed*] phone-number
cu phone-number [**−t**] [**−s** *speed*] [**-a** *acu*] [**−l** *line*] [**−#**]

DESCRIPTION

tip and *cu* establish a full-duplex connection to another machine, giving the appearance of being logged in directly on the remote cpu. It goes without saying that you must have a login on the machine (or equivalent) to which you wish to connect. The preferred interface is *tip*. The *cu* interface is included for those people attached to the "call UNIX" command of version 7. This manual page describes only *tip*.

Typed characters are normally transmitted directly to the remote machine (which does the echoing as well). A tilde ('~') appearing as the first character of a line is an escape signal; the following are recognized:

~^D ~.	Drop the connection and exit (you may still be logged in on the remote machine).
~c [*name*]	Change directory to name (no argument implies change to your home directory).
~!	Escape to a shell (exiting the shell will return you to tip).
~>	Copy file from local to remote. *tip* prompts for the name of a local file to transmit.
~<	Copy file from remote to local. *tip* prompts first for the name of the file to be sent, then for a command to be executed on the remote machine.

~p *from* [*to*]
Send a file to a remote UNIX host. The put command causes the remote UNIX system to run the command string "cat > 'to'", while *tip* sends it the "from" file. If the "to" file isn't specified the "from" file name is used. This command is actually a UNIX specific version of the "~>" command.

~t *from* [*to*]
Take a file from a remote UNIX host. As in the put command the "to" file defaults to the "from" file name if it isn't specified. The remote host executes the command string "cat 'from';echo ^A" to send the file to *tip*.

~|	Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.
~$	Pipe the output from a local UNIX process to the remote host. The command string sent to the local UNIX system is processed by the shell.
~#	Send a BREAK to the remote system. For systems which don't support the necessary *ioctl* call the break is simulated by a sequence of line speed changes and DEL characters.
~s	Set a variable (see the discussion below).
~^Z	Stop *tip* (only available with job control).
~^Y	Stop only the "local side" of *tip* (only available with job control); the "remote side" of *tip*, the side that displays output from the remote host, is left running.
~?	Get a summary of the tilde escapes

tip uses the file /etc/remote to find how to reach a particular system and to find out how it should operate while talking to the system; refer to *remote*(5) for a full description. Each system has a default baud rate with which to establish a connection. If this value is not suitable, the baud rate to be used may be specified on the command line, e.g. "tip -300 mds".

When *tip* establishes a connection it sends out a connection message to the remote system; the default value, if any, is defined in /etc/remote.

When *tip* prompts for an argument (e.g. during setup of a file transfer) the line typed may be edited with the standard erase and kill characters. A null line in response to a prompt, or an interrupt, will abort the dialogue and return you to the remote machine.

tip guards against multiple users connecting to a remote system by opening modems and terminal lines with exclusive access, and by honoring the locking protocol used by *uucp*(1C).

During file transfers *tip* provides a running count of the number of lines transferred. When using the ~> and ~< commands, the "eofread" and "eofwrite" variables are used to recognize end-of-file when reading, and specify end-of-file when writing (see below). File transfers normally depend on tandem mode for flow control. If the remote system does not support tandem mode, "echocheck" may be set to indicate *tip* should synchronize with the remote system on the echo of each transmitted character.

When *tip* must dial a phone number to connect to a system it will print various messages indicating its actions. *tip* supports the DEC DN-11 and Racal-Vadic 831 auto-call-units; the DEC DF02 and DF03, Ventel 212+, Racal-Vadic 3451, Bizcomp 1031 and 1032, Hayes SmartModem, and Concord Data Systems 224 integral call unit/modems.

VARIABLES

tip maintains a set of *variables* which control its operation. Some of these variable are read-only to normal users (root is allowed to change anything of interest). Variables may be displayed and set through the "s" escape. The syntax for variables is patterned after *vi*(1) and *Mail*(1). Supplying "all" as an argument to the set command displays all variables readable by the user. Alternatively, the user may request display of a particular variable by attaching a '?' to the end. For example "escape?" displays the current escape character.

Variables are numeric, string, character, or boolean values. Boolean variables are set merely by specifying their name; they may be reset by prepending a '!' to the name. Other variable types are set by concatenating an '=' and the value. The entire assignment must not have any blanks in it. A single set command may be used to interrogate as well as set a number of variables. Variables may be initialized at run time by placing set commands (without the "~s" prefix in a file *.tiprc* in one's home directory). The **−v** option causes *tip* to display the sets as they are made. Certain common variables have abbreviations. The following is a list of common variables, their abbreviations, and their default values.

beautify
(bool) Discard unprintable characters when a session is being scripted; abbreviated *be*.

baudrate
(num) The baud rate at which the connection was established; abbreviated *ba*.

dialtimeout
(num) When dialing a phone number, the time (in seconds) to wait for a connection to be established; abbreviated *dial*.

echocheck
(bool) Synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted; default is *off*.

eofread
(str) The set of characters which signify and end-of-tranmission during a ~< file transfer command; abbreviated *eofr*.

eofwrite

(str) The string sent to indicate end-of-transmission during a ~> file transfer command; abbreviated *eofw*.

eol

(str) The set of characters which indicate an end-of-line. *tip* will recognize escape characters only after an end-of-line.

escape

(char) The command prefix (escape) character; abbreviated *es*; default value is '~'.

exceptions

(str) The set of characters which should not be discarded due to the beautification switch; abbreviated *ex*; default value is "\t\n\f\b".

force

(char) The character used to force literal data transmission; abbreviated *fo*; default value is '^P'.

framesize

(num) The amount of data (in bytes) to buffer between file system writes when receiving files; abbreviated *fr*.

host

(str) The name of the host to which you are connected; abbreviated *ho*.

prompt

(char) The character which indicates and end-of-line on the remote host; abbreviated *pr*; default value is '\n'. This value is used to synchronize during data transfers. The count of lines transferred during a file transfer command is based on recipt of this character.

raise

(bool) Upper case mapping mode; abbreviated *ra*; default value is *off*. When this mode is enabled, all lower case letters will be mapped to upper case by *tip* for transmission to the remote machine.

raisechar

(char) The input character used to toggle upper case mapping mode; abbreviated *rc*; default value is '^A'.

record

(str) The name of the file in which a session script is recorded; abbreviated *rec*; default value is "tip.record".

script

(bool) Session scripting mode; abbreviated *sc*; default is *off*. When *script* is *true*, *tip* will record everything transmitted by the remote machine in the script record file specified in *record*. If the *beautify* switch is on, only printable ASCII characters will be included in the script file (those characters betwee 040 and 0177). The variable *exceptions* is used to indicate characters which are an exception to the normal beautification rules.

tabexpand

(bool) Expand tabs to spaces during file transfers; abbreviated *tab*; default value is *false*. Each tab is expanded to 8 spaces.

verbose

(bool) Verbose mode; abbreviated *verb*; default is *true*. When verbose mode is enabled, *tip* prints messages while dialing, shows the current number of lines transferred during a file transfer operations, and more.

SHELL

(str) The name of the shell to use for the ~! command; default value is "/bin/sh", or taken from the environment.

HOME

(str) The home directory to use for the ~c command; default value is taken from the environment.

FILES

/etc/remote	global system descriptions
/etc/phones	global phone number data base
${REMOTE}	private system descriptions
${PHONES}	private phone numbers
~/.tiprc	initialization file.
/usr/spool/locks/LCK..*	lock file to avoid conflicts with *uucp*

DIAGNOSTICS

Diagnostics are, hopefully, self explanatory.

SEE ALSO

remote(5), phones(5)

BUGS

The full set of variables is undocumented and should, probably, be paired down.

NAME

tk − paginator for the Tektronix 4014

SYNOPSIS

tk [**−t**] [**−***N*] [**−p***L*] [file]

DESCRIPTION

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

The command line options are:

−t	Don't wait between pages; for directing output into a file.
−*N*	Divide the screen into *N* columns and wait after the last column.
−p*L*	Set page length to *L* lines.

SEE ALSO

pr(1)

NAME

tr – translate characters

SYNOPSIS

tr [**−cds**] [string1 [string2]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options **−cds** may be used: **−c** complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; **−d** deletes all input characters in *string1;* **−s** squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a***−***b* means a range of characters from *a* to *b* in increasing ASCII order. The character '\' followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A '\' followed by any other character stands for that character.

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabetics. The second string is quoted to protect '\' from the Shell. 012 is the ASCII code for newline.

```
tr −cs A−Za−z '\012' <file1 >file2
```

SEE ALSO

ed(1), ascii(7), expand(1)

BUGS

Won't handle ASCII NUL in *string1* or *string2;* always deletes NUL from input.

NAME

touch – update date last modified of a file

SYNOPSIS

touch [**−c**] [**−f**] file ...

DESCRIPTION

touch attempts to set the modified date of each *file*. If a *file* exists, this is done by reading a character from the file and writing it back. If a *file* does not exist, an attempt will be made to create it unless the **−c** option is specified. The **−f** option will attempt to force the touch in spite of read and write permissions on a *file*.

SEE ALSO

utimes(2)

NAME

tsort – topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**−udc** [**+**n] [**−**n]] [input [output]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **−u** flag is used, just the lines that are not repeated in the original file are output. The **−d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **−u** and **−d** mode outputs.

The **−c** option supersedes **−u** and **−d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

−*n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), comm(1)

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch
You want: cm
  ** 2.54000e+00*
  / 3.93701e–01

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 pounds force/in2
You want: atm
  ** 1.02069e+00*
  / 9.79730e–01

units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

NAME

w – who is on and what they are doing

SYNOPSIS

w [**−h**] [**−s**] [user]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The **−h** flag suppresses the heading. The **−s** flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. **−l** gives the long output, which is the default.

If a *user* name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), finger(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, *w* prints "–".)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wall – write to all users

SYNOPSIS

wall

DESCRIPTION

wall reads its standard input until an end-of-file. It then sends this message, preceded by 'Broadcast Message ...', to all logged in users.

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?
/etc/utmp

SEE ALSO

mesg(1), write(1)

DIAGNOSTICS

'Cannot send to ...' when the open on a user's tty file fails.

NAME

whatis – describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

whatis looks up a given command and gives the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'whatis ed' and then you should do 'man 1 ed' to get the manual.

whatis is actually just the **−f** option to the *man*(1) command.

FILES

/usr/man/whatis	Data base

SEE ALSO

man(1), catman(8)

NAME

whereis – locate source, binary, and or manual for program

SYNOPSIS

whereis [**−sbm**] [**−u**] [**−SBM** dir ... **−f**] name ...

DESCRIPTION

whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. *whereis* then attempts to locate the desired program in a list of standard places. If any of the **−b, −s** or **−m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **−u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

Finally, the **−B −M** and **−S** flags may be used to change or otherwise limit the places where *whereis* searches. The **−f** file flags is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis −u −M /usr/man/man1 −S /usr/src/cmd −f *
```

FILES

/usr/src/*
/usr/{doc,man}/*
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the **−M −S** and **−B** must be full; i.e. they must begin with a "/".

NAME

window – window environment

SYNOPSIS

window [**−t**] [**−f**] [**−d**] [**−e escape-char**] [**−c command**] [**−D**] [**−x**]

DESCRIPTION

window implements a window environment on ASCII terminals.

A window is a rectangular portion of the physical terminal screen associated with a set of processes. Its size and position can be changed by the user at any time. Processes communicate with their window in the same way they normally interact with a terminal–through their standard input, output, and diagnostic file descriptors. The window program handles the details of redirecting input an output to and from the windows. At any one time, only one window can receive input from the keyboard, but all windows can simultaneously send output to the display.

windows can overlap and are framed as necessary. Each window is named by one of the digits "1" to "9". This one character identifier, as well as a user definable label string, are displayed with the window on the top edge of its frame. A window can be designated to be in the *foreground,* in which case it will always be on top of all normal, non-foreground windows, and can be covered only by other foreground windows. A window need not be completely within the edges of the terminal screen. Thus a large window (possibly larger than the screen) may be positioned to show only a portion of its full size.

Each window has a cursor and a set of control functions. Most intelligent terminal operations such as line and character deletion and insertion are supported. Display modes such as underlining and reverse video are available if they are supported by the terminal. In addition, similar to terminals with multiple pages of memory, each window has a text buffer which can have more lines than the window itself.

OPTIONS

When *window* starts up, the commands (see long commands below) contained in the file *.windowrc* in the user's home directory are executed. If it does not exist, two equal sized windows spanning the terminal screen are created by default.

The command line options are

−t Turn on terse mode (see *terse* command below).

−f Fast. Don't perform any startup action.

−d Ignore *.windowrc* and create the two default windows instead.

−e escape-char
Set the escape character to *escape-char*. *Escape-char* can be a single character, or in the form ^*X* where *X* is any character, meaning control-*X*.

−c command
Execute the string *command* as a long command (see below) before doing anything else.

−D Toggle debug mode. An odd number of -D options sets debug mode; an even number turns it off.

−x Process start and stop characters (usually ^S and ^Q) locally, instead of passing them to the window for processing. This is especially useful on terminals that use ^S/^Q for handshaking.

PROCESS ENVIRONMENT

With each newly created window, a shell program is spawned with its process environment tailored to that window. Its standard input, output, and diagnostic file descriptors are bound to one end of either a pseudo-terminal (*pty* (4)) or a UNIX domain socket (*socketpair* (4)). If a pseudo-terminal is used, then its special characters and modes (see *stty* (1)) are copied from the physical terminal. A *termcap* (5) entry tailored to this window is created and passed as environment (*environ* (5)) variable *TERMCAP.* The termcap entry contains the window's size and characteristics as well as information from the physical terminal, such as the existence of underline, reverse video, and other display modes, and the codes produced by the terminal's function keys, if any. The environment variable *WINDOW_ID* is set to the current window number for use in prompts or other places. In addition, the window size attributes of the pseudo-terminal are set to reflect the size of this window, and updated whenever it is changed by the user. In particular, the editor *vi* (1) uses this information to redraw its display.

OPERATION

During normal execution, *window* can be in one of two states: conversation mode and command mode. In conversation mode, the terminal's real cursor is placed at the cursor position of a particular window--called the current window--and input from the keyboard is sent to the process in that window. The current window is always on top of all other windows, except those in foreground. In addition, it is set apart by highlighting its identifier and label in reverse video.

Typing *window*'s escape character (normally ^P) in conversation mode switches it into command mode. In command mode, the top line of the terminal screen becomes the command prompt window, and *window* interprets input from the keyboard as commands to manipulate windows.

There are two types of commands: short commands are usually one or two key strokes; long commands are strings either typed by the user in the command window (see the ":" command below), or read from a file (see *source* below).

SHORT COMMANDS

Below, # represents one of the digits "1" to "9" corresponding to the windows 1 to 9. ^*X* means control-*X*, where *X* is any character. In particular, ^^ is control-^. *Escape* is the escape key, or ^*[*.

#	Select window # as the current window and return to conversation mode.
%#	Select window # but stay in command mode.
^^	Select the previous window and return to conversation mode. This is useful for toggling between two windows.
escape	Return to conversation mode.
^P	Return to conversation mode and write ^P to the current window. Thus, typing two ^P's in conversation mode sends one to the current window. If the *window* escape is changed to some other character, that character takes the place of ^P here.
?	List a short summary of commands.
^L	Redraw the screen.
q	Exit *window*. Confirmation is requested.
^Z	Suspend *window*.
w	Create a new window. The user is prompted for the positions of the upper left and lower right corners of the window. The cursor is placed on the screen and the keys "h", "j", "k", and "l" move the cursor left, down, up, and right, respectively. The keys "H", "J", "K", and "L" move the cursor to the respective limits of the screen. Typing a number before the movement keys repeats the movement that number of times. Return enters the cursor position as the upper left corner of the window. The lower right corner is entered in the same manner. During this process, the placement

of the new window is indicated by a rectangular box drawn on the screen, corresponding to where the new window will be framed. Typing escape at any point cancels this command.

This window becomes the current window, and is given the first available ID. The default buffer size is used (see *nline* command below).

Only fully visible windows can be created this way.

c# Close window #. The process in the window is sent the hangup signal (see *kill* (1)). *Csh* (1) should handle this signal correctly and cause no problems.

m# Move window # to another location. A box in the shape of the window is drawn on the screen to indicate the new position of the window, and the same keys as those for the *w* command are used to position the box. The window can be moved partially off-screen.

M# Move window # to its previous position.

s# Change the size of window #. The user is prompted to enter the new lower right corner of the window. A box is drawn to indicate the new window size. The same keys used in *w* and *m* are used to enter the position.

S# Change window # to its previous size.

^Y Scroll the current window up by one line.

^E Scroll the current window down by one line.

^U Scroll the current window up by half the window size.

^D Scroll the current window down by half the window size.

^B Scroll the current window up by the full window size.

^F Scroll the current window down by the full window size.

h Move the cursor of the current window left by one column.

j Move the cursor of the current window down by one line.

k Move the cursor of the current window up by one line.

l Move the cursor of the current window right by one column.

^S Stop output in the current window.

^Q Start output in the current window.

: Enter a line to be executed as long commands. Normal line editing characters (erase character, erase word, erase line) are supported.

LONG COMMANDS

Long commands are a sequence of statements parsed much like a programming language, with a syntax similar to that of C. Numeric and string expressions and variables are supported, as well as conditional statements.

There are two data types: string and number. A string is a sequence of letters or digits beginning with a letter. "_" and "." are considered letters. Alternately, non-alphanumeric characters can be included in strings by quoting them in """ or escaping them with "\". In addition, the "\" sequences of C are supported, both inside and outside quotes (e.g., "\n" is a new line, "\r" a carriage return). For example, these are legal strings: abcde01234, "&#$^*&#", ab"$#"cd, ab\$\#cd, "/usr/ucb/window".

A number is an integer value in one of three forms: a decimal number, an octal number preceded by "0", or a hexadecimal number preceded by "0x" or "0X". The natural machine integer size is used (i.e., the signed integer type of the C compiler). As in C, a non-zero number represents a boolean true.

The character "#" begins a comment which terminates at the end of the line.

A statement is either a conditional or an expression. Expression statements are terminated with a new line or ";". To continue an expression on the next line, terminate the first line with "\".

CONDITIONAL STATEMENT

window has a single control structure: the fully bracketed if statement in the form

```
if <expr> then
        <statement>
        . . .
elsif <expr> then
        <statement>
        . . .
else
        <statement>
        . . .
endif
```

The *else* and *elsif* parts are optional, and the latter can be repeated any number of times. *<Expr>* must be numeric.

EXPRESSIONS

Expressions in *window* are similar to those in the C language, with most C operators supported on numeric operands. In addition, some are overloaded to operate on strings.

When an expression is used as a statement, its value is discarded after evaluation. Therefore, only expressions with side effects (assignments and function calls) are useful as statements.

Single valued (no arrays) variables are supported, of both numeric and string values. Some variables are predefined. They are listed below.

The operators in order of increasing precedence:

<expr1> = <expr2>
Assignment. The variable of name *<expr1>*, which must be string valued, is assigned the result of *<expr2>*. Returns the value of *<expr2>*.

<expr1> ? <expr2> : <expr3>
Returns the value of *<expr2>* if *<expr1>* evaluates true (non-zero numeric value); returns the value of *<expr3>* otherwise. Only one of *<expr2>* and *<expr3>* is evaluated. *<Expr1>* must be numeric.

<expr1> || <expr2>
Logical or. Numeric values only. Short circuit evaluation is supported (i.e., if *<expr1>* evaluates true, then *<expr2>* is not evaluated).

<expr1> && <expr2>
Logical and with short circuit evaluation. Numeric values only.

<expr1> | <expr2>
Bitwise or. Numeric values only.

<expr1> ^ <expr2>
Bitwise exclusive or. Numeric values only.

<expr1> & <expr2>
Bitwise and. Numeric values only.

<expr1> == <expr2>, <expr1> != <expr2>
Comparison (equal and not equal, respectively). The boolean result (either 1 or 0) of

the comparison is returned. The operands can be numeric or string valued. One string operand forces the other to be converted to a string in necessary.

<expr1> < <expr2>, <expr1> > <expr2>,
Less than, greater than, less than or equal to, greater than or equal to. Both numeric and string values, with automatic conversion as above.

<expr1> << <expr2>, <expr1> >> <expr2>
If both operands are numbers, *<expr1>* is bit shifted left (or right) by *<expr2>* bits. If *<expr1>* is a string, then its first (or last) *<expr2>* characters are returns (if *<expr2>* is also a string, then its length is used in place of its value).

<expr1> + <expr2>, <expr1> - <expr2>
Addition and subtraction on numbers. For "+", if one argument is a string, then the other is converted to a string, and the result is the concatenation of the two strings.

<expr1> * <expr2>, <expr1> / <expr2>,
Multiplication, division, modulo. Numbers only.

-<expr>, ~<expr>, !<expr>, $<expr>, $?<expr>
The first three are unary minus, bitwise complement and logical complement on numbers only. The operator, "$", takes *<expr>* and returns the value of the variable of that name. If *<expr>* is numeric with value *n* and it appears within an alias macro (see below), then it refers to the nth argument of the alias invocation. "$?" tests for the existence of the variable *<expr>*, and returns 1 if it exists or 0 otherwise.

<expr>(<arglist>)
Function call. *<Expr>* must be a string that is the unique prefix of the name of a builtin *window* function or the full name of a user defined alias macro. In the case of a builtin function, *<arglist>* can be in one of two forms:
　　<expr1>, <expr2>, . . .
　　argname1 = <expr1>, argname2 = <expr2>, . . .
The two forms can in fact be intermixed, but the result is unpredictable. Most arguments can be omitted; default values will be supplied for them. The *argnames* can be unique prefixes of the the argument names. The commas separating arguments are used only to disambiguate, and can usually be omitted.

Only the first argument form is valid for user defined aliases. Aliases are defined using the *alias* builtin function (see below). Arguments are accessed via a variant of the variable mechanism (see "$" operator above).

Most functions return value, but some are used for side effect only and so must be used as statements. When a function or an alias is used as a statement, the parenthesis surrounding the argument list may be omitted. Aliases return no value.

BUILTIN FUNCTIONS

The arguments are listed by name in their natural order. Optional arguments are in square brackets ("[ ]"). Arguments that have no names are in angle brackets ("<>").

alias([<string>], [<string-list>])
If no argument is given, all currently defined alias macros are listed. Otherwise, *<string>* is defined as an alias, with expansion *<string-list>*. The previous definition of *<string>*, if any, is returned. Default for *<string-list>* is no change.

close(<window-list>)
Close the windows specified in *<window-list>*. If *<window-list>* is the word *all*, than all windows are closed. No value is returned.

cursormodes([modes])
Set the window cursor to *modes*. *Modes* is the bitwise or of the mode bits defined as

the variables *m_ul* (underline), *m_rev* (reverse video), *m_blk* (blinking), and *m_grp* (graphics, terminal dependent). Return value is the previous modes. Default is no change. For example, cursor($m_rev|$m_blk) sets the window cursors to blinking reverse video.

echo([window], [<string-list>])
Write the list of strings, *<string-list>*, to *window*, separated by spaces and terminated with a new line. The strings are only displayed in the window, the processes in the window are not involved (see *write* below). No value is returned. Default is the current window.

escape([escapec])
Set the escape character to *escape-char*. Returns the old escape character as a one character string. Default is no change. *Escapec* can be a string of a single character, or in the form ^*X*, meaning control-*X*.

foreground([window], [flag])
Move *window* in or out of foreground. *Flag* can be one of *on*, *off*, *yes*, *no*, *true*, or *false*, with obvious meanings, or it can be a numeric expression, in which case a non-zero value is true. Returns the old foreground flag as a number. Default for *window* is the current window, default for *flag* is no change.

label([window], [label])
Set the label of *window* to *label*. Returns the old label as a string. Default for *window* is the current window, default for *label* is no change. To turn off a label, set it to an empty string ("").

list() No arguments. List the identifiers and labels of all windows. No value is returned.

nline([nline])
Set the default buffer size to *nline*. Initially, it is 48 lines. Returns the old default buffer size. Default is no change. Using a very large buffer can slow the program down considerably.

select([window])
Make *window* the current window. The previous current window is returned. Default is no change.

shell([<string-list>])
Set the default window shell program to *<string-list>*. Returns the first string in the old shell setting. Default is no change. Initially, the default shell is taken from the environment variable *SHELL*.

source(filename)
Read and execute the long commands in *filename*. Returns -1 if the file cannot be read, 0 otherwise.

terse([flag])
Set terse mode to *flag*. In terse mode, the command window stays hidden even in command mode, and errors are reported by sounding the terminal's bell. *Flag* can take on the same values as in *foreground* above. Returns the old terse flag. Default is no change.

unalias(alias)
Undefine *alias*. Returns -1 if *alias* does not exist, 0 otherwise.

unset(variable)
Undefine *variable*. Returns -1 if *variable* does not exist, 0 otherwise.

variables()
No arguments. List all variables. No value is returned.

window([row], [column], [nrow], [ncol], [nline], [frame], [pty], [mapnl], [shell])
Open a window with upper left corner at *row, column* and size *nrow, ncol*. If *nline* is specified, then that many lines are allocated for the text buffer. Otherwise, the default buffer size is used. Default values for *row, column, nrow,* and *ncol* are, respectively, the upper, left-most, lower, or right-most extremes of the screen. *Frame, pty,* and *mapnl* are flag values interpreted in the same way as the argument to *foreground* (see above); they mean, respectively, put a frame around this window (default true), allocate pseudo-terminal for this window rather than socketpair (default true), and map new line characters in this window to carriage return and line feed (default true if socketpair is used, false otherwise). *Shell* is a list of strings that will be used as the shell program to place in the window (default is the program specified by *shell,* see below). The created window's identifier is returned as a number.

write([window], [<string-list>])
Send the list of strings, *<string-list>*, to *window,* separated by spaces but not terminated with a new line. The strings are actually given to the window as input. No value is returned. Default is the current window.

PREDEFINED VARIABLES

These variables are for information only. Redefining them does not affect the internal operation of *window.*

baud The baud rate as a number between 50 and 38400.

modes The display modes (reverse video, underline, blinking, graphics) supported by the physical terminal. The value of *modes* is the bitwise or of some of the one bit values, *m_blk, m_grp, m_rev,* and *m_ul* (see below). These values are useful in setting the window cursors' modes (see *cursormodes* above).

m_blk The blinking mode bit.

m_grp The graphics mode bit (not very useful).

m_rev The reverse video mode bit.

m_ul The underline mode bit.

ncol The number of columns on the physical screen.

nrow The number of rows on the physical screen.

term The terminal type. The standard name, found in the second name field of the terminal's *TERMCAP* entry, is used.

FILES

~/.windowrc	startup command file.
/dev/[pt]ty[pq]?	pseudo-terminal devices.

DIAGNOSTICS

Should be self explanatory.

BUGS

When "insert line" is done in a window and line insert is supported by the physical terminal, line insertion will be used instead of screen redraw. In some cases, this can cause the screen to contain gaps, which are fixed when text is written into the gaps.

NAME

write – write to another user

SYNOPSIS

write user [ttyname]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message

Message from yourname@yoursystem on yourttyname at time...

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes 'EOT' on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *ttyname* argument may be used to indicate the appropriate terminal name.

Permission to write may be denied or granted by use of the *mesg* command. At the outset writing is allowed. Certain commands, in particular *nroff* and *pr*(1) disallow messages in order to prevent messy output.

If the character '!' is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first write to another user, wait for him to write back before starting to send. Each party should end each message with a distinctive signal–**(o)** for 'over' is conventional–that the other may reply. **(oo)** for 'over and out' is suggested when conversation is about to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute '!'

SEE ALSO

mesg(1), who(1), mail(1)

NAME

xsend, xget, enroll – secret mail

SYNOPSIS

xsend person
xget
enroll

DESCRIPTION

These commands implement a secure communication channel; it is like *mail*(1), but no one can read the messages except the intended recipient. The method embodies a public-key cryptosystem using knapsacks.

To receive messages, use *enroll*; it asks you for a password that you must subsequently quote in order to receive secret mail.

To receive secret mail, use *xget*. It asks for your password, then gives you the messages.

To send secret mail, use *xsend* in the same manner as the ordinary mail command. (However, it will accept only one target). A message announcing the receipt of secret mail is also sent by ordinary mail.

FILES

/usr/spool/secretmail/*.key: keys
/usr/spool/secretmail/*.[0-9]: messages

SEE ALSO

mail (1)

BUGS

It should be integrated with ordinary mail. The announcement of secret mail makes traffic analysis possible.

NAME

yes – be repetitively affirmative

SYNOPSIS

yes [*expletive*]

DESCRIPTION

yes repeatedly outputs "y", or if *expletive* is given, that is output repeatedly. Termination is by rubout.

NAME

adventure − an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-descriptive to a point, but part of the game is to discover its rules.

To terminate a game, type 'quit'; to save a game for later resumption, type 'suspend'.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [**+−x/**] [range]

DESCRIPTION

arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. After every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+−x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+−.**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game

SYNOPSIS

/usr/games/backgammon [**−f**] [**−n**] [**−b**] [**−r**] [**−w**] [**−p**[**brw**]] [**−t**[*term*]] [**−s**[*file*]] [*filename*]

DESCRIPTION

This program plays the game of backgammon in a full-screen mode. Instructions are provided by the game itself. If no filename is given (either with the **−s** option or separately), a new game is started. Otherwise, the file is taken to be a file into which a previous game has been saved.

The environment variable BACKGAMMON is read for options. All options except for **−s**, **-p**, and **−s** may be given in the variable. Unknown characters (including dashes and whitespace) are silently ignored.

OPTIONS

−f Fast mode. Do not pause to print message about being able to move. This makes the screen a little jumpy at (hopefully rare) times, but is often preferable to the three-second wait done normally.

−n Don't ask if rules or instructions are needed.

−b User will play both red and white.

−r User will play red.

−w User will play white.

−p[**brw**]
Print board after given player moves. By default, the board is printed upon request. This option is ignored in full-screen mode.

−t[*term*]
Use the given terminal type. If none is given, "su" (standard unknown terminal) is used.

−s[*file*]
Recover saved game from the named file. If no filename is given, this option is ignored.

NAME

banner − print large banner on printer

SYNOPSIS

/usr/games/banner [**−w***n*] message ...

DESCRIPTION

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

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

BUGS

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

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

AUTHOR

Mark Horton

NAME

battlestar – a tropical adventure game

SYNOPSIS

battlestar [**-r (recover a saved game)**]

DESCRIPTION

battlestar is an adventure game in the classic style. However, It's slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

> Three He made and gave them to His daughters,
> Beautiful nymphs, the goddesses of the waters.
> One to bring good luck and simple feats of wonder,
> Two to wash the lands and churn the waves asunder,
> Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

SAMPLE COMMANDS

take	—	take an object
drop	—	drop an object
wear	—	wear an object you are holding
draw	—	carry an object you are wearing
puton	—	take an object and wear it
take off	–	draw an object and drop it

throw <object> <direction>

! <shell esc>

IMPLIED OBJECTS

```
>-: take watermellon
watermellon:
Taken.
>-: eat
watermellon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
Taken.
broadsword:
Taken.
```

```
apple:
Taken.
knife:
Dropped.
broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.
```

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying "take knife" and then "drop" will drop the knife you just took.

SCORE & INVEN

The two commands "score" and "inven" will print out your current status in the game.

SAVING A GAME

The command "save" will save your game in a file called "Bstar." You can recover a saved game by using the "-r" option when you start up the game.

DIRECTIONS

The compass directions N, S, E, and W can be used if you have a compass. If you don't have a compass, you'll have to say R, L, A, or B, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

HISTORY

I wrote battlestar in 1979 in order to experiment with the niceties of the C Language. Most interesting things that happen in the game are hardwired into the code, so don't send me any hate mail about it! Instead, enjoy art for art's sake!

AUTHOR

David Riggle

INSPIRATION & ASSISTANCE

Chris Guthrie
Peter Da Silva
Kevin Brown
Edward Wang
Ken Arnold & Company

BUGS

Countless.

FAN MAIL

Send to edward%ucbarpa@Berkeley.arpa, chris%ucbcory@berkeley.arpa, riggle.pa@xerox.arpa.

NAME

bcd – convert to antique media

SYNOPSIS

/usr/games/bcd text

DESCRIPTION

bcd converts the literal *text* into a form familiar to old-timers.

SEE ALSO

dd(1)

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [**+**] [**++**]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

canfield, cfscores – the solitaire card game canfield

SYNOPSIS

/usr/games/canfield [**−a**] [**−b**] [**−c**] [**−n**] [**−x**]
/usr/games/cfscores [**−a**]

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase one's chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as possible without moving any cards from your hand to the talon. (The initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

Normally, input must followed by a newline. In abbreviated mode, input is read until the command is known. In some cases this is one character, and in other cases it is two characters. It is a good idea to use the standard input mode until you understand all of the commands, since playing too fast can lose the game.

The environment variable CANFIELD is read before processing the command-line options. The processing of this data is naive. If any of the known option characters are found, those options are set, and all other characters are silently ignored.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account.

OPTIONS

−a In **canfield**, the **−a** option says to turn on abbreviated mode (see above). In **cfscores, this option says to print scores for all players.**

−b Start with the betting box. The default is to start with the instruction box.

−c Turn on card counting.

−n No asking if instructions are desired.

−x Start with no box.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

AUTHORS

Originally written: Steve Levine
Further random hacking by: Steve Feldman, Kirk McKusick, Mikey Olson, and Eric Allman.

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**−req**] *name* ...

DESCRIPTION

cribbage plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed – if so, it will print out the appropriate section from *According to Hoyle* with *more (I).*

cribbage options include:

−e When the player makes a mistake scoring his hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

−q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

−r Instead of asking the player to cut the deck, the program will randomly cut the deck.

cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' will result in a short game, any other response will play a long game.

At the start of the first game, the program asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. The ranks may be specified as one of: 'a', '2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', and 'k', or alternatively, one of: "ace", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "jack", "queen", and "king". Suits may be specified as: 's', 'h', 'd', and 'c', or alternatively as: "spades", "hearts", "diamonds", and "clubs". A card may be specified as: <rank> " " <suit>, or: <rank> " of " <suit>. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand was "2H, 4D, 5C, 6H, JC, KD" and it was desired to discard the king of diamonds, any of the following could be typed: "k", "king", "kd", "k d", "k of d", "king d", "king of d", "k diamonds", "k of diamonds", "king diamonds", or "king of diamonds".

FILES

/usr/games/cribbage

AUTHORS

Earl T. Cohen wrote the logic. Ken Arnold added the screen oriented interface.

NAME

fish – play "Go fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of "Go fish", a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [**–**] [**–wslaod**] [*file*]

DESCRIPTION

fortune with no arguments prints out a random adage. The flags mean:

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

–o Choose from an alternate list of adages, often used for potentially offensive ones.

–a Choose from either list of adages.

–d Dump the table information. This gives information about which file was read, the sizes of the adages, the internal state, and the file that was written (see below).

In order to cause *fortune* to be random, the program must store a table in a file. By default, this information is stored in the data file, but this can cause the data file to be backed up all of the time. In order to defeat this, if the file */usr/games/lib/fortunes.tbl* exists and is owned by daemon, it is used as the table file. If the file is empty, the table is read from the standard file, but the table will be written to the alternate file. The result is that there is a 28-byte file that gets backed up instead of a 400Kbyte file.

The user may specify a file of adages. This file must be created by the program **strfile**, which is only available with the *fortune* source, and be given by the user as *file*. Only one such file may be named, subsequent ones are ignored. In addition, this defeats the use of the alternate table file.

FILES

/usr/games/lib/fortunes.dat
Data file

/usr/games/lib/fortunes.tbl
Alternate table file

AUTHOR

Ken Arnold

NAME

hangman – Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In *hangman,* the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	On-line word list

AUTHOR

Ken Arnold

NAME

hunt – a multi-player multi-terminal game

SYNOPSIS

/usr/games/hunt [-q] [-**m**] [hostname] [-**l** name]

DESCRIPTION

The object of the game *hunt* is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players. The more players you kill before you die, the better your score is. If the **−m** flag is given, you enter the game as a monitor (you can see the action but you cannot play).

hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname* argument. The player name may be specified on the command line by using the **-l** option. This command syntax was chosen for *rlogin/rsh* compatibility. If the **−q** flag is given, *hunt* queries the network and reports if an active game were found. This is useful for .login scripts.

hunt only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that don't fit in the status area.

hunt uses the same keys to move as *vi* does, *i.e.*, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (*i.e.*, HJKL).

Other commands are:

f	– Fire (in the direction you're facing) (Takes 1 charge)
g	– Throw grenade (in the direction you're facing) (Takes 9 charges)
F	– Throw satchel charge (Takes 25 charges)
G	– Throw bomb (Takes 49 charges)
o	– Throw small slime bomb (Takes 15 charges)
O	– Throw big slime bomb (Takes 30 charges)
s	– Scan (show where other players are) (Takes 1 charge)
c	– Cloak (hide from scanners) (Takes 1 charge)
^L	– Redraw screen
q	– Quit

Knowing what the symbols on the screen often helps:

-|+	– walls
/\	– diagonal (deflecting) walls
#	– doors (dispersion walls)
;	– small mine
g	– large mine
:	– shot
o	– grenade
O	– satchel charge
@	– bomb
s	– small slime bomb

```
$        – big slime bomb
> < ^ v  – you facing right, left, up, or down
} { i !  – other players facing right, left, up, or down
*        – explosion
\ | /
– * –    – grenade and large mine explosion
/ | \
```

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

- You can only fire in the direction you are facing.
- You can only fire three shots in a row, then the gun must cool.
- A shot only affects the square it hits.
- Shots and grenades move 5 times faster than you do.
- To stab someone, you must face that player and move at them.
- Stabbing does 2 points worth of damage and shooting does 5 points.
- Slime does 5 points of damage each time it hits.
- You start with 15 charges and get 5 more for every new player.
- A grenade affects the nine squares centered about the square it hits.
- A satchel affects the twenty-five squares centered about the square it hits.
- A bomb affects the forty-nine squares centered about the square it hits.
- Slime affects all squares it oozes over (15 or 30 respectively).
- One small mine and one large mine is placed in the maze for every new player. A mine has a 5% probability of tripping when you walk directly at it; 50% when going sideways on to it; 95% when backing up on to it. Tripping a mine costs you 5 points or 10 points respectively. Defusing a mine is worth 1 charge or 9 charges respectively.
- You cannot see behind you.
- Scanning lasts for (20 times the number of players) turns. Scanning takes 1 ammo charge, so don't waste all your charges scanning.
- Cloaking lasts for 20 turns.
- Whenever you kill someone, you get 2 more damage capacity points and 2 damage points taken away.
- Maximum typeahead is 5 characters.
- A shot destroys normal (*i.e.,* non-diagonal, non-door) walls.
- Diagonal walls deflect shots and change orientation.
- Doors disperse shots in random directions (up, down, left, right).
- Diagonal walls and doors cannot be destroyed by direct shots but may be destroyed by an adjacent grenade explosion.
- Slime goes around walls, not through them.
- Walls regenerate, reappearing in the order they were destroyed. One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.

- The environment variable **HUNT** is checked to get the player name. If you don't have this variable set, *hunt* will ask you what name you want to play under. If it is set, you may also set up a single character keyboard map, but then you have to enumerate the options:

e.g. setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"

sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**. The *mapkey* option must be last.

- It's a boring game if you're the only one playing.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of *hunt*.

hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a *hunt* game executing. A limit of three players per host and nine players total is enforced by *hunt*.

FILES

/usr/games/lib/hunt.driver game coordinator

AUTHORS

Conrad Huang, Ken Arnold, and Greg Couch; University of California, San Francisco, Computer Graphics Lab

ACKNOWLEDGEMENTS

We thank Don Kneller, John Thomason, Eric Pettersen, and Scott Weiner for providing endless hours of play-testing to improve the character of the game. We hope their significant others will forgive them; we certainly don't.

BUGS

To keep up the pace, not everything is as realistic as possible.

There were some bugs in early releases of 4.2 BSD that *hunt* helped discover; *hunt* will crash your system if those bugs haven't been fixed.

NAME

mille – play mille Bournes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

mille plays a two-handed game reminiscent of the Parker Brother's game of mille Bournes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P	Pick a card from the deck. This card is placed in the 'P' slot in your hand.
D	Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a <RETURN> or <SPACE>. The <RETURN or <SPACE> is required to allow recovery from typos which can be very expensive, like discarding safeties.
U	Use a card. The card is again indicated by its number, followed by a <RETURN> or <SPACE>.
O	Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
Q	Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.
S	Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a <RETURN> without a name, the save will be terminated and the game resumed.
R	Redraw the screen from scratch. The command ^L (control 'L') will also work.
W	Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save.

AUTHOR

Ken Arnold
(The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3X), *Screen Updating and Cursor Movement Optimization: A Library Package*, Ken Arnold

CARDS

Here is some useful information. The number in parentheses after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

RULES

Object: The point of this game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. They can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card other than a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand*. A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

Coup Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.
Each Safety: 100 points for each safety in the Safety area.
All 4 Safeties: 300 points if all four safeties are played.
Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.
Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.
Delayed Action: 300 points bonus for finishing after the deck was exhausted.
Extension: 200 points bonus for completing a 1000 mile trip.
Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

monop is reminiscent of the Parker Brother's game monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of monopoly are known. The game follows the standard rules, with the exception that, if a property goes up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings:
List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type "done".

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage: mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. It operates in an analogous manner to *buy*.

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/cards.pck	Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [**−i** file] [**−t**] [category1 category2]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **−t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **−i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category  = alternate | category '|' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with *sh*(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

NAME

robots – fight off villainous robots

SYNOPSIS

/usr/games/robots [**−sjta**] [**scorefile**]

DESCRIPTION

robots pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

robots are represented on the screen by a '**+**', the junk heaps from their collisions by a '*', and you (the good guy) by a '**@**'.

The commands are:

h	move one square left
l	move one square right
k	move one square up
j	move one square down
y	move one square up and left
u	move one square up and right
b	move one square down and left
n	move one square down and right
.	(also space) do nothing for one turn
HJKLBNYU	run as far as possible in the given direction
>	do nothing for as long as possible
t	teleport to a random location
w	wait until you die or they all do
q	quit
^L	redraw the screen

All commands can be preceded by a count.

If you use the '**w**' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with '**w**' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

The options are

−s	Don't play, just show the score file
−j	Jump, *i.e.*, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.
−t	Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.
−a	Advance into the higher levels directly, skipping the lower, easier levels.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/robots_roll	the score file

BUGS

Bugs? You *crazy*, man?!?

NAME

sail – multi-user wooden ships and iron men

SYNOPSIS

sail [**−s** [**−l**]] [**−x**] [**−b**] [**num**]

DESCRIPTION

sail is a computer version of Avalon Hill's game of fighting sail originally developed by S. Craig Taylor.

Players of *sail* take command of an old fashioned Man of War and fight other players or the computer. They may re-enact one of the many historical sea battles recorded in the game, or they can choose a fictional battle.

As a sea captain in the *sail* Navy, the player has complete control over the workings of his ship. He must order every maneuver, change the set of his sails, and judge the right moment to let loose the terrible destruction of his broadsides. In addition to fighting the enemy, he must harness the powers of the wind and sea to make them work for him. The outcome of many battles during the age of sail was decided by the ability of one captain to hold the 'weather gage.'

The flags are:

−s	Print the names and ships of the top ten sailors.
−l	Show the login name. Only effective with **-s**.
−x	Play the first available ship instead of prompting for a choice.
−b	No bells.

IMPLEMENTATION

sail is really two programs in one. Each player starts up a process which runs his own ship. In addition, a *driver* process is forked (by the first player) to run the computer ships and take care of global bookkeeping.

Because the *driver* must calculate moves for each ship it controls, the more ships the computer is playing, the slower the game will appear.

If a player joins a game in progress, he will synchronize with the other players (a rather slow process for everyone), and then he may play along with the rest.

To implement a multi-user game in Version 7 UNIX, which was the operating system *sail* was first written under, the communicating processes must use a common temporary file as a place to read and write messages. In addition, a locking mechanism must be provided to ensure exclusive access to the shared file. For example, *sail* uses a temporary file named /tmp/#sailsink.21 for scenario 21, and corresponding file names for the other scenarios. To provide exclusive access to the temporary file, *sail* uses a technique stolen from an old game called "pubcaves" by Jeff Cohen. Processes do a busy wait in the loop

```
for (n = 0; link(sync_file, sync_lock) < 0 && n < 30; n++)
        sleep(2);
```

until they are able to create a link to a file named "/tmp/#saillock.??". The "??" correspond to the scenario number of the game. Since UNIX guarantees that a link will point to only one file, the process that succeeds in linking will have exclusive access to the temporary file.

Whether or not this really works is open to speculation. When ucbmiro was rebooted after a crash, the file system check program found 3 links between the *sail* temporary file and its link file.

CONSEQUENCES OF SEPARATE PLAYER AND DRIVER

When players do something of global interest, such as moving or firing, the driver must coordinate the action with the other ships in the game. For example, if a player wants to move in a certain direction, he writes a message into the temporary file requesting the driver to move his ship. Each "turn," the driver reads all the messages sent from the players and decides what happened. It then writes back into the temporary file new values of variables, etc.

The most noticeable effect this communication has on the game is the delay in moving. Suppose a player types a move for his ship and hits return. What happens then? The player process saves up messages to be written to the temporary file in a buffer. Every 7 seconds or so, the player process gets exclusive access to the temporary file and writes out its buffer to the file. The driver, running asynchronously, must read in the movement command, process it, and write out the results. This takes two exclusive accesses to the temporary file. Finally, when the player process gets around to doing another 7 second update, the results of the move are displayed on the screen. Hence, every movement requires four exclusive accesses to the temporary file (anywhere from 7 to 21 seconds depending upon asynchrony) before the player sees the results of his moves.

In practice, the delays are not as annoying as they would appear. There is room for "pipelining" in the movement. After the player writes out a first movement message, a second movement command can then be issued. The first message will be in the temporary file waiting for the driver, and the second will be in the file buffer waiting to be written to the file. Thus, by always typing moves a turn ahead of the time, the player can sail around quite quickly.

If the player types several movement commands between two 7 second updates, only the last movement command typed will be seen by the driver. Movement commands within the same update "overwrite" each other, in a sense.

THE HISTORY OF SAIL

I wrote the first version of *sail* on a PDP 11/70 in the fall of 1980. Needless to say, the code was horrendous, not portable in any sense of the word, and didn't work. The program was not very modular and had fseeks() and fwrites() every few lines. After a tremendous rewrite from the top down, I got the first working version up by 1981. There were several annoying bugs concerning firing broadsides and finding angles. *sail* uses no floating point, by the way, so the direction routines are rather tricky. Ed Wang rewrote my angle() routine in 1981 to be more correct (although it still doesn't work perfectly), and he added code to let a player select which ship he wanted at the start of the game (instead of the first one available).

Captain Happy (Craig Leres) is responsible for making *sail* portable for the first time. This was no easy task, by the way. Constants like 2 and 10 were very frequent in the code. I also became famous for using "Riggle Memorial Structures" in *sail*. Many of my structure references are so long that they run off the line printer page. Here is an example, if you promise not to laugh.

```
specs[scene[flog.fgamenum].ship[flog.fshipnum].shipnum].pts
```

sail received its fourth and most thorough rewrite in the summer and fall of 1983. Ed Wang rewrote and modularized the code (a monumental feat) almost from scratch. Although he introduced many new bugs, the final result was very much cleaner and (?) faster. He added window movement commands and find ship commands.

HISTORICAL INFO

Old Square Riggers were very maneuverable ships capable of intricate sailing. Their only disadvantage was an inability to sail very close to the wind. The design of a wooden ship allowed only for the guns to bear to the left and right sides. A few guns of small aspect (usually 6 or 9 pounders) could point forward, but their effect was small compared to a 68 gun broadside of 24 or 32 pounders. The guns bear approximately like so:

```
 \
  b----------
--0
  \
   \
    \   up to a range of ten (for round shot)
     \
      \
       \
```

An interesting phenomenon occurred when a broadside was fired down the length of an enemy ship. The shot tended to bounce along the deck and did several times more damage. This phenomenon was called a rake. Because the bows of a ship are very strong and present a smaller target than the stern, a stern rake (firing from the stern to the bow) causes more damage than a bow rake.

```
        b
        00 --- Stern rake!
         a
```

Most ships were equipped with carronades, which were very large, close range cannons. American ships from the revolution until the War of 1812 were almost entirely armed with carronades.

The period of history covered in *sail* is approximately from the 1770's until the end of Napoleanic France in 1815. There are many excellent books about the age of sail. My favorite author is Captain Frederick Marryat. More contemporary authors include C.S. Forester and Alexander Kent.

Fighting ships came in several sizes classed by armament. The mainstays of any fleet were its "Ships of the Line", or "Line of Battle Ships". They were so named because these ships fought together in great lines. They were close enough for mutual support, yet every ship could fire both its broadsides. We get the modern words "ocean liner," or "liner," and "battleship" from "ship of the line." The most common size was the the 74 gun two decked ship of the line. The two gun decks usually mounted 18 and 24 pounder guns.

The pride of the fleet were the first rates. These were huge three decked ships of the line mounting 80 to 136 guns. The guns in the three tiers were usually 18, 24, and 32 pounders in that order from top to bottom.

Various other ships came next. They were almost all "razees," or ships of the line with one deck sawed off. They mounted 40-64 guns and were a poor cross between a frigate and a line of battle ship. They neither had the speed of the former nor the firepower of the latter.

Next came the "eyes of the fleet." Frigates came in many sizes mounting anywhere from 32 to 44 guns. They were very handy vessels. They could outsail anything bigger and outshoot anything smaller. Frigates didn't fight in lines of battle as the much bigger 74's did. Instead, they harassed the enemy's rear or captured crippled ships. They were much more useful in missions away from the fleet, such as cutting out expeditions or boat actions. They could hit hard and get away fast.

Lastly, there were the corvettes, sloops, and brigs. These were smaller ships mounting typically fewer than 20 guns. A corvette was only slightly smaller than a frigate, so one might have up to 30 guns. Sloops were used for carrying dispatches or passengers. Brigs were something you built for land-locked lakes.

SAIL PARTICULARS

Ships in *sail* are represented by two characters. One character represents the bow of the ship, and the other represents the stern. Ships have nationalities and numbers. The first ship of a nationality is number 0, the second number 1, etc. Therefore, the first British ship in a game would be printed as "b0". The second Brit would be "b1", and the fifth Don would be "s4".

Ships can set normal sails, called Battle sails, or bend on extra canvas called Full sails. A ship under full sail is a beautiful sight indeed, and it can move much faster than a ship under Battle sails. The only trouble is, with full sails set, there is so much tension on sail and rigging that a well aimed round shot can burst a sail into ribbons where it would only cause a little hole in a loose sail. For this reason, rigging damage is doubled on a ship with full sails set. Don't let that discourage you from using full sails. I like to keep them up right into the heat of battle. A ship with full sails set has a capital letter for its nationality. E.g., a Frog, "f0", with full sails set would be printed as "F0".

When a ship is battered into a listing hulk, the last man aboard "strikes the colors." This ceremony is the ship's formal surrender. The nationality character of a surrendered ship is printed as "!". E.g., the Frog of our last example would soon be "!0".

A ship has a random chance of catching fire or sinking when it reaches the stage of listing hulk. A sinking ship has a "~" printed for its nationality, and a ship on fire and about to explode has a "#" printed.

Captured ships become the nationality of the prize crew. Therefore, if an American ship captures a British ship, the British ship will have an "a" printed for its nationality. In addition, the ship number is changed to "&","'", "(", ,")", "*", or "+" depending upon the original number, be it 0,1,2,3,4, or 5. E.g., the "b0" captured by an American becomes the "a&". The "s4" captured by a Frog becomes the "f*".

The ultimate example is, of course, an exploding Brit captured by an American: "#&".

MOVEMENT

Movement is the most confusing part of *sail* to many. Ships can head in 8 directions:

```
                         0      0      0
  b      b      b0      b      b      b      0b     b
  0      0                                    0
```

The stern of a ship moves when it turns. The bow remains stationary. Ships can always turn, regardless of the wind (unless they are becalmed). All ships drift when they lose headway. If a ship doesn't move forward at all for two turns, it will begin to drift. If a ship has begun to drift, then it must move forward before it turns, if it plans to do more than make a right or left turn, which is always possible.

Movement commands to *sail* are a string of forward moves and turns. An example is "l3". It will turn a ship left and then move it ahead 3 spaces. In the drawing above, the "b0" made 7 successive left turns. When *sail* prompts you for a move, it prints three characters of import. E.g.,

    move (7, 4):

The first number is the maximum number of moves you can make, including turns. The second number is the maximum number of turns you can make. Between the numbers is sometimes printed a quote "'". If the quote is present, it means that your ship has been drifting, and you must move ahead to regain headway before you turn (see note above). Some of the possible moves for the example above are as follows:

```
move (7, 4): 7
move (7, 4): 1
move (7, 4): d          /* drift, or do nothing */
move (7, 4): 6r
move (7, 4): 5r1
move (7, 4): 4r1r
move (7, 4): l1r1r2
move (7, 4): 1r1r1r1
```

Because square riggers performed so poorly sailing into the wind, if at any point in a movement command you turn into the wind, the movement stops there. E.g.,

```
move (7, 4): l1l4
Movement Error;
Helm: l1l
```

Moreover, whenever you make a turn, your movement allowance drops to min(what's left, what you would have at the new attitude). In short, if you turn closer to the wind, you most likely won't be able to sail the full allowance printed in the "move" prompt.

Old sailing captains had to keep an eye constantly on the wind. Captains in *sail* are no different. A ship's ability to move depends on its attitide to the wind. The best angle possible is to have the wind off your quarter, that is, just off the stern. The direction rose on the side of the screen gives the possible movements for your ship at all positions to the wind. Battle sail speeds are given first, and full sail speeds are given in parenthesis.

```
 0 1(2)
\|/
-^-3(6)
/|\
 |4(7)
3(6)
```

Pretend the bow of your ship (the "^") is pointing upward and the wind is blowing from the bottom to the top of the page. The numbers at the bottom "3(6)" will be your speed under battle or full sails in such a situation. If the wind is off your quarter, then you can move "4(7)". If the wind is off your beam, "3(6)". If the wind is off your bow, then you can only move "1(2)". Facing into the wind, you can't move at all. Ships facing into the wind were said to be "in irons".

WINDSPEED AND DIRECTION

The windspeed and direction is displayed as a little weather vane on the side of the screen. The number in the middle of the vane indicates the wind speed, and the + to - indicates the wind direction. The wind blows from the + sign (high pressure) to the - sign (low pressure). E.g.,

```
|
3
+
```

The wind speeds are 0 = becalmed, 1 = light breeze, 2 = moderate breeze, 3 = fresh breeze, 4 = strong breeze, 5 = gale, 6 = full gale, 7 = hurricane. If a hurricane shows up, all ships are destroyed.

GRAPPLING AND FOULING

If two ships collide, they run the risk of becoming tangled together. This is called "fouling." Fouled ships are stuck together, and neither can move. They can unfoul each other if they want to. Boarding parties can only be sent across to ships when the antagonists are either fouled or grappled.

Ships can grapple each other by throwing grapnels into the rigging of the other.

The number of fouls and grapples you have are displayed on the upper right of the screen.

BOARDING

Boarding was a very costly venture in terms of human life. Boarding parties may be formed in *sail* to either board an enemy ship or to defend your own ship against attack. Men organized as Defensive Boarding Parties fight twice as hard to save their ship as men left unorganized.

The boarding strength of a crew depends upon its quality and upon the number of men sent.

CREW QUALITY

The British seaman was world renowned for his sailing abilities. American sailors, however, were actually the best seamen in the world. Because the American Navy offered twice the wages of the Royal Navy, British seamen who liked the sea defected to America by the thousands.

In *sail,* crew quality is quantized into 5 energy levels. "Elite" crews can outshoot and outfight all other sailors. "Crack" crews are next. "Mundane" crews are average, and "Green" and "Mutinous" crews are below average. A good rule of thumb is that "Crack" or "Elite" crews get one extra hit per broadside compared to "Mundane" crews. Don't expect too much from "Green" crews.

BROADSIDES

Your two broadsides may be loaded with four kinds of shot: grape, chain, round, and double. You have guns and carronades in both the port and starboard batteries. Carronades only have a range of two, so you have to get in close to be able to fire them. You have the choice of firing at the hull or rigging of another ship. If the range of the ship is greater than 6, then you may only shoot at the rigging.

The types of shot and their advantages are:

ROUND

Range of 10. Good for hull or rigging hits.

DOUBLE

Range of 1. Extra good for hull or rigging hits. Double takes two turns to load.

CHAIN

Range of 3. Excellent for tearing down rigging. Cannot damage hull or guns, though.

GRAPE

Range of 1. Sometimes devastating against enemy crews.

On the side of the screen is displayed some vital information about your ship:

```
Load  D! R!
Hull  9
Crew  4  4  2
Guns  4  4
Carr  2  2
Rigg  5 5 5 5
```

"Load" shows what your port (left) and starboard (right) broadsides are loaded with. A "!" after the type of shot indicates that it is an initial broadside. Initial broadside were loaded with care before battle and before the decks ran red with blood. As a consequence, initial broadsides are a little more effective than broadsides loaded later. A "*" after the type of shot indicates that the gun crews are still loading it, and you cannot fire yet. "Hull" shows how much hull you have left. "Crew" shows your three sections of crew. As your crew dies off, your ability to fire decreases. "Guns" and "Carr" show your port and starboard guns. As you lose guns, your ability to fire decreases. "Rigg" shows how much rigging you have on your 3 or 4 masts. As rigging is shot away, you lose mobility.

EFFECTIVENESS OF FIRE

It is very dramatic when a ship fires its thunderous broadsides, but the mere opportunity to fire them does not guarantee any hits. Many factors influence the destructive force of a broadside. First of all, and the chief factor, is distance. It is harder to hit a ship at range ten than it is to hit one sloshing alongside. Next is raking. Raking fire, as mentioned before, can sometimes dismast a ship at range ten. Next, crew size and quality affects the damage done by a broadside. The number of guns firing also bears on the point, so to speak. Lastly, weather affects the accuracy of a broadside. If the seas are high (5 or 6), then the lower gunports of ships of the line can't even be opened to run out the guns. This gives frigates and other flush decked vessels an advantage in a storm. The scenario *Pellew vs. The Droits de L'Homme* takes advantage of this peculiar circumstance.

REPAIRS

Repairs may be made to your Hull, Guns, and Rigging at the slow rate of two points per three turns. The message "Repairs Completed" will be printed if no more repairs can be made.

PECULIARITIES OF COMPUTER SHIPS

Computer ships in *sail* follow all the rules above with a few exceptions. Computer ships never repair damage. If they did, the players could never beat them. They play well enough as it is. As a consolation, the computer ships can fire double shot every turn. That fluke is a good reason to keep your distance. The *Driver* figures out the moves of the computer ships. It computes them with a typical A.I. distance function and a depth first search to find the maximum "score." It seems to work fairly well, although I'll be the first to admit it isn't perfect.

HOW TO PLAY

Commands are given to *sail* by typing a single character. You will then be prompted for further input. A brief summary of the commands follows.

COMMAND SUMMARY

'f'	Fire broadsides if they bear
'l'	Reload
'L'	Unload broadsides (to change ammo)
'm'	Move
'i'	Print the closest ship
'I'	Print all ships
'F'	Find a particular ship or ships (e.g. "a?" for all Americans)
's'	Send a message around the fleet
'b'	Attempt to board an enemy ship
'B'	Recall boarding parties
'c'	Change set of sail
'r'	Repair
'u'	Attempt to unfoul
'g'	Grapple/ungrapple
'v'	Print version number of game
'^L'	Redraw screen
'Q'	Quit

'C'	Center your ship in the window
'U'	Move window up
'D','N'	Move window down
'H'	Move window left
'J'	Move window right
'S'	Toggle window to follow your ship or stay where it is

SCENARIOS

Here is a summary of the scenarios in *sail:*

Ranger vs. Drake:

Wind from the N, blowing a fresh breeze.

(a) Ranger	19 gun Sloop (crack crew) (7 pts)
(b) Drake	17 gun Sloop (crack crew) (6 pts)

The Battle of Flamborough Head:

Wind from the S, blowing a fresh breeze.

This is John Paul Jones' first famous battle. Aboard the Bonhomme Richard, he was able to overcome the Serapis's greater firepower by quickly boarding her.

(a) Bonhomme Rich	42 gun Corvette (crack crew) (11 pts)
(b) Serapis	44 gun Frigate (crack crew) (12 pts)

Arbuthnot and Des Touches:

Wind from the N, blowing a gale.

(b) America	64 gun Ship of the Line (crack crew) (20 pts)
(b) Befford	74 gun Ship of the Line (crack crew) (26 pts)
(b) Adamant	50 gun Ship of the Line (crack crew) (17 pts)
(b) London	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Royal Oak	74 gun Ship of the Line (crack crew) (26 pts)
(f) Neptune	74 gun Ship of the Line (average crew) (24 pts)

(f) Duc Bougogne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Conquerant	74 gun Ship of the Line (average crew) (24 pts)
(f) Provence	64 gun Ship of the Line (average crew) (18 pts)
(f) Romulus	44 gun Ship of the Line (average crew) (10 pts)

Suffren and Hughes:
Wind from the S, blowing a fresh breeze.

(b) Monmouth	74 gun Ship of the Line (average crew) (24 pts)
(b) Hero	74 gun Ship of the Line (crack crew) (26 pts)
(b) Isis	50 gun Ship of the Line (crack crew) (17 pts)
(b) Superb	74 gun Ship of the Line (crack crew) (27 pts)
(b) Burford	74 gun Ship of the Line (average crew) (24 pts)
(f) Flamband	50 gun Ship of the Line (average crew) (14 pts)
(f) Annibal	74 gun Ship of the Line (average crew) (24 pts)
(f) Severe	64 gun Ship of the Line (average crew) (18 pts)
(f) Brilliant	80 gun Ship of the Line (crack crew) (31 pts)
(f) Sphinx	80 gun Ship of the Line (average crew) (27 pts)

Nymphe vs. Cleopatre:
Wind from the S, blowing a fresh breeze.

(b) Nymphe	36 gun Frigate (crack crew) (11 pts)
(f) Cleopatre	36 gun Frigate (average crew) (10 pts)

Mars vs. Hercule:
Wind from the S, blowing a fresh breeze.

(b) Mars	74 gun Ship of the Line (crack crew) (26 pts)
(f) Hercule	74 gun Ship of the Line (average crew) (23 pts)

Ambuscade vs. Baionnaise:
Wind from the N, blowing a fresh breeze.

(b) Ambuscade	32 gun Frigate (average crew) (9 pts)
(f) Baionnaise	24 gun Corvette (average crew) (9 pts)

Constellation vs. Insurgent:
Wind from the S, blowing a gale.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Insurgent	36 gun Corvette (average crew) (11 pts)

Constellation vs. Vengeance:
Wind from the S, blowing a fresh breeze.

(a) Constellation	38 gun Corvette (elite crew) (17 pts)
(f) Vengeance	40 gun Frigate (average crew) (15 pts)

The Battle of Lissa:
Wind from the S, blowing a fresh breeze.

(b) Amphion	32 gun Frigate (elite crew) (13 pts)
(b) Active	38 gun Frigate (elite crew) (18 pts)
(b) Volage	22 gun Frigate (elite crew) (11 pts)
(b) Cerberus	32 gun Frigate (elite crew) (13 pts)
(f) Favorite	40 gun Frigate (average crew) (15 pts)
(f) Flore	40 gun Frigate (average crew) (15 pts)

(f) Danae	40 gun Frigate (crack crew) (17 pts)
(f) Bellona	32 gun Frigate (green crew) (9 pts)
(f) Corona	40 gun Frigate (green crew) (12 pts)
(f) Carolina	32 gun Frigate (green crew) (7 pts)

Constitution vs. Guerriere:
Wind from the SW, blowing a gale.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Guerriere	38 gun Frigate (crack crew) (15 pts)

United States vs. Macedonian:
Wind from the S, blowing a fresh breeze.

(a) United States	44 gun Frigate (elite crew) (24 pts)
(b) Macedonian	38 gun Frigate (crack crew) (16 pts)

Constitution vs. Java:
Wind from the S, blowing a fresh breeze.

(a) Constitution	44 gun Corvette (elite crew) (24 pts)
(b) Java	38 gun Corvette (crack crew) (19 pts)

Chesapeake vs. Shannon:
Wind from the S, blowing a fresh breeze.

(a) Chesapeake	38 gun Frigate (average crew) (14 pts)
(b) Shannon	38 gun Frigate (elite crew) (17 pts)

The Battle of Lake Erie:
Wind from the S, blowing a light breeze.

(a) Lawrence	20 gun Sloop (crack crew) (9 pts)
(a) Niagara	20 gun Sloop (elite crew) (12 pts)
(b) Lady Prevost	13 gun Brig (crack crew) (5 pts)
(b) Detroit	19 gun Sloop (crack crew) (7 pts)
(b) Q. Charlotte	17 gun Sloop (crack crew) (6 pts)

Wasp vs. Reindeer:
Wind from the S, blowing a light breeze.

(a) Wasp	20 gun Sloop (elite crew) (12 pts)
(b) Reindeer	18 gun Sloop (elite crew) (9 pts)

Constitution vs. Cyane and Levant:
Wind from the S, blowing a moderate breeze.

(a) Constitution 44 gun Corvette (elite crew) (24 pts) (b) Cyane 24 gun Sloop (crack crew) (11 pts) (b) Levant 20 gun Sloop (crack crew) (10 pts)

Pellew vs. Droits de L'Homme:
Wind from the N, blowing a gale.

(b) Indefatigable	44 gun Frigate (elite crew) (14 pts)
(b) Amazon	36 gun Frigate (crack crew) (14 pts)
(f) Droits L'Hom	74 gun Ship of the Line (average crew) (24 pts)

Algeciras:

Wind from the SW, blowing a moderate breeze.

(b) Caesar	80 gun Ship of the Line (crack crew) (31 pts)
(b) Pompee	74 gun Ship of the Line (crack crew) (27 pts)
(b) Spencer	74 gun Ship of the Line (crack crew) (26 pts)
(b) Hannibal	98 gun 3 Decker SOL (crack crew) (28 pts)
(s) Real-Carlos	112 gun 3 Decker SOL (green crew) (27 pts)
(s) San Fernando	96 gun 3 Decker SOL (green crew) (24 pts)
(s) Argonauta	80 gun Ship of the Line (green crew) (23 pts)
(s) San Augustine	74 gun Ship of the Line (green crew) (20 pts)
(f) Indomptable	80 gun Ship of the Line (average crew) (27 pts)
(f) Desaix	74 gun Ship of the Line (average crew) (24 pts)

Lake Champlain:

Wind from the N, blowing a fresh breeze.

(a) Saratoga	26 gun Sloop (crack crew) (12 pts)
(a) Eagle	20 gun Sloop (crack crew) (11 pts)
(a) Ticonderoga	17 gun Sloop (crack crew) (9 pts)
(a) Preble	7 gun Brig (crack crew) (4 pts)
(b) Confiance	37 gun Frigate (crack crew) (14 pts)
(b) Linnet	16 gun Sloop (elite crew) (10 pts)
(b) Chubb	11 gun Brig (crack crew) (5 pts)

Last Voyage of the USS President:

Wind from the N, blowing a fresh breeze.

(a) President	44 gun Frigate (elite crew) (24 pts)
(b) Endymion	40 gun Frigate (crack crew) (17 pts)
(b) Pomone	44 gun Frigate (crack crew) (20 pts)
(b) Tenedos	38 gun Frigate (crack crew) (15 pts)

Hornblower and the Natividad:

Wind from the E, blowing a gale.

A scenario for you Horny fans. Remember, he sank the Natividad against heavy odds and winds. Hint: don't try to board the Natividad, her crew is much bigger, albeit green.

(b) Lydia	36 gun Frigate (elite crew) (13 pts)
(s) Natividad	50 gun Ship of the Line (green crew) (14 pts)

Curse of the Flying Dutchman:

Wind from the S, blowing a fresh breeze.

Just for fun, take the Piece of cake.

(s) Piece of Cake	24 gun Corvette (average crew) (9 pts)
(f) Flying Dutchy	120 gun 3 Decker SOL (elite crew) (43 pts)

The South Pacific:

Wind from the S, blowing a strong breeze.

(a) USS Scurvy	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(b) HMS Tahiti	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Australian	32 gun Frigate (average crew) (9 pts)

(f) Bikini Atoll	7 gun Brig (crack crew) (4 pts)

Hornblower and the battle of Rosas

Wind from the E, blowing a fresh breeze.

The only battle Hornblower ever lost. He was able to dismast one ship and stern rake the others though. See if you can do as well.

(b) Sutherland	74 gun Ship of the Line (crack crew) (26 pts)
(f) Turenne	80 gun 3 Decker SOL (average crew) (27 pts)
(f) Nightmare	74 gun Ship of the Line (average crew) (24 pts)
(f) Paris	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Napolean	74 gun Ship of the Line (green crew) (20 pts)

Cape Horn:

Wind from the NE, blowing a strong breeze.

(a) Concord	80 gun Ship of the Line (average crew) (27 pts)
(a) Berkeley	98 gun 3 Decker SOL (crack crew) (28 pts)
(b) Thames	120 gun 3 Decker SOL (elite crew) (43 pts)
(s) Madrid	112 gun 3 Decker SOL (green crew) (27 pts)
(f) Musket	80 gun 3 Decker SOL (average crew) (27 pts)

New Orleans:

Wind from the SE, blowing a fresh breeze.

Watch that little Cypress go!

(a) Alligator	120 gun 3 Decker SOL (elite crew) (43 pts)
(b) Firefly	74 gun Ship of the Line (crack crew) (27 pts)
(b) Cypress	44 gun Frigate (elite crew) (14 pts)

Botany Bay:

Wind from the N, blowing a fresh breeze.

(b) Shark	64 gun Ship of the Line (average crew) (18 pts)
(f) Coral Snake	44 gun Corvette (elite crew) (24 pts)
(f) Sea Lion	44 gun Frigate (elite crew) (24 pts)

Voyage to the Bottom of the

Wind from the NW, blowing a fresh breeze.

This one is dedicated to Richard Basehart and David Hedison.

(a) Seaview	120 gun 3 Decker SOL (elite crew) (43 pts)
(a) Flying Sub	40 gun Frigate (crack crew) (17 pts)
(b) Mermaid	136 gun 3 Decker SOL (mutinous crew) (27 pts)
(s) Giant Squid	112 gun 3 Decker SOL (green crew) (27 pts)

Frigate Action:

Wind from the E, blowing a fresh breeze.

(a) Killdeer	40 gun Frigate (average crew) (15 pts)
(b) Sandpiper	40 gun Frigate (average crew) (15 pts)
(s) Curlew	38 gun Frigate (crack crew) (16 pts)

The Battle of Midway:

Wind from the E, blowing a moderate breeze.

(a) Enterprise	80 gun Ship of the Line (crack crew) (31 pts)
(a) Yorktown	80 gun Ship of the Line (average crew) (27 pts)
(a) Hornet	74 gun Ship of the Line (average crew) (24 pts)
(j) Akagi	112 gun 3 Decker SOL (green crew) (27 pts)
(j) Kaga	96 gun 3 Decker SOL (green crew) (24 pts)
(j) Soryu	80 gun Ship of the Line (green crew) (23 pts)

Star Trek:

Wind from the S, blowing a fresh breeze.

(a) Enterprise	450 gun Ship of the Line (elite crew) (75 pts)
(a) Yorktown	450 gun Ship of the Line (elite crew) (75 pts)
(a) Reliant	450 gun Ship of the Line (elite crew) (75 pts)
(a) Galileo	450 gun Ship of the Line (elite crew) (75 pts)
(k) Kobayashi Maru	450 gun Ship of the Line (elite crew) (75 pts)
(k) Klingon II	450 gun Ship of the Line (elite crew) (75 pts)
(o) Red Orion	450 gun Ship of the Line (elite crew) (75 pts)
(o) Blue Orion	450 gun Ship of the Line (elite crew) (75 pts)

CONCLUSION

sail has been a group effort.

Ken Arnold Code

curses library (pu!)

AUTHOR

Dave Riggle

CO-AUTHOR

Ed Wang

REFITTING

Craig Leres

CONSULTANTS

Chris Guthrie
Captain Happy
Horatio Nelson
Nancy Reagan
  and many valiant others...

REFERENCES

Wooden Ships & Iron Men, by Avalon Hill
Captain Horatio Hornblower Novels, (13 of them) by C.S. Forester
Captain Richard Bolitho Novels, (12 of them) by Alexander Kent
The Complete Works of Captain Frederick Marryat, (about 20) especially
  Mr. Midshipman Easy
  Peter Simple
  Jacob Faithful
  Japhet in Search of a Father
  Snarleyyow, or The Dog Fiend

Frank Mildmay, or The Naval Officer

SEE ALSO

midway(PUBLIC)

BUGS

Probably a few, and please report them to "riggle@ernie" and "edward@arpa."

NAME

snake, snscore − display chase game

SYNOPSIS

/usr/games/snake [**−w***n*] [**−l***n*]
/usr/games/snscore

DESCRIPTION

Snake is a display-based game which must be played on a CRT terminal from among those supported by vi(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **−l** and **−w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc	These keys are like hjkl but form a directed pad around the d key.
HJKL	These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.
SEFC	Likewise for the upper case versions on the left.
ATPB	These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.
x	This lets you quit the game at any time.
p	Points in a direction you might want to go.
w	Space warp to get out of tight squeezes, at a price.
!	Shell escape
^Z	Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run */usr/games/snscore* .

FILES

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played
/usr/games/busy	program to determine if system too busy

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[**–a**] file]

DESCRIPTION

trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

AUTHOR

Eric Allman

SEE ALSO

/usr/doc/trek

COMMAND SUMMARY

abandon	**capture**
cloak **up/d**own	
computer request; ...	**da**mages
destruct	**dock**
help	**i**mpulse course distance
lrscan	**m**ove course distance
phasers **a**utomatic amount	
phasers **m**anual amt1 course1 spread1 ...	
torpedo course [**y**es] angle/**n**o	
ram course distance	**r**est time
shell	**sh**ields **up/d**own
srscan [**y**es/**n**o]	
status	**terminate y**es/**n**o
undock	**v**isual course
warp warp_factor	

NAME

worm – Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms − animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**−field**] [**−length** #] [**−number** #] [**−trail**]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

−field makes a "field" for the worm(s) to eat; **−trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

SEE ALSO

Snails, by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of 'Hunt the wumpus.' A wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the wumpus with an arrow, meanwhile avoiding being eaten by the wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

NAME

zork – the game of dungeon

SYNOPSIS

/usr/games/zork

DESCRIPTION

Dungeon is a computer fantasy simulation based on Adventure and on Dungeons & Dragons, originally written by Lebling, Blank, and Anderson of MIT. In it you explore a dungeon made up of various rooms, caves, rivers, and so on. The object of the game is to collect as much treasure as possible and stow it safely in the trophy case (and, of course, to stay alive.)

Figuring out the rules is part of the game, but if you are stuck, you should start off with "open mailbox", "take leaflet", and then "read leaflet". Additional useful commands that are not documented include:

quit	(to end the game)
!cmd	(the usual shell escape convention)
>	(to save a game)
<	(to restore a game)

FILES

/usr/games/lib/d*