SYSTEM V/68
ADMINISTRATOR'S MANUAL

PRODUCT CODE 72900
41963-00

MOTOROLA
Computer Systems

Product Code 72912
41975G01

SYSTEM V/68 DOCUMENTATION SET

VOLUME I

SYSTEM V/68 USER'S REFERENCE MANUAL, 72905 (41968-00)

- Introduction
- Permuted Index
- Section 1 - Commands

VOLUME II

SYSTEM V/68 USER'S REFERENCE MANUAL, 72905 (41968-00)

- Section 2 - System Calls
- Section 3 - Subroutines
- Section 4 - File Formats
- Section 5 - Miscellaneous Facilities
- Section 6 - Games

VOLUME III

SYSTEM V/68 ADMINISTRATOR'S MANUAL, 72900 (41963-00)

- Introduction
- Permuted Index
- Section 1M - Commands
- Section 7 - Special Files
- Section 8 - Procedures

SYSTEM V/68 ADMINISTRATOR'S GUIDE, 72901 (41964-00)

- Introduction
- Administrative Guidelines
- Using the System
- Accounting
- File System Checking
- LP Spooling System
- System Activity Package

SYSTEM V/68 OPERATOR'S GUIDE, 72904 (41967-00)

- Chapter 1 - System Overview
- Chapter 2 - Getting Started
- Chapter 3 - Using the System
- Appendix A - System Specifications
- Appendix B - Error Messages

SYSTEM V/68 USER'S GUIDE, 72903 (41966-00)

- Introduction
- Primer
- Basics for Beginners
- Text Editors
- An Introduction to Shell
- Source Code Control System (SCCS)
- UNIX-to-UNIX CoPy: A Tutorial

VOLUME IV

SYSTEM V/68 PROGRAMMING GUIDE, 72908 (41971-00)

- Introduction
- An Introduction to Shell
- C Programming Language
- FORTRAN
- Curses and Terminfo Package
- Programming Language EFL

SYSTEM V/68 SUPPORT TOOLS GUIDE, 72909 (41972-000)

- Introduction
- Maintaining Computer Programs (MAKE)
- Augmented Version of MAKE
- The M4 Macro Processor
- The AWK Programming Language
- Desk Calculator Language (BC)
- Desk Calculator Program (DC)
- Lexical Analyzer Generator (LEX)
- Yet Another Compiler-Compiler (YACC)
- Common Object File Format

SYSTEM V/68 ASSEMBLER USER'S GUIDE, 72910 (41973-00)

- Introduction
- Warnings
- General Syntax Rules
- Segments, Location Counters, and Labels
- Types
- Expressions
- Pseudo-Operations
- Span-Dependent Optimization
- Address Mode Syntax
- Machine Instructions

SYSTEM V/68 COMMON LINK EDITOR REFERENCE MANUAL, 72911 (41974-00)

- Introduction
- Using the Link Editor
- Link Editor Command Language
- Notes and Special Procedures
- Error Messages
- Syntax Diagram for Input Directives

VOLUME V

SYSTEM V/68 DOCUMENT PROCESSING GUIDE, 72906 (41969-00)

- Introduction
- Advanced Editing
- Stream Editor
- Nroff and Troff User's Manual
- Table Formatting Program
- Mathematics Typesetting Program
- Memorandum Macros
- Viewgraphs and Slides Macros

SYSTEM V/68 ERROR MESSAGE MANUAL, 72902 (41965-00)

- Introduction
- Error Messages
- Index

SYSTEM V/68
ADMINISTRATOR'S MANUAL

Product Code 72900

Part Number 41963-00

Version 1

PREFACE

The *SYSTEM V/68 Administrator's Manual* (Part Number 41963-00, Product Code 72900) is intended to supplement the information contained in the *UNIX System V User's Manual* and to provide an easy reference volume for those who administer the UNIX-derived operating system.

While reasonable efforts have been made to assure the accuracy of this document, Motorola assumes no liability resulting from any omissions in this document or from the use of the information obtained therein. Motorola reserves the right to revise this document and to make changes from time to time in its content without being obligated to notify any person of such revision or changes.

INTRODUCTION

1. GENERAL

The *Administrator's Manual* is intended to supplement the information contained in the *UNIX System V User's Manual* and to provide an easy reference volume for those who must administer UNIX operating system. Accordingly, only those commands and descriptions deemed appropriate for system administrators have been included here.

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

2. ADMINISTRATOR'S MANUAL ORGANIZATION

2.1 Description of Contents

The manual is divided into three sections:

Section 1 ("System Maintenance Commands and Application Programs") contains system maintenance programs such as *fsck, mkfs,* etc., which generally reside in the directory **/etc;** these entries carry a sub-class designation of "1M" for cross-referencing reasons.

Section 7 ("Special Files") discusses the characteristics of each system file that actually refer to an input/output device. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 ("System Maintenance Procedures") discusses crash recovery and boot procedures, facility descriptions, etc.

2.2 Section Organization

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

The *UNIX System V User's Manual*, which contains sections 1 - 6, is organized in the same manner as this *Administrator's Manual*. Throughout the documentation, references to the contents of either manual are given as **name**(section). For example, **chroot**(1M) is a reference to the chroot entry in section 1M of the *Administrator's Manual*.

A table of contents and a permuted index derived from that table precede Section 1M. The permuted index contains entries from both the *UNIX System V User's Manual* and this volume, and on each "index" line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

2.3 Entry Format

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

NAME gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS summarizes the use of the program being described.

DESCRIPTION provides additional information about the program or facility outlined in the "Name" and "Synopsis" parts.

EXAMPLE gives an example(s) of usage, where appropriate.

FILES gives the filenames that are built into the program.

SEE ALSO gives pointers to related information.

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

WARNINGS points out potential pitfalls.

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

2.4 Conventions

A few conventions are used, particularly in Section 1 ("Commands"):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual. Note that this convention is not used in the "SYNOPSIS" or "SEE ALSO" part.

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

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

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

TABLE OF CONTENTS

1. System Maintenance Commands and Application Programs

intro . . . . . introduction to system maintenance commands and application programs
accept . . . . . allow/prevent LP requests
acct . . . . . overview of accounting and miscellaneous accounting commands
acctcms . . . . . command summary from per-process accounting records
acctcon . . . . . connect-time accounting
acctmerg . . . . . merge or add total accounting files
acctprc . . . . . process accounting
acctsh . . . . . shell procedures for accounting
bcopy . . . . . interactive block copy
brc . . . . . system initialization shell scripts
checkall . . . . . faster file system checking procedure
chroot . . . . . change root directory for a command
clri . . . . . clear inode
config.68 . . . . . configure SYSTEM V/68
cpset . . . . . install object files in binary directories
crash . . . . . examine system images
cron . . . . . clock daemon
dcopy . . . . . copy file systems for optimal access time
devnm . . . . . device name
df . . . . . report number of free disk blocks
dinit . . . . . disk initializer
diskusg . . . . . generate disk accounting data by user ID
errdead . . . . . extract error records from dump
errdemon . . . . . error-logging daemon
errpt . . . . . process a report of logged errors
errstop . . . . . terminate the error-logging daemon
ff . . . . . list filenames and statistics for a file system
filesave . . . . . daily/weekly SYSTEM V/68 file system backup
finc . . . . . fast incremental backup
frec . . . . . recover files from a backup tape
fsba . . . . . file system block analyzer
fsck . . . . . file system consistency check and interactive repair
fscv . . . . . convert files between M68000 and VAX-11/780 processors
fsdb . . . . . file system debugger
fuser . . . . . identify processes using a file or file structure
fwtmp . . . . . manipulate connect accounting records
getty . . . . . set terminal type, modes, speed, and line discipline
init . . . . . process control initialization
install . . . . . install commands
killall . . . . . kill all active processes
link . . . . . exercise link and unlink system calls
lpadmin . . . . . configure the LP spooling system
lpsched . . . . . start/stop the LP request scheduler and move requests
mkfs . . . . . construct a file system
mknod . . . . . build special file
mount . . . . . mount and dismount file system
mvdir . . . . . move a directory
ncheck . . . . . generate names from i-numbers
profiler . . . . . operating system profiler
pwck . . . . . password/group file checkers
runacct . . . . . run daily accounting

sadp . . . . . disk access profiler
sar . . . . . system activity report package
setmnt . . . . . establish mount table
shutdown . . . . . terminate all processing
sysdef . . . . . system definition
tic . . . . . terminfo compiler
trenter . . . . . enter a trouble report
uuclean . . . . . uucp spool directory clean-up
uusub . . . . . monitor uucp network
vm22fmt . . . . . format disks on the VM22 disk controller
volcopy . . . . . copy file systems with label checking
wall . . . . . write to all users
wffmt . . . . . format floppies for the VME/10 processor
whodo . . . . . who is doing what

7. Special Files

intro . . . . . introduction to special files
acia . . . . . Asynchronous Communications Interface Adapter
cm16 . . . . . 16Mb Cartridge Module Drive for Universal Disk Driver
cm80 . . . . . 80Mb Cartridge Module Drive for Universal Disk Driver
cmd16 . . . . . 16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver
cmd80 . . . . . 80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver
err . . . . . error-logging interface
fl8 . . . . . 8-inch Floppy Disk Drive for Universal Disk Driver
lark25 . . . . . 50Mb LARK Module Drive for VM21 Driver and VM22 Driver
lark8 . . . . . 16Mb LARK Module Drive for VM21 Driver and VM22 Driver
lp . . . . . MVME410 line printer interface
lrk25 . . . . . 25Mb LARK Module Drive for Universal Disk Driver
m400 . . . . . MVME400 Dual RS-232C Serial Port Module
mem . . . . . core memory
null . . . . . the null file
prf . . . . . operating system profiler
sa400fl22 . . . . . 5¼-inch Floppy Disk Drive for VM22 Driver
sa400flwd . . . . . 5¼-inch Floppy Disk Drive for the Winchester Disk Driver
sa800fl . . . . . 8-inch Floppy Disk Drive for VM21 Driver
sa800fl22 . . . . . 8-inch Floppy Disk Drive for VM22 Driver
sxt . . . . . pseudo-device driver
termio . . . . . general terminal interface
tty . . . . . controlling terminal interface
ud . . . . . general driver for all disk units supported by the M68KVM21 disk controller
v10graph . . . . . VME/10 graphics subsystem interface
vm21 . default general driver for all disk units supported by the M68KVM21 disk controller
vm22 . default general driver for all disk units supported by the M68KVM22 disk controller
wd15 . . . . . 15Mb Winchester Disk Drive
wd40 . . . . . 40Mb Winchester Disk Drive

8. System Maintenance Procedures

intro . . . . . introduction to system maintenance procedures
bo.macs . . . . . bootstrap operating procedure for system restart on EXORmacs
bo.vme . . . . . bootstrap operating procedure for system restart on VME/10
crash.macs . . . . . what to do when the system crashes
mk . . . . . how to remake the system and commands
ops.macs . . . . . EXORmacs operations

PERMUTED INDEX

wd15: 15Mb Winchester Disk Drive. . . . . wd15(7)
Driver. cm16: 16Mb Cartridge Module Drive for Universal Disk . . . cm16(7)
VM22 Driver. cmd16: 16Mb Cartridge Module Drive for VM21 Driver and . cmd16(7)
Driver. lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 lark8(7)
lrk25: 25Mb LARK Module Drive for Universal Disk Driver. lrk25(7)
hp: handle special functions of HP 2640 and 2621-series terminals. . . . . hp(1)
hp: handle special functions of HP 2640 and 2621-series terminals. . . . . hp(1)
300s terminals. 300, 300s: handle special functions of DASI 300 and . 300(1)
300, 300s: handle special functions of DASI 300 and 300s terminals. . . . . 300(1)
terminals. 300, 300s: handle special functions of DASI 300 and 300s . 300(1)
300, 300s: handle special functions of DASI 300 and 300s terminals. . . . . 300(1)
13tol, ltol3: convert between 3-byte integers and long integers. . . . . 13tol(3C)
diff3: 3-way differential file comparison. . . . . diff3(1)
4014: paginator for the Tektronix 4014 terminal. . . . 4014(1)
4014: paginator for the Tektronix 4014 terminal. . . . . 4014(1)
wd40: 40Mb Winchester Disk Drive. . . . . wd40(7)
terminal. 450: handle special functions of the DASI 450 . . . . 450(1)
450: handle special functions of the DASI 450 terminal. . . . . 450(1)
config.68: configure SYSTEM V/68. . . . . config.68(1M)
filesave, tapesave: daily/weekly SYSTEM V/68 file system backup. . . . . filesave(1M)
cu: call another SYSTEM V/68 system. . . . . cu(1C)
Driver. lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 lark25(7)
Disk Driver. sa400flwd: 5¼-inch Floppy Disk Drive for the Winchester . . . . sa400flwd(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22 Driver. . . . . sa400fl22(7)
f77: FORTRAN 77 compiler. . . . . f77(1)
Driver. cm80: 80Mb Cartridge Module Drive for Universal Disk . . . cm80(7)
VM22 Driver. cmd80: 80Mb Cartridge Module Drive for VM21 Driver and . cmd80(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . fl8(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21 Driver. . . . . sa800fl21(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22 Driver. . . . . sa800fl22(7)
base-64 ASCII string. a64l, l64a: convert between long integer and . . . . . a64l(3C)
abort: generate an IOT fault. . . . . abort(3C)
abort: terminate FORTRAN program. . . . . abort(3F)
value. abs, iabs, dabs, cabs, zabs: FORTRAN absolute . . . . . abs(3F)
abs: return integer absolute value. . . . . abs(3C)
abs: return integer absolute value. . . . . abs(3C)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . . abs(3F)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value functions. . . . . floor(3M)
accept, reject: allow/prevent LP requests. . . . . accept(1M)
touch: update access and modification times of a file. . . . . touch(1)
utime: set file access and modification times. . . . . utime(2)
access: determine accessibility of a file. . . . . access(2)
graphics: access graphical and numerical commands. . . . . graphics(1G)
fashion.. sputl, sgetl: access long integer data in a machine independent . . . sputl(3X)
sadp: disk access profiler. . . . . sadp(1)
ldfcn: common object file access routines. . . . . ldfcn(4)
dcopy: copy file systems for optimal access time. . . . . dcopy(1M)
getutline, pututline, setutent, endutent, utmpname: access utmp file entry. getutent, getutid, ... getut(3C)
access: determine accessibility of a file. . . . . access(2)
acct: enable or disable process accounting. . . . . acct(2)
acctcon1, acctcon2: connect-time accounting. . . . . acctcon(1M)
acctprc1, acctprc2: process accounting. . . . . acctprc(1M)
shutacct, startup, turnacct: shell procedures for accounting. /prctmp, prdaily, prtacct, runacct, ... acctsh(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and miscellaneous accounting commands. . acct(1M)
acctwtmp: overview of accounting and miscellaneous accounting commands. acctdisk, acctdusg, accton, . . . acct(1M)
diskusg: generate disk accounting data by user ID. . . . . diskusg(1M)
acct: per-process accounting file format. . . . . acct(4)
acctcom: search and print process accounting file(s). . . . . acctcom(1)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
mclock: return FORTRAN time accounting. . . . . mclock(3F)
acctcms: command summary from per-process accounting records. . . . . acctcms(1M)
fwtmp, wtmpfix: manipulate connect accounting records. . . . . fwtmp(1M)
runacct: run daily accounting. . . . . runacct(1M)
acct: enable or disable process accounting. . . . . acct(2)
acct: per-process accounting file format. . . . . acct(4)
accounting records. acctcms: command summary from per-process . . . . acctcms(1M)
file(s). acctcom: search and print process accounting . . . . . acctcom(1)
acctcon1, acctcon2: connect-time accounting. . . . . acctcon(1M)
acctcon1, acctcon2: connect-time accounting. . . . . acctcon(1M)
accounting and miscellaneous accounting commands. acctdisk, acctdusg, accton, acctwtmp: overview of . . . acct(1M)

and miscellaneous accounting commands. acctdisk, acctdusg, accton, acctwtmp: overview of accounting . . acct(1M)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
miscellaneous accounting/ acctdisk, acctdusg, accton, acctwtmp: overview of accounting and . . . . acct(1M)
acctprc1, acctprc2: process accounting. . . . . acctprc(1M)
acctprc1, acctprc2: process accounting. . . . . acctprc(1M)
accounting commands. acctdisk, acctdusg, accton, acctwtmp: overview of accounting and miscellaneous . acct(1M)
Adapter. acia: Asynchronous Communications Interface . . . . acia(7)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. . . . . trig(3M)
acos, dacos: FORTRAN arccosine intrinsic function. . . acos(3F)
killall: kill all active processes. . . . . killall(1M)
sag: system activity graph. . . . . sag(1G)
sa1, sa2, sadc: system activity report package. . . . . sar(1M)
sar: system activity reporter. . . . . sar(1)
sact: print current SCCS file editing activity. . . . . sact(1)
time a command; report process data and system activity. timex: . . . . . timex(1)
acia: Asynchronous Communications Interface Adapter. . . . . acia(7)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
putenv: change or add value to environment. . . . . putenv(3C)
admin: create and administer SCCS files. . . . . admin(1)
admin: create and administer SCCS files. . . . . admin(1)
argument. aimag, dimag: FORTRAN imaginary part of complex . aimag(3F)
function. aint, dint: FORTRAN integer part intrinsic . . . . . aint(3F)
alarm: set a process's alarm clock. . . . . alarm(2)
alarm: set a process's alarm clock. . . . . alarm(2)
crypt, encrypt: a one way hashing encryption algorithm. . . . . crypt(3C)
brk, sbrk: change data segment space allocation. . . . . brk(2)
malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)
calloc, mallopt, mallinfo: fast main memory allocator. malloc, free, realloc, ... malloc(3X)
accept, reject: allow/prevent LP requests. . . . . accept(1M)
intrinsic function. log, alog, dlog, clog: FORTRAN natural logarithm . . . . . log(3F)
function. log10, alog10, dlog10: FORTRAN common logarithm intrinsic log10(3F)
functions. max, max0, amax0, max1, dmax1: FORTRAN maximum-value max(3F)
max, max0, amax0, max1, amax1, dmax1: FORTRAN maximum-value functions. . max(3F)
functions. min, min0, amin0, min1, dmin1: FORTRAN minimum-value min(3F)
min, min0, amin0, min1, amin1, dmin1: FORTRAN minimum-value functions. . min(3F)
functions. mod, amod, dmod: FORTRAN remaindering intrinsic . . . . mod(3F)
fsba: file system block analyzer. . . . . fsba(1M)
Boolean functions. and, or, xor, not, lshift, rshift: FORTRAN bitwise . . . bool(3F)
sort: sort and/or merge files. . . . . sort(1)
functions. anint, dnint, nint, idnint: FORTRAN nearest integer . . round(3F)
a.out: common assembler and link editor output. . . . a.out(4)
aouthdr: optional aout header. . . . . aouthdr(4)
aouthdr: optional aout header. . . . . aouthdr(4)
intro: introduction to commands and application programs. . . . . intro(1)
introduction to system maintenance commands and application programs. intro: . . . . . intro(1M)
archives. ar: archive and library maintainer for portable . . . . ar(1)
ar: common archive file format. . . . . ar(4)
bc: arbitrary-precision arithmetic language. . . . . bc(1)
acos, dacos: FORTRAN arccosine intrinsic function. . . . . acos(3F)
archives. ar: archive and library maintainer for portable . . . . . ar(1)
cpio: format of cpio archive. . . . . cpio(4)
ar: common archive file format. . . . . ar(4)
ldahread: read the archive header of a member of an archive file. . . . . ldahread(3X)
ldahread: read the archive header of a member of an archive file. . . . ldahread(3X)
tar: tape file archiver. . . . . tar(1)
ar: archive and library maintainer for portable archives. . . . . ar(1)
cpio: copy file archives in and out. . . . . cpio(1)
asin, dasin: FORTRAN arcsine intrinsic function. . . . . asin(3F)
atan2, datan2: FORTRAN arctangent intrinsic function. . . . . atan2(3F)
atan, datan: FORTRAN arctangent intrinsic function. . . . . atan(3F)
aimag, dimag: FORTRAN imaginary part of complex argument. . . . . aimag(3F)
getarg: return FORTRAN command-line argument. . . . . getarg(3F)
varargs: handle variable argument list. . . . . varargs(5)
vsprintf: print formatted output of a varargs argument list. vprintf, vfprintf, ... vprintf(3S)
xargs: construct argument list(s) and execute command. . . . . xargs(1)
getopt: get option letter from argument vector. . . . . getopt(3C)
expr: evaluate arguments as an expression. . . . . expr(1)
echo: echo arguments. . . . . echo(1)
bc: arbitrary-precision arithmetic language. . . . . bc(1)
arithmetic: provide drill in number facts. . . . . arithmetic(6)
expr: evaluate arguments as an expression. . . . . expr(1)
as: common assembler. . . . . as(1)
asa: interpret ASA carriage control characters. . . . . asa(1)
asa: interpret ASA carriage control characters. . . . asa(1)
ascii: map of ASCII character set. . . . . ascii(5)

ascii: map of ASCII character set. . . . . ascii(5)
164a: convert between long integer and base-64 ASCII string. a64l, ... a64l(3C)
atof: convert ASCII string to floating-point number. . . . . atof(3C)
ctime, localtime, gmtime, asctime, tzset: convert date and time to string. . . . . ctime(3C)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. . . . . trig(3M)
asin, dasin: FORTRAN arcsine intrinsic function. . . . . asin(3F)
help: ask for help. . . . . help(1)
a.out: common assembler and link editor output. . . . . a.out(4)
as: common assembler. . . . . as(1)
assert: verify program assertion. . . . . assert(3X)
assert: verify program assertion. . . . . assert(3X)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
acia: Asynchronous Communications Interface Adapter. . . acia(7)
at, batch: execute commands at a later time. . . . . at(1)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. . . . . trig(3M)
atan, datan: FORTRAN arctangent intrinsic function. . atan(3F)
function. atan2, datan2: FORTRAN arctangent intrinsic . . . . atan2(3F)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. . . . . trig(3M)
number. atof: convert ASCII string to floating-point . . . . . atof(3C)
strtod, atof: convert string to double-precision number. . . . . strtod(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
wait: await completion of process. . . . . wait(1)
awk: pattern scanning and processing language. . . . . awk(1)
ungetc: push character back into input stream. . . . . ungetc(3S)
back: the game of backgammon. . . . . back(6)
back: the game of backgammon. . . . . back(6)
filesave, tapesave: daily/weekly SYSTEM V/68 file system backup. . . . . filesave(1M)
finc: fast incremental backup. . . . . finc(1M)
frec: recover files from a backup tape. . . . . frec(1M)
banner: make posters. . . . . banner(1)
terminfo: terminal capability data base. . . . . terminfo(4)
a64l, 164a: convert between long integer and base-64 ASCII string. . . . . a64l(3C)
vi: screen-oriented (visual) display editor based on ex. . . . . vi(1)
basename, dirname: deliver portions of pathnames. . . basename(1)
at, batch: execute commands at a later time. . . . . at(1)
bc: arbitrary-precision arithmetic language. . . . . bc(1)
shell scripts. brc, bcheckrc, rc, powerfail: system initialization . . . . . brc(1M)
bcopy: interactive block copy. . . . . bcopy(1M)
bdiff: file comparator for large files. . . . . bdiff(1)
cb: C program beautifier. . . . . cb(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
bfs: big file scanner. . . . . bfs(1)
cpset: install object files in binary directories. . . . . cpset(1M)
fread, fwrite: binary input/output. . . . . fread(3S)
bsearch: binary search. . . . . bsearch(3C)
tsearch, tfind, tdelete, twalk: manage binary search trees. . . . . tsearch(3C)
and, or, xor, not, lshift, rshift: FORTRAN bitwise Boolean functions. . . . . bool(3F)
bj: the game of blackjack. . . . . bj(6)
bj: the game of blackjack. . . . . bj(6)
fsba: file system block analyzer. . . . . fsba(1M)
bcopy: interactive block copy. . . . . bcopy(1M)
sum: print checksum and block count of a file. . . . . sum(1)
sync: update the super block. . . . . sync(1)
df: report number of free disk blocks. . . . . df(1M)
restart on EXORmacs. bo.macs: bootstrap operating procedure for system . . . bo.macs(8)
and, or, xor, not, lshift, rshift: FORTRAN bitwise Boolean functions. . . . . bool(3F)
EXORmacs. bo.macs: bootstrap operating procedure for system restart on . . bo.macs(8)
VME/10. bo.vme: bootstrap operating procedure for system restart on . . bo.vme(8)
restart on VME/10. bo.vme: bootstrap operating procedure for system . . . bo.vme(8)
shell scripts. brc, bcheckrc, rc, powerfail: system initialization . . . brc(1M)
brk, sbrk: change data segment space allocation. . . . . brk(2)
programs. bs: a compiler/interpreter for modest-sized . . . . . bs(1)
bsearch: binary search. . . . . bsearch(3C)
stdio: standard buffered input/output package. . . . . stdio(3S)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
mknod: build special file. . . . . mknod(1M)
swab: swap bytes. . . . . swab(3C)
cc: C compiler. . . . . cc(1)
scc: C compiler for stand-alone programs. . . . . scc(1)
cflow: generate C flow graph. . . . . cflow(1)
cpp: the C language preprocessor. . . . . cpp(1)
cb: C program beautifier. . . . . cb(1)
lint: a C program checker. . . . . lint(1)
cxref: generate C program cross-reference. . . . . cxref(1)

ctrace: C program debugger. . . . . ctrace(1)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . . abs(3F)
cal: print calendar. . . . . cal(1)
dc: desk calculator. . . . . dc(1)
cal: print calendar. . . . . cal(1)
calendar: reminder service. . . . . calendar(1)
cu: call another SYSTEM V/68 system. . . . . cu(1C)
stat: data returned by stat system call. . . . . stat(5)
malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)
allocator. malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory . . . . . malloc(3X)
intro: introduction to system calls and error numbers. . . . . intro(2)
link, unlink: exercise link and unlink system calls. . . . . link(1M)
lp, cancel: send/cancel requests to an LP line printer. . . . lp(1)
terminfo: terminal capability data base. . . . . terminfo(4)
pnch: file format for card images. . . . . pnch(4)
asa: interpret ASA carriage control characters. . . . . asa(1)
cm16: 16Mb Cartridge Module Drive for Universal Disk Driver. . . cm16(7)
cm80: 80Mb Cartridge Module Drive for Universal Disk Driver. . . cm80(7)
Driver. cmd16: 16Mb Cartridge Module Drive for VM21 Driver and VM22 . cmd16(7)
Driver. cmd80: 80Mb Cartridge Module Drive for VM21 Driver and VM22 . cmd80(7)
edit: text editor (variant of ex for casual users). . . . . edit(1)
cat: concatenate and print files. . . . . cat(1)
cb: C program beautifier. . . . . cb(1)
cc: C compiler. . . . . cc(1)
cos, dcos, ccos: FORTRAN cosine intrinsic function. . . . . cos(3F)
cd: change working directory. . . . . cd(1)
cdc: change the delta commentary of an SCCS delta. . . cdc(1)
absolute value functions. floor, ceil, fmod, fabs: floor, ceiling, remainder, ... floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value functions. . . . . floor(3M)
exp, dexp, cexp: FORTRAN exponential intrinsic function. . . . . exp(3F)
cflow: generate C flow graph. . . . . cflow(1)
delta: make a delta (change) to an SCCS file. . . . . delta(1)
pipe: create an interprocess channel. . . . . pipe(2)
real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type conversion. /idint, . . . ftype(3F)
ungetc: push character back into input stream. . . . . ungetc(3S)
cuserid: get character login name of the user. . . . . cuserid(3S)
getc, getchar, fgetc, getw: get character or word from stream. . . . . getc(3S)
putc, putchar, fputc, putw: put character or word on a stream. . . . . putc(3S)
ascii: map of ASCII character set. . . . . ascii(5)
asa: interpret ASA carriage control characters. . . . . asa(1)
tolower, _toupper, _tolower, toascii: translate characters. toupper, ... conv(3C)
isprint, isgraph, iscntrl, isascii: classify characters. /isxdigit, isalnum, isspace, ispunct, ... ctype(3C)
tr: translate characters. . . . . tr(1)
nulladm, prctmp, prdaily, prtacct, runacct,/ chargefee, ckpacct, dodisk, lastlogin, monacct, ... acctsh(1M)
chdir: change working directory. . . . . chdir(2)
fsck, dfsck: file system consistency check and interactive repair. . . . . fsck(1M)
checkall: faster file system checking procedure. . . . . checkall(1M)
lint: a C program checker. . . . . lint(1)
pwck, grpck: password/group file checkers. . . . . pwck(1M)
checkall: faster file system checking procedure. . . . . checkall(1M)
volcopy, labelit: copy file systems with label checking. . . . . volcopy(1M)
checklist: list of file systems processed by fsck. . . . checklist(4)
sum: print checksum and block count of a file. . . . . sum(1)
chess: the game of chess. . . . . chess(6)
chess: the game of chess. . . . . chess(6)
chown, chgrp: change owner or group. . . . . chown(1)
times: get process and child process times. . . . . times(2)
wait: wait for child process to stop or terminate. . . . . wait(2)
chmod: change mode. . . . . chmod(1)
chmod: change mode of file. . . . . chmod(2)
chown: change owner and group of a file. . . . . chown(2)
chown, chgrp: change owner or group. . . . . chown(1)
chroot: change root directory. . . . . chroot(2)
chroot: change root directory for a command. . . . . chroot(1M)
prctmp, prdaily, prtacct, runacct,/ chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, ... acctsh(1M)
ispunct, isprint, isgraph, iscntrl, isascii: classify characters. /isxdigit, isalnum, isspace, ... ctype(3C)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
clri: clear inode. . . . . clri(1M)
ferror, feof, clearerr, fileno: stream status inquiries. . . . . ferror(3S)
alarm: set a process's alarm clock. . . . . alarm(2)
cron: clock daemon. . . . . cron(1M)
clock: report CPU time used. . . . . clock(3C)
log, alog, dlog, clog: FORTRAN natural logarithm intrinsic function. . log(3F)
ldclose, ldaclose: close a common object file. . . . . ldclose(3X)
close: close a file descriptor. . . . . close(2)

close: close a file descriptor. . . . close(2)
fclose, fflush: close or flush a stream. . . . fclose(3S)
clri: clear inode. . . . clri(1M)
Disk Driver. cm16: 16Mb Cartridge Module Drive for Universal . . cm16(7)
Disk Driver. cm80: 80Mb Cartridge Module Drive for Universal . . cm80(7)
and VM22 Driver. cmd16: 16Mb Cartridge Module Drive for VM21 Driver cmd16(7)
and VM22 Driver. cmd80: 80Mb Cartridge Module Drive for VM21 Driver cmd80(7)
cmp: compare two files. . . . cmp(1)
int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type/ . ftype(3F)
col: filter reverse line feeds. . . . col(1)
comb: combine SCCS deltas. . . . comb(1)
comb: combine SCCS deltas. . . . comb(1)
files. comm: select, or reject lines common to two sorted . . comm(1)
nice: run a command at low priority. . . . nice(1)
chroot: change root directory for a command. . . . chroot(1M)
env: set environment for command execution. . . . env(1)
uux: UNIX-to-UNIX system command execution. . . . uux(1C)
system: issue a shell command from FORTRAN. . . . system(3F)
nohup: run a command immune to hangups and quits. . . . nohup(1)
getopt: parse command options. . . . getopt(1)
sh, rsh: shell, the standard/restricted command programming language. . . . sh(1)
timex: time a command; report process data and system activity. . . timex(1)
records. acctcms: command summary from per-process accounting . . . acctcms(1M)
system: issue a shell command. . . . system(3S)
test: condition evaluation command. . . . test(1)
time: time a command. . . . time(1)
xargs: construct argument list(s) and execute command. . . . xargs(1)
getarg: return FORTRAN command-line argument. . . . getarg(3F)
overview of accounting and miscellaneous accounting commands. acctdisk, acctdusg, accton, acctwtmp: . . . acct(1M)
intro: introduction to commands and application programs. . . . intro(1)
intro: introduction to system maintenance commands and application programs. . . . intro(1M)
at, batch: execute commands at a later time. . . . at(1)
graphics: access graphical and numerical commands. . . . graphics(1G)
install: install commands. . . . install(1M)
mk: how to remake the system and commands. . . . mk(8)
stat: statistical network useful with graphical commands. . . . stat(1G)
cdc: change the delta commentary of an SCCS delta. . . . cdc(1)
ar: common archive file format. . . . ar(4)
a.out: common assembler and link editor output. . . . a.out(4)
as: common assembler. . . . as(1)
log10, alog10, dlog10: FORTRAN common logarithm intrinsic function. . . . log10(3F)
ldfcn: common object file access routines. . . . ldfcn(4)
ldopen, ldaopen: open a common object file for reading. . . . ldopen(3X)
ldlitem: manipulate line number entries of a common object file function. ldlread, ldlinit, . . . ldlread(3X)
ldclose, ldaclose: close a common object file. . . . ldclose(3X)
ldfhread: read the file header of a common object file. . . . ldfhread(3X)
seek to line number entries of a section of a common object file. ldlseek, ldnlseek: . . . ldlseek(3X)
ldohseek: seek to the optional file header of a common object file. . . . ldohseek(3X)
seek to relocation entries of a section of a common object file. ldrseek, ldnrseek: . . . ldrseek(3X)
read an indexed/named section header of a common object file. ldshread, ldnshread: . . . ldshread(3X)
ldnsseek: seek to an indexed/named section of a common object file. ldsseek, . . . ldsseek(3X)
compute the index of a symbol table entry of a common object file. ldtbindex: . . . ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object file. . . . ldtbread(3X)
ldtbseek: seek to the symbol table of a common object file. . . . ldtbseek(3X)
linenum: line number entries in a common object file. . . . linenum(4)
nm: print name list of common object file. . . . nm(1)
reloc: relocation information for a common object file. . . . reloc(4)
scnhdr: section header for a common object file. . . . scnhdr(4)
syms: common object file symbol table format. . . . syms(4)
filehdr: file header for common object files. . . . filehdr(4)
ld: link editor for common object files. . . . ld(1)
size: print section sizes of common object files. . . . size(1)
comm: select or reject lines common to two sorted files. . . . comm(1)
ipcs: report inter-process communication facilities status. . . . ipcs(1)
stdipc: standard interprocess communication package. . . . stdipc(3C)
acia: Asynchronous Communications Interface Adapter. . . . acia(7)
diff: differential file comparator. . . . diff(1)
bdiff: file comparator for large files. . . . bdiff(1)
cmp: compare two files. . . . cmp(1)
sccsdiff: compare two versions of an SCCS file. . . . sccsdiff(1)
lge, lgt, lle, llt: string comparision intrinsic functions. . . . strcmp(3F)
diff3: 3-way differential file comparison. . . . diff3(1)
dircmp: directory comparison. . . . dircmp(1)
regcmp, regex: compile and execute a regular expression. . . . regcmp(3X)
regexp: regular expression compile and match routines. . . . regexp(5)

regcmp: regular expression compile. . . . . regcmp(1)
term: format of compiled term file. . . . . term(4)
cc: C compiler. . . . . cc(1)
f77: FORTRAN 77 compiler. . . . . f77(1)
scc: C compiler for stand-alone programs. . . . . scc(1)
tic: terminfo compiler. . . . . tic(1M)
yacc: yet another compiler-compiler. . . . . yacc(1)
bs: a compiler/interpreter for modest-sized programs. . . . bs(1)
erf, erfc: error function and complementary error function. . . . . erf(3M)
wait: await completion of process. . . . . wait(1)
aimag, dimag: FORTRAN imaginary part of complex argument. . . . . aimag(3F)
conjg, dconjg: FORTRAN complex conjugate intrinsic function. . . . . conjg(3F)
pack, pcat, unpack: compress and expand files. . . . . pack(1)
common object file. ldtbindex: compute the index of a symbol table entry of a . . . . ldtbindex(3X)
cat: concatenate and print files. . . . . cat(1)
test: condition evaluation command. . . . . test(1)
config.68: configure SYSTEM V/68. . . . . config.68(1M)
config.68: configure SYSTEM V/68. . . . . config.68(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
function. conjg, dconjg: FORTRAN complex conjugate intrinsic . conjg(3F)
conjg, dconjg: FORTRAN complex conjugate intrinsic function. . . . . conjg(3F)
fwtmp, wtmpfix: manipulate connect accounting records. . . . . fwtmp(1M)
dial: establish an out-going terminal line connection. . . . . dial(3C)
acctcon1, acctcon2: connect-time accounting. . . . . acctcon(1M)
fsck, dfsck: file system consistency check and interactive repair. . . . . fsck(1M)
rjestat: RJE status report and interactive status console. . . . . rjestat(1C)
math: math functions and constants. . . . . math(5)
mkfs: construct a file system. . . . . mkfs(1M)
xargs: construct argument list(s) and execute command. . . . xargs(1)
ls: list contents of directories. . . . . ls(1)
toc: graphical table of contents routines. . . . . toc(1G)
csplit: context split. . . . . csplit(1)
asa: interpret ASA carriage control characters. . . . . asa(1)
ioctl: control device. . . . . ioctl(2)
fcntl: file control. . . . . fcntl(2)
init, telinit: process control initialization. . . . . init(1M)
msgctl: message control operations. . . . . msgctl(2)
semctl: semaphore control operations. . . . . semctl(2)
shmctl: shared memory control operations. . . . . shmctl(2)
fcntl: file control options. . . . . fcntl(5)
uustat: uucp status inquiry and job control. . . . . uustat(1C)
vc: version control. . . . . vc(1)
for all disk units supported by the M68K VM21 disk controller. ud: general driver . . . . . ud(7)
for all disk units supported by the M68K VM21 disk controller. vm21: default general driver . . . . . vm21(7)
for all disk units supported by the M68K VM22 disk controller. vm22: default general driver . . . . . vm22(7)
vm22fmt: format disks on the VM22 disk controller. . . . . vm22fmt(1M)
tty: controlling terminal interface. . . . . tty(7)
conv: object file converter. . . . . conv(1)
term: conventional names for terminals. . . . . term(5)
cmplx, dcmplx, ichar, char: explicit FORTRAN type conversion. /ifix, idint, real, float, sngl, dble, ... ftype(3F)
units: conversion program. . . . . units(1)
dd: convert and copy a file. . . . . dd(1)
atof: convert ASCII string to floating-point number. . . . atof(3C)
l3tol, ltol3: convert between 3-byte integers and long integers. . . l3tol(3C)
string. a64l, l64a: convert between long integer and base-64 ASCII . . . a64l(3C)
ctime, localtime, gmtime, asctime, tzset: convert date and time to string. . . . . ctime(3C)
processors. fscv: convert files between M68000 and VAX-11/780 . . . fscv(1M)
ecvt, fcvt, gcvt: convert floating-point number to string. . . . . ecvt(3C)
scanf, fscanf, sscanf: convert formatted input. . . . . scanf(3S)
strtod, atof: convert string to double-precision number. . . . . strtod(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
conv: object file converter. . . . . conv(1)
dd: convert and copy a file. . . . . dd(1)
bcopy: interactive block copy. . . . . bcopy(1M)
cpio: copy file archives in and out. . . . . cpio(1)
dcopy: copy file systems for optimal access time. . . . . dcopy(1M)
volcopy, labelit: copy file systems with label checking. . . . . volcopy(1M)
cp, ln, mv: copy, link or move files. . . . . cp(1)
uucp, uulog, uuname: unix to unix copy. . . . . uucp(1C)
uupick: public UNIX System-to-UNIX System file copy. uuto, ... uuto(1C)
core: format of core image file. . . . . core(4)
core: format of core image file. . . . . core(4)
mem, kmem: core memory. . . . . mem(7)
cos, dcos, ccos: FORTRAN cosine intrinsic function. . . cos(3F)
functions. sin, cos, tan, asin, acos, atan, atan2: trigonometric . . . . . trig(3M)

function. cosh, dcosh: FORTRAN hyperbolic cosine intrinsic . . . cosh(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)
cos, dcos, ccos: FORTRAN cosine intrinsic function. . . . . cos(3F)
cosh, dcosh: FORTRAN hyperbolic cosine intrinsic function. . . . . cosh(3F)
sum: print checksum and block count of a file. . . . . sum(1)
wc: word count. . . . . wc(1)
cp, ln, mv: copy, link or move files. . . . . cp(1)
cpio: format of cpio archive. . . . . cpio(4)
cpio: copy file archives in and out. . . . . cpio(1)
cpio: format of cpio archive. . . . . cpio(4)
cpp: the C language preprocessor. . . . . cpp(1)
cpset: install object files in binary directories. . . . . cpset(1M)
clock: report CPU time used. . . . . clock(3C)
craps: the game of craps. . . . . craps(6)
craps: the game of craps. . . . . craps(6)
crash: examine system images. . . . . crash(1M)
crash: what to do when the system crashes. . . . . crash.macs(8)
crash: what to do when the system crashes. . . . . crash.macs(8)
one. creat: create a new file or rewrite an existing . . . . . creat(2)
tmpnam, tempnam: create a name for a temporary file. . . . . tmpnam(3S)
creat: create a new file or rewrite an existing one. . . . . creat(2)
fork: create a new process. . . . . fork(2)
tmpfile: create a temporary file. . . . . tmpfile(3S)
pipe: create an interprocess channel. . . . . pipe(2)
admin: create and administer SCCS files. . . . . admin(1)
umask: set and get file creation mask. . . . . umask(2)
cron: clock daemon. . . . . cron(1M)
crontab: user crontab file. . . . . crontab(1)
crontab: user crontab file. . . . . crontab(1)
cxref: generate C program cross-reference. . . . . cxref(1)
curses: CRT screen handling and optimization package. . . . . curses(3X)
algorithm. crypt, encrypt: a one way hashing encryption . . . . crypt(3C)
sin, dsin, csin: FORTRAN sine intrinsic function. . . . . sin(3F)
csplit: context split. . . . . csplit(1)
sqrt, dsqrt, csqrt: FORTRAN square root intrinsic function. . . . . sqrt(3F)
ct: spawn getty to a remote terminal. . . . . ct(1C)
ctermid: generate filename for terminal. . . . . ctermid(3S)
date and time to string. ctime, localtime, gmtime, asctime, tzset: convert . . . ctime(3C)
ctrace: C program debugger. . . . . ctrace(1)
cu: call another SYSTEM V/68 system. . . . . cu(1C)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
uname: get name of current operating system. . . . . uname(2)
sact: print current SCCS file editing activity. . . . . sact(1)
uname: print name of current UNIX System. . . . . uname(1)
ttyslot: find the slot in the utmp file of the current user. . . . . ttyslot(3C)
getcwd: get pathname of current working directory. . . . . getcwd(3C)
package. curses: CRT screen handling and optimization . . . . curses(3X)
spline: interpolate smooth curve. . . . . spline(1G)
cuserid: get character login name of the user. . . . . cuserid(3S)
file. cut: cut out selected fields of each line of a . . . . . cut(1)
cut: cut out selected fields of each line of a file. . . . . cut(1)
cxref: generate C program cross-reference. . . . . cxref(1)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . . abs(3F)
acos, dacos: FORTRAN arccosine intrinsic function. . . . . acos(3F)
cron: clock daemon. . . . . cron(1M)
errdemon: error-logging daemon. . . . . errdemon(1M)
errstop: terminate the error-logging daemon. . . . . errstop(1M)
lpd: line printer daemon. . . . . lpd(1C)
runacct: run daily accounting. . . . . runacct(1M)
filesave, tapesave: daily/weekly SYSTEM V/68 file system backup. . . . filesave(1M)
300, 300s: handle special functions of DASI 300 and 300s terminals. . . . . 300(1)
450: handle special functions of the DASI 450 terminal. . . . . 450(1)
asin, dasin: FORTRAN arcsine intrinsic function. . . . . asin(3F)
timex: time a command; report process data and system activity. . . . . timex(1)
terminfo: terminal capability data base. . . . . terminfo(4)
diskusg: generate disk accounting data by user ID. . . . . diskusg(1M)
sputl, sgetl: access long integer data in a machine independent fashion. . . . . sputl(3X)
plock: lock process, text, or data in memory. . . . . plock(2)
prof: display profile data. . . . . prof(1)
stat: data returned by stat system call. . . . . stat(5)
brk, sbrk: change data segment space allocation. . . . . brk(2)
types: primitive system data types. . . . . types(5)
join: relational database operator. . . . . join(1)
tput: query terminfo database. . . . . tput(1)
atan, datan: FORTRAN arctangent intrinsic function. . . . . atan(3F)

atan2, datan2: FORTRAN arctangent intrinsic function. . . . atan2(3F)
ctime, localtime, gmtime, asctime, tzset: convert date and time to string. . . . . ctime(3C)
date: print and set the date. . . . . date(1)
date: print and set the date. . . . . date(1)
type/ int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN . . ftype(3F)
dc: desk calculator. . . . . dc(1)
int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type/ . . . . . ftype(3F)
function. conjg, dconjg: FORTRAN complex conjugate intrinsic . . . . conjg(3F)
dcopy: copy file systems for optimal access time. . . . dcopy(1M)
cos, dcos, ccos: FORTRAN cosine intrinsic function. . . . cos(3F)
function. cosh, dcosh: FORTRAN hyperbolic cosine intrinsic . . . . . cosh(3F)
dd: convert and copy a file. . . . . dd(1)
functions. dim, ddim, idim: positive difference intrinsic . . . . . dim(3F)
ctrace: C program debugger. . . . . ctrace(1)
fsdb: file system debugger. . . . . fsdb(1M)
sdb: symbolic debugger. . . . . sdb(1)
by the M68KVM21 disk controller. vm21: default general driver for all disk units supported . . vm21(7)
by the M68KVM22 disk controller. vm22: default general driver for all disk units supported . . vm22(7)
sysdef: system definition. . . . . sysdef(1M)
basename, dirname: deliver portions of pathnames. . . . . basename(1)
tail: deliver the last part of a file. . . . . tail(1)
cdc: change the delta commentary of an SCCS delta. . . . . cdc(1)
delta: make a delta (change) to an SCCS file. . . . . delta(1)
cdc: change the delta commentary of an SCCS delta. . . . . cdc(1)
rmdel: remove a delta from an SCCS file. . . . . rmdel(1)
delta: make a delta (change) to an SCCS file. . . . . delta(1)
comb: combine SCCS deltas. . . . . comb(1)
mesg: permit or deny messages. . . . . mesg(1)
close: close a file descriptor. . . . . close(2)
dup: duplicate an open file descriptor. . . . . dup(2)
dc: desk calculator. . . . . dc(1)
access: determine accessibility of a file. . . . . access(2)
file: determine file type. . . . . file(1)
dfile: device information file. . . . . dfile(4)
master: master device information table. . . . . master.dec(4)
ioctl: control device. . . . . ioctl(2)
devnm: device name. . . . . devnm(1M)
hpd, erase, hardcopy, tekset, td: graphical device routines and filters. . . . . gdev(1G)
devnm: device name. . . . . devnm(1M)
exp, dexp, cexp: FORTRAN exponential intrinsic function. . exp(3F)
df: report number of free disk blocks. . . . . df(1M)
dfile: device information file. . . . . dfile(4)
interactive repair. fsck, dfsck: file system consistency check and . . . . . fsck(1M)
connection. dial: establish an out-going terminal line . . . . . dial(3C)
ratfor: rational FORTRAN dialect. . . . . ratfor(1)
diff: differential file comparator. . . . . diff(1)
diff3: 3-way differential file comparison. . . . . diff3(1)
dim, ddim, idim: positive difference intrinsic functions. . . . . dim(3F)
sdiff: side-by-side difference program. . . . . sdiff(1)
diffmk: mark differences between files. . . . . diffmk(1)
diff: differential file comparator. . . . . diff(1)
diff3: 3-way differential file comparison. . . . . diff3(1)
diffmk: mark differences between files. . . . . diffmk(1)
functions. dim, ddim, idim: positive difference intrinsic . . . . . dim(3F)
aimag, dimag: FORTRAN imaginary part of complex argument. aimag(3F)
dinit: disk initializer. . . . . dinit(1M)
aint, dint: FORTRAN integer part intrinsic function. . . . aint(3F)
dir: format of directories. . . . . dir(4)
dircmp: directory comparison. . . . . dircmp(1)
cpset: install object files in binary directories. . . . . cpset(1M)
dir: format of directories. . . . . dir(4)
ls: list contents of directories. . . . . ls(1)
rm, rmdir: remove files or directories. . . . . rm(1)
cd: change working directory. . . . . cd(1)
chdir: change working directory. . . . . chdir(2)
chroot: change root directory. . . . . chroot(2)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
dircmp: directory comparison. . . . . dircmp(1)
unlink: remove directory entry. . . . . unlink(2)
chroot: change root directory for a command. . . . . chroot(1M)
getcwd: get pathname of current working directory. . . . . getcwd(3C)
mkdir: make a directory. . . . . mkdir(1)
mvdir: move a directory. . . . . mvdir(1M)
pwd: working directory name. . . . . pwd(1)
mknod: make a directory, or a special or ordinary file. . . . . mknod(2)

basename, dirname: deliver portions of pathnames. . . . . basename(1)
dis: disassembler. . . . . dis(1)
enable, disable: enable/disable LP printers. . . . . enable(1)
acct: enable or disable process accounting. . . . . acct(2)
dis: disassembler. . . . . dis(1)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
sadp: disk access profiler. . . . . sadp(1)
diskusg: generate disk accounting data by user ID. . . . . diskusg(1M)
df: report number of free disk blocks. . . . . df(1M)
driver for all disk units supported by the M68KVM21 disk controller. ud: general . . . . . ud(7)
driver for all disk units supported by the M68KVM21 disk controller. vm21: default general . . . . . vm21(7)
driver for all disk units supported by the M68KVM22 disk controller. vm22: default general . . . . . vm22(7)
vm22fmt: format disks on the VM22 disk controller. . . . . vm22fmt(1M)
sa400flwd: 5¼-inch Floppy Disk Drive for the Winchester Disk Driver. . . . . sa400flwd(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . . . . fl8(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21 Driver. . . . . sa800fl21(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22 Driver. . . . . sa400fl22(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22 Driver. . . . . sa800fl22(7)
wd15: 15Mb Winchester Disk Drive. . . . . wd15(7)
wd40: 40Mb Winchester Disk Drive. . . . . wd40(7)
cm16: 16Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm16(7)
cm80: 80Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm80(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . . . . fl8(7)
lrk25: 25Mb LARK Module Drive for Universal Disk Driver. . . . . lrk25(7)
5¼-inch Floppy Disk Drive for the Winchester Disk Driver. sa400flwd: . . . . . sa400flwd(7)
dinit: disk initializer. . . . . dinit(1M)
controller. ud: general driver for all disk units supported by the M68KVM21 disk . . . . ud(7)
controller. vm21: default general driver for all disk units supported by the M68KVM21 disk . . . . vm21(7)
controller. vm22: default general driver for all disk units supported by the M68KVM22 disk . . . . vm22(7)
du: summarize disk usage. . . . . du(1)
vm22fmt: format disks on the VM22 disk controller. . . . . vm22fmt(1M)
diskusg: generate disk accounting data by user ID. . . . diskusg(1M)
mount, umount: mount and dismount file system. . . . . mount(1M)
vi: screen-oriented (visual) display editor based on ex. . . . . vi(1)
prof: display profile data. . . . . prof(1)
hypot: Euclidean distance function. . . . . hypot(3M)
srand48, seed48, lcong48: generate uniformly distributed pseudo-random numbers. /jrand48, . . . drand48(3C)
function. log, alog, dlog, clog: FORTRAN natural logarithm intrinsic . . . log(3F)
function. log10, alog10, dlog10: FORTRAN common logarithm intrinsic . . . log10(3F)
max, max0, amax0, max1, amax1, dmax1: FORTRAN maximum-value functions. . . . max(3F)
min, min0, amin0, min1, amin1, dmin1: FORTRAN minimum-value functions. . . . min(3F)
mod, amod, dmod: FORTRAN remaindering intrinsic functions. . . mod(3F)
functions. anint, dnint, nint, idnint: FORTRAN nearest integer . . . . round(3F)
prdaily, prtacct, runacct,/ chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, ... acctsh(1M)
whodo: who is doing what. . . . . whodo(1M)
dprod: double precision product intrinsic function. . . . . dprod(3F)
strtod, atof: convert string to double-precision number. . . . . strtod(3C)
dprod: double precision product intrinsic function. . . dprod(3F)
reversi: a game of dramatic reversals. . . . . reversi(6)
jrand48, srand48, seed48, lcong48: generate/ drand48, erand48, lrand48, nrand48, mrand48, ... drand48(3C)
graph: draw a graph. . . . . graph(1G)
arithmetic: provide drill in number facts. . . . . arithmetic(6)
sa400flwd: 5¼-inch Floppy Disk Drive for the Winchester Disk Driver. . . . . sa400flwd(7)
cm16: 16Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm16(7)
cm80: 80Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm80(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . . . . fl8(7)
lrk25: 25Mb LARK Module Drive for Universal Disk Driver. . . . . lrk25(7)
cmd16: 16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd16(7)
cmd80: 80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd80(7)
lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark25(7)
lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark8(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21 Driver. . . . . sa800fl21(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22 Driver. . . . . sa400fl22(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22 Driver. . . . . sa800fl22(7)
wd15: 15Mb Winchester Disk Drive. . . . . wd15(7)
wd40: 40Mb Winchester Disk Drive. . . . . wd40(7)
cmd16: 16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd16(7)
cmd80: 80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd80(7)
lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark25(7)
lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark8(7)
16Mb Cartridge Module Drive for Universal Disk Driver. cm16: . . . . . cm16(7)
80Mb Cartridge Module Drive for Universal Disk Driver. cm80: . . . . . cm80(7)
Cartridge Module Drive for VM21 Driver and VM22 Driver. cmd16: 16Mb . . . . . cmd16(7)
Cartridge Module Drive for VM21 Driver and VM22 Driver. cmd80: 80Mb . . . . . cmd80(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . . . . fl8(7)

disk controller. ud: general	driver for all disk units supported by the M68KVM21	ud(7)
disk controller. vm21: default general	driver for all disk units supported by the M68KVM21	vm21(7)
disk controller. vm22: default general	driver for all disk units supported by the M68KVM22	vm22(7)
50Mb LARK Module Drive for VM21 Driver and VM22	Driver. lark25:	lark25(7)
16Mb LARK Module Drive for VM21 Driver and VM22	Driver. lark8:	lark8(7)
lrk25: 25Mb LARK Module Drive for Universal Disk	Driver.	lrk25(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22	Driver.	sa400fl22(7)
Floppy Disk Drive for the Winchester Disk	Driver. sa400flwd: 5¼-inch	sa400flwd(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21	Driver.	sa800fl21(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22	Driver.	sa800fl22(7)
sxt: pseudo-device	driver.	sxt(7)
sign, isign,	dsign: FORTRAN transfer-of-sign intrinsic function.	sign(3F)
sin,	dsin, csin: FORTRAN sine intrinsic function.	sin(3F)
sinh,	dsinh: FORTRAN hyperbolic sine intrinsic function.	sinh(3F)
function. sqrt,	dsqrt, csqrt: FORTRAN square root intrinsic	sqrt(3F)
tan,	dtan: FORTRAN tangent intrinsic function.	tan(3F)
function. tanh,	dtanh: FORTRAN hyperbolic tangent intrinsic	tanh(3F)
	du: summarize disk usage.	du(1)
m400: MVME400	Dual RS-232C Serial Port Module.	m400(7)
	dump: dump selected parts of an object file.	dump(1)
errdead: extract error records from	dump.	errdead(1M)
od: octal	dump.	od(1)
dump:	dump selected parts of an object file.	dump(1)
	dup: duplicate an open file descriptor.	dup(2)
dup:	duplicate an open file descriptor.	dup(2)
echo:	echo arguments.	echo(1)
	echo: echo arguments.	echo(1)
string.	ecvt, fcvt, gcvt: convert floating-point number to	ecvt(3C)
	ed, red: text editor.	ed(1)
end, etext,	edata: last locations in program.	end(3C)
	edit: text editor (variant of ex for casual users).	edit(1)
sact: print current SCCS file	editing activity.	sact(1)
vi: screen-oriented (visual) display	editor based on ex.	vi(1)
ed, red: text	editor.	ed(1)
ex: text	editor.	ex(1)
ld: link	editor for common object files.	ld(1)
ged: graphical	editor.	ged(1G)
a.out: common assembler and link	editor output.	a.out(4)
sed: stream	editor.	sed(1)
edit: text	editor (variant of ex for casual users).	edit(1)
get real user, effective user, real group, and	effective group IDs. /geteuid, getgid, getegid:	getuid(2)
getuid, geteuid, getgid, getegid: get real user,	effective user, real group, and effective group/	getuid(2)
	efl: Extended FORTRAN Language.	efl(1)
fsplit: split f77, ratfor, or	efl files.	fsplit(1)
grep,	egrep, fgrep: search a file for a pattern.	grep(1)
	enable, disable: enable/disable LP printers.	enable(1)
acct:	enable or disable process accounting.	acct(2)
enable, disable:	enable/disable LP printers.	enable(1)
crypt,	encrypt: a one way hashing encryption algorithm.	crypt(3C)
crypt, encrypt: a one way hashing	encryption algorithm.	crypt(3C)
makekey: generate	encryption key.	makekey(1)
	end, etext, edata: last locations in program.	end(3C)
getgrent, getgrgid, getgrnam, setgrent,	endgrent: obtain.	getgrent(3C)
getpwent, getpwuid, getpwnam, setpwent,	endpwent, fgetpwent: get password file entry.	getpwent(3C)
getutent, getutid, getutline, pututline, setutent,	endutent, utmpname: access utmp file entry.	getut(3C)
trenter:	enter a trouble report.	trenter(1M)
nlist: get	entries from name list.	nlist(3C)
linenum: line number	entries in a common object file.	linenum(4)
man, manprog: print	entries in this manual.	man(1)
ldlread, ldlinit, ldlitem: manipulate line number	entries of a common object file function.	ldlread(3X)
ldlseek, ldnlseek: seek to line number	entries of a section of a common object file.	ldlseek(3X)
ldrseek, ldnrseek: seek to relocation	entries of a section of a common object file.	ldrseek(3X)
utmp, wtmp: utmp and wtmp	entry formats.	utmp(4)
setpwent, endpwent, fgetpwent: get password file	entry. getpwent, getpwuid, getpwnam,	getpwent(3C)
setutent, endutent, utmpname: access utmp file	entry. getutent, getutid, getutline, pututline,	getut(3C)
ldtbindex: compute the index of a symbol table	entry of a common object file.	ldtbindex(3X)
ldtbread: read an indexed symbol table	entry of a common object file.	ldtbread(3X)
putpwent: write password file	entry.	putpwent(3C)
unlink: remove directory	entry.	unlink(2)
	env: set environment for command execution.	env(1)
	environ: user environment.	environ(5)
profile: setting up an	environment at login time.	profile(4)
environ: user	environment.	environ(5)
env: set	environment for command execution.	env(1)
getenv: return value for	environment name.	getenv(3C)

putenv: change or add value to environment. . . . putenv(3C)
getenv: return FORTRAN environment variable. . . . getenv(3F)
sky: obtain ephemerides. . . . sky(6)
srand48, seed48, lcong48: generate/ drand48, erand48, lrand48, nrand48, mrand48, jrand48, ... drand48(3C)
routines and filters. hpd, erase, hardcopy, tekset, td: graphical device . . . . gdev(1G)
function. erf, erfc: error function and complementary error . . . erf(3M)
function. erf, erfc: error function and complementary error . . . . erf(3M)
err: error-logging interface. . . . err(7)
errdead: extract error records from dump. . . . errdead(1M)
errdemon: error-logging daemon. . . . errdemon(1M)
errfile: error-log file format. . . . errfile(4)
messages. perror, errno, sys_errlist, sys_nerr: system error . . . . perror(3C)
erf, erfc: error function and complementary error function. . . erf(3M)
erf, erfc: error function and complementary error function. . . . erf(3M)
perror, errno, sys_errlist, sys_nerr: system error messages. . . . perror(3C)
intro: introduction to system calls and error numbers. . . . intro(2)
errdead: extract error records from dump. . . . errdead(1M)
matherr: error-handling function. . . . matherr(3M)
errfile: error-log file format. . . . errfile(4)
errdemon: error-logging daemon. . . . errdemon(1M)
errstop: terminate the error-logging daemon. . . . errstop(1M)
err: error-logging interface. . . . err(7)
errpt: process a report of logged errors. . . . errpt(1M)
spell, hashmake, spellin, hashcheck: find spelling errors. . . . spell(1)
errpt: process a report of logged errors. . . . errpt(1M)
errstop: terminate the error-logging daemon. . . . errstop(1M)
dial: establish an out-going terminal line connection. . . . dial(3C)
setmnt: establish mount table. . . . setmnt(1M)
end, etext, edata: last locations in program. . . . end(3C)
hypot: Euclidean distance function. . . . hypot(3M)
expr: evaluate arguments as an expression. . . . expr(1)
test: condition evaluation command. . . . test(1)
edit: text editor (variant of ex for casual users). . . . edit(1)
ex: text editor. . . . ex(1)
screen-oriented (visual) display editor based on ex. vi: . . . vi(1)
crash: examine system images. . . . crash(1M)
execute a file. execl, execv, execle, execve, execlp, execvp: . . . . exec(2)
execl, execv, execle, execve, execlp, execvp: execute a file. . . . . exec(2)
execl, execv, execle, execve, execlp, execvp: execute a file. . . . . exec(2)
execl, execv, execle, execve, execlp, execvp: execute a file. . . . . exec(2)
regcmp, regex: compile and execute a regular expression. . . . regcmp(3X)
xargs: construct argument list(s) and execute command. . . . xargs(1)
at, batch: execute commands at a later time. . . . at(1)
env: set environment for command execution. . . . env(1)
sleep: suspend execution for an interval. . . . sleep(1)
sleep: suspend execution for interval. . . . sleep(3C)
monitor: prepare execution profile. . . . monitor(3C)
profil: execution time profile. . . . profil(2)
uux: UNIX-to-UNIX system command execution. . . . uux(1C)
file. execl, execv, execle, execve, execlp, execvp: execute a . . . . exec(2)
execl, execv, execle, execve, execlp, execvp: execute a file. . . . . exec(2)
execl, execv, execle, execve, execlp, execvp: execute a file. . . . . exec(2)
link, unlink: exercise link and unlink system calls. . . . link(1M)
creat: create a new file or rewrite an existing one. . . . creat(2)
exit, _exit: terminate process. . . . exit(2)
exit, _exit: terminate process. . . . exit(2)
bootstrap operating procedure for system restart on EXORmacs. bo.macs: . . . bo.macs(8)
ops.macs: EXORmacs operations. . . . ops.macs(8)
function. exp, dexp, cexp: FORTRAN exponential intrinsic . . . exp(3F)
power, square root functions. exp, log, log10, pow, sqrt: exponential, logarithm, . . . exp(3M)
pack, pcat, unpack: compress and expand files. . . . pack(1)
float, sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type conversion. /idint, real, . . . ftype(3F)
exp, dexp, cexp: FORTRAN exponential intrinsic function. . . . exp(3F)
functions. exp, log, log10, pow, sqrt: exponential, logarithm, power, square root . . . . exp(3M)
expr: evaluate arguments as an expression. . . . expr(1)
regexp: regular expression compile and match routines. . . . regexp(5)
regcmp: regular expression compile. . . . regcmp(1)
expr: evaluate arguments as an expression. . . . expr(1)
regcmp, regex: compile and execute a regular expression. . . . regcmp(3X)
efl: Extended FORTRAN Language. . . . efl(1)
errdead: extract error records from dump. . . . errdead(1M)
f77: FORTRAN 77 compiler. . . . f77(1)
fsplit: split f77, ratfor, or efl files. . . . fsplit(1)
functions. floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value . . . . floor(3M)
factor: factor a number. . . . factor(1)

	factor: factor a number.	factor(1)
true,	false: provide truth values.	true(1)
access long integer data in a machine independent	fashion.. sputl, sgetl:	sputl(3X)
finc:	fast incremental backup.	finc(1M)
malloc, free, realloc, calloc, mallopt, mallinfo:	fast main memory allocator.	malloc(3X)
checkall:	faster file system checking procedure.	checkall(1M)
abort: generate an IOT	fault.	abort(3C)
	fclose, fflush: close or flush a stream.	fclose(3S)
	fcntl: file control.	fcntl(2)
	fcntl: file control options.	fcntl(5)
string. ecvt,	fcvt, gcvt: convert floating-point number to	ecvt(3C)
fopen, freopen,	fdopen: open a stream.	fopen(3S)
col: filter reverse line	feeds.	col(1)
ferror,	feof, clearerr, fileno: stream status inquiries.	ferror(3S)
inquiries.	ferror, feof, clearerr, fileno: stream status	ferror(3S)
system.	ff: list filenames and statistics for a file	ff(1M)
fclose,	fflush: close or flush a stream.	fclose(3S)
getc, getchar,	fgetc, getw: get character or word from stream.	getc(3S)
getpwent, getpwuid, getpwnam, setpwent, endpwent,	fgetpwent: get password file entry.	getpwent(3C)
gets,	fgets: get a string from a stream.	gets(3S)
grep, egrep,	fgrep: search a file for a pattern.	grep(1)
utime: set	file access and modification times.	utime(2)
ldfcn: common object	file access routines.	ldfcn(4)
access: determine accessibility of a	file.	access(2)
tar: tape	file archiver.	tar(1)
cpio: copy	file archives in and out.	cpio(1)
pwck, grpck: password/group	file checkers.	pwck(1M)
chmod: change mode of	file.	chmod(2)
chown: change owner and group of a	file.	chown(2)
diff: differential	file comparator.	diff(1)
bdiff:	file comparator for large files.	bdiff(1)
diff3: 3-way differential	file comparison.	diff3(1)
fcntl:	file control.	fcntl(2)
fcntl:	file control options.	fcntl(5)
conv: object	file converter.	conv(1)
uuto, uupick: public UNIX System-to-UNIX System	file copy.	uuto(1C)
core: format of core image	file.	core(4)
umask: set and get	file creation mask.	umask(2)
crontab: user crontab	file.	crontab(1)
cut: cut out selected fields of each line of a	file.	cut(1)
dd: convert and copy a	file.	dd(1)
delta: make a delta (change) to an SCCS	file.	delta(1)
close: close a	file descriptor.	close(2)
dup: duplicate an open	file descriptor.	dup(2)
	file: determine file type.	file(1)
dfile: device information	file.	dfile(4)
dump: dump selected parts of an object	file.	dump(1)
sact: print current SCCS	file editing activity.	sact(1)
setpwent, endpwent, fgetpwent: get password	file entry. getpwent, getpwuid, getpwnam,	getpwent(3C)
setutent, endutent, utmpname: access utmp	file entry. /getutid, getutline, pututline,	getut(3C)
putpwent: write password	file entry.	putpwent(3C)
execv, execle, execve, execlp, execvp: execute a	file. execl,	exec(2)
grep, egrep, fgrep: search a	file for a pattern.	grep(1)
ldopen, ldaopen: open a common object	file for reading.	ldopen(3X)
acct: per-process accounting	file format.	acct(4)
ar: common archive	file format.	ar(4)
errfile: error-log	file format.	errfile(4)
pnch:	file format for card images.	pnch(4)
intro: introduction to	file formats.	intro(4)
manipulate line number entries of a common object	file function. ldlread, ldlinit, ldlitem:	ldlread(3X)
get: get a version of an SCCS	file.	get(1)
group: group	file.	group(4)
filehdr:	file header for common object files.	filehdr(4)
ldfhread: read the	file header of a common object file.	ldfhread(3X)
ldohseek: seek to the optional	file header of a common object file.	ldohseek(3X)
split: split a	file into pieces.	split(1)
issue: issue identification	file.	issue(4)
read the archive header of a member of an archive	file. ldahread:	ldahread(3X)
ldclose, ldaclose: close a common object	file.	ldclose(3X)
ldfhread: read the file header of a common object	file.	ldfhread(3X)
ldgetname: retrieve symbol name for object	file.	ldgetname(3X)
line number entries of a section of a common object	file. ldlseek, ldnlseek: seek to	ldlseek(3X)
seek to the optional file header of a common object	file. ldohseek:	ldohseek(3X)
relocation entries of a section of a common object	file. ldrseek, ldnrseek: seek to	ldrseek(3X)
an indexed/named section header of a common object	file. ldshread, ldnshread: read	ldshread(3X)

to an indexed/named section of a common object file. ldsseek, ldnsseek: seek . . . . . ldsseek(3X)
index of a symbol table entry of a common object file. ldtbindex: compute the . . . . . ldtbindex(3X)
an indexed symbol table entry of a common object file. ldtbread: read . . . . . ldtbread(3X)
seek to the symbol table of a common object file. ldtbseek: . . . . . ldtbseek(3X)
linenum: line number entries in a common object file. . . . . linenum(4)
link: link to a file. . . . . link(2)
mknod: build special file. . . . . mknod(1M)
mknod: make a directory, or a special or ordinary file. . . . . mknod(2)
newform: change the format of a text file. . . . . newform(1)
nm: print name list of common object file. . . . . nm(1)
null: the null file. . . . . null(7)
ttyslot: find the slot in the utmp file of the current user. . . . . ttyslot(3C)
fuser: identify processes using a file or file structure. . . . . fuser(1M)
creat: create a new file or rewrite an existing one. . . . . creat(2)
passwd: password file. . . . . passwd(4)
lines of several files or subsequent lines of one file. paste: merge same . . . . . paste(1)
pg: file perusal filter for soft-copy terminals. . . . . pg(1)
fseek, rewind, ftell: reposition a file pointer in a stream. . . . . fseek(3S)
lseek: move read/write file pointer. . . . . lseek(2)
prs: print an SCCS file. . . . . prs(1)
read: read from file. . . . . read(2)
reloc: relocation information for a common object file. . . . . reloc(4)
rmdel: remove a delta from an SCCS file. . . . . rmdel(1)
bfs: big file scanner. . . . . bfs(1)
sccsdiff: compare two versions of an SCCS file. . . . . sccsdiff(1)
sccsfile: format of SCCS file. . . . . sccsfile(4)
scnhdr: section header for a common object file. . . . . scnhdr(4)
stat, fstat: get file status. . . . . stat(2)
symbol and line number information from an object file. strip: strip . . . . . strip(1)
fuser: identify processes using a file or file structure. . . . . fuser(1M)
sum: print checksum and block count of a file. . . . . sum(1)
syms: common object file symbol table format. . . . . syms(4)
filesave, tapesave: daily/weekly SYSTEM V/68 file system backup. . . . . filesave(1M)
fsba: file system block analyzer. . . . . fsba(1M)
checkall: faster file system checking procedure. . . . . checkall(1M)
repair. fsck, dfsck: file system consistency check and interactive . . . . . fsck(1M)
fsdb: file system debugger. . . . . fsdb(1M)
ff: list filenames and statistics for a file system. . . . . ff(1M)
file system: format of system volume. . . . . fs(4)
mkfs: construct a file system. . . . . mkfs(1M)
mount, umount: mount and dismount file system. . . . . mount(1M)
mount: mount a file system. . . . . mount(2)
ustat: get file system statistics. . . . . ustat(2)
mnttab: mounted file system table. . . . . mnttab(4)
umount: unmount a file system. . . . . umount(2)
dcopy: copy file systems for optimal access time. . . . . dcopy(1M)
checklist: list of file systems processed by fsck. . . . . checklist(4)
volcopy, labelit: copy file systems with label checking. . . . . volcopy(1M)
tail: deliver the last part of a file. . . . . tail(1)
term: format of compiled term file. . . . . term(4)
tmpfile: create a temporary file. . . . . tmpfile(3S)
tmpnam, tempnam: create a name for a temporary file. . . . . tmpnam(3S)
touch: update access and modification times of a file. . . . . touch(1)
ftw: walk a file tree. . . . . ftw(3C)
file: determine file type. . . . . file(1)
unget: undo a previous get of an SCCS file. . . . . unget(1)
uniq: report repeated lines in a file. . . . . uniq(1)
val: validate SCCS file. . . . . val(1)
write: write on a file. . . . . write(2)
umask: set file-creation mode mask. . . . . umask(1)
filehdr: file header for common object files. . . . . filehdr(4)
ctermid: generate filename for terminal. . . . . ctermid(3S)
mktemp: make a unique filename. . . . . mktemp(3C)
ff: list filenames and statistics for a file system. . . . . ff(1M)
ferror, feof, clearerr, fileno: stream status inquiries. . . . . ferror(3S)
acctcom: search and print process accounting file(s). . . . . acctcom(1)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
admin: create and administer SCCS files. . . . . admin(1)
bdiff: file comparator for large files. . . . . bdiff(1)
fscv: convert files between M68000 and VAX-11/780 processors. . . fscv(1M)
cat: concatenate and print files. . . . . cat(1)
cmp: compare two files. . . . . cmp(1)
comm: select or reject lines common to two sorted files. . . . . comm(1)
cp, ln, mv: copy, link or move files. . . . . cp(1)
diffmk: mark differences between files. . . . . diffmk(1)

filehdr: file header for common object files. . . . filehdr(4)
find: find files. . . . find(1)
frec: recover files from a backup tape. . . . frec(1M)
fspec: format specification in text files. . . . fspec(4)
fsplit: split f77, ratfor, or efl files. . . . fsplit(1)
graphical primitive string, format of graphical files. gps: . . . gps(4)
cpset: install object files in binary directories. . . . cpset(1M)
intro: introduction to special files. . . . intro(7)
ld: link editor for common object files. . . . ld(1)
rm, rmdir: remove files or directories. . . . rm(1)
paste: merge same lines of several files or subsequent lines of one file. . . . paste(1)
pack, pcat, unpack: compress and expand files. . . . pack(1)
pr: print files. . . . pr(1)
size: print section sizes of common object files. . . . size(1)
sort: sort and/or merge files. . . . sort(1)
what: identify SCCS files. . . . what(1)
backup. filesave, tapesave: daily/weekly 5) file system . . . filesave(1M)
pg: file perusal filter for soft-copy terminals. . . . pg(1)
greek: select terminal filter. . . . greek(1)
nl: line numbering filter. . . . nl(1)
col: filter reverse line feeds. . . . col(1)
hardcopy, tekset, td: graphical device routines and filters. hpd, erase, . . . gdev(1G)
tplot: graphics filters. . . . tplot(1G)
finc: fast incremental backup. . . . finc(1M)
find: find files. . . . find(1)
find: find files. . . . find(1)
hyphen: find hyphenated words. . . . hyphen(1)
ttyname, isatty: find name of a terminal. . . . ttyname(3C)
lorder: find ordering relation for an object library. . . . lorder(1)
spell, hashmake, spellin, hashcheck: find spelling errors. . . . spell(1)
ttyslot: find the slot in the utmp file of the current user. . . . ttyslot(3C)
tee: pipe fitting. . . . tee(1)
Driver. fl8: 8-inch Floppy Disk Drive for Universal Disk . . . fl8(7)
explicit FORTRAN type/ int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char: . . . . ftype(3F)
atof: convert ASCII string to floating-point number. . . . atof(3C)
ecvt, fcvt, gcvt: convert floating-point number to string. . . . ecvt(3C)
frexp, ldexp, modf: manipulate parts of floating-point numbers. . . . frexp(3C)
absolute value functions. floor, ceil, fmod, fabs: floor, ceiling, remainder, . . . floor(3M)
functions. floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value . . . . floor(3M)
wffmt: format floppies for the VME/10 processor. . . . wffmt(1M)
sa400flwd: 5¼-inch Floppy Disk Drive for the Winchester Disk Driver. . . sa400flwd(7)
fl8: 8-inch Floppy Disk Drive for Universal Disk Driver. . . . fl8(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21 Driver. . . . sa800fl21(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22 Driver. . . . sa400fl22(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22 Driver. . . . sa800fl22(7)
cflow: generate C flow graph. . . . cflow(1)
fclose, fflush: close or flush a stream. . . . fclose(3S)
value functions. floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute . . . . floor(3M)
fopen, freopen, fdopen: open a stream. . . . fopen(3S)
fork: create a new process. . . . fork(2)
acct: per-process accounting file format. . . . acct(4)
ar: common archive file format. . . . ar(4)
vm22fmt: format disks on the VM22 disk controller. . . . vm22fmt(1M)
errfile: error-log file format. . . . errfile(4)
wffmt: format floppies for the VME/10 processor. . . . wffmt(1M)
pnch: file format for card images. . . . pnch(4)
newform: change the format of a text file. . . . newform(1)
inode: format of an inode. . . . inode(4)
term: format of compiled term file. . . . term(4)
core: format of core image file. . . . core(4)
cpio: format of cpio archive. . . . cpio(4)
dir: format of directories. . . . dir(4)
gps: graphical primitive string, format of graphical files. . . . gps(4)
sccsfile: format of SCCS file. . . . sccsfile(4)
file system: format of system volume. . . . fs(4)
fspec: format specification in text files. . . . fspec(4)
syms: common object file symbol table format. . . . syms(4)
intro: introduction to file formats. . . . intro(4)
utmp, wtmp: utmp and wtmp entry formats. . . . utmp(4)
scanf, fscanf, sscanf: convert formatted input. . . . scanf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs argument list. . . . . vprintf(3S)
printf, fprintf, sprintf: print formatted output. . . . printf(3S)
f77: FORTRAN 77 compiler. . . . f77(1)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . abs(3F)
signal: specify FORTRAN action on receipt of a system signal. . . . signal(3F)

acos, dacos: FORTRAN arccosine intrinsic function. . . . . acos(3F)
asin, dasin: FORTRAN arcsine intrinsic function. . . . . asin(3F)
atan2, datan2: FORTRAN arctangent intrinsic function. . . . . atan2(3F)
atan, datan: FORTRAN arctangent intrinsic function. . . . . atan(3F)
and, or, xor, not, lshift, rshift: FORTRAN bitwise Boolean functions. . . . . bool(3F)
getarg: return FORTRAN command-line argument. . . . . getarg(3F)
log10, alog10, dlog10: FORTRAN common logarithm intrinsic function. . . . log10(3F)
conjg, dconjg: FORTRAN complex conjugate intrinsic function. . . . conjg(3F)
cos, dcos, ccos: FORTRAN cosine intrinsic function. . . . . cos(3F)
ratfor: rational FORTRAN dialect. . . . . ratfor(1)
getenv: return FORTRAN environment variable. . . . . getenv(3F)
exp, dexp, cexp: FORTRAN exponential intrinsic function. . . . . exp(3F)
cosh, dcosh: FORTRAN hyperbolic cosine intrinsic function. . . . . cosh(3F)
sinh, dsinh: FORTRAN hyperbolic sine intrinsic function. . . . . sinh(3F)
tanh, dtanh: FORTRAN hyperbolic tangent intrinsic function. . . . tanh(3F)
aimag, dimag: FORTRAN imaginary part of complex argument. . . . aimag(3F)
aint, dint: FORTRAN integer part intrinsic function. . . . . aint(3F)
efl: Extended FORTRAN Language. . . . . efl(1)
max, max0, amax0, max1, amax1, dmax1: FORTRAN maximum-value functions. . . . . max(3F)
min, min0, amin0, min1, amin1, dmin1: FORTRAN minimum-value functions. . . . . min(3F)
log, alog, dlog, clog: FORTRAN natural logarithm intrinsic function. . . . log(3F)
anint, dnint, nint, idnint: FORTRAN nearest integer functions. . . . . round(3F)
abort: terminate FORTRAN program. . . . . abort(3F)
mod, amod, dmod: FORTRAN remaindering intrinsic functions. . . . . mod(3F)
sin, dsin, csin: FORTRAN sine intrinsic function. . . . . sin(3F)
sqrt, dsqrt, csqrt: FORTRAN square root intrinsic function. . . . . sqrt(3F)
len: return length of FORTRAN string. . . . . len(3F)
index: return location of FORTRAN substring. . . . . index(3F)
system: issue a shell command from FORTRAN. . . . . system(3F)
tan, dtan: FORTRAN tangent intrinsic function. . . . . tan(3F)
mclock: return FORTRAN time accounting. . . . . mclock(3F)
sign, isign, dsign: FORTRAN transfer-of-sign intrinsic function. . . . sign(3F)
sngl, dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type conversion. /ifix, idint, real, float, . . ftype(3F)
printf, fprintf, sprintf: print formatted output. . . . . printf(3S)
putc, putchar, fputc, putw: put character or word on a stream. . . . putc(3S)
puts, fputs: put a string on a stream. . . . . puts(3S)
fread, fwrite: binary input/output. . . . . fread(3S)
frec: recover files from a backup tape. . . . . frec(1M)
df: report number of free disk blocks. . . . . df(1M)
malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)
memory allocator. malloc, free, realloc, calloc, mallopt, mallinfo: fast main . . . malloc(3X)
fopen, freopen, fdopen: open a stream. . . . . fopen(3S)
floating-point numbers. frexp, ldexp, modf: manipulate parts of . . . . . frexp(3C)
frec: recover files from a backup tape. . . . . frec(1M)
gets, fgets: get a string from a stream. . . . . gets(3S)
strip: strip symbol and line number information from an object file. . . . . strip(1)
rmdel: remove a delta from an SCCS file. . . . . rmdel(1)
getopt: get option letter from argument vector. . . . . getopt(3C)
errdead: extract error records from dump. . . . . errdead(1M)
read: read from file. . . . . read(2)
system: issue a shell command from FORTRAN. . . . . system(3F)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
nlist: get entries from name list. . . . . nlist(3C)
acctcms: command summary from per-process accounting records. . . . . acctcms(1M)
getc, getchar, fgetc, getw: get character or word from stream. . . . . getc(3S)
getpw: get name from UID. . . . . getpw(3C)
fsba: file system block analyzer. . . . . fsba(1M)
scanf, fscanf, sscanf: convert formatted input. . . . . scanf(3S)
checklist: list of file systems processed by fsck. . . . . checklist(4)
interactive repair. fsck, dfsck: file system consistency check and . . . . . fsck(1M)
processors. fscv: convert files between M68000 and VAX-11/780 . fscv(1M)
fsdb: file system debugger. . . . . fsdb(1M)
a stream. fseek, rewind, ftell: reposition a file pointer in . . . . fseek(3S)
fspec: format specification in text files. . . . . fspec(4)
fsplit: split f77, ratfor, or efl files. . . . . fsplit(1)
stat, fstat: get file status. . . . . stat(2)
fseek, rewind, ftell: reposition a file pointer in a stream. . . . . fseek(3S)
ftw: walk a file tree. . . . . ftw(3C)
acos, dacos: FORTRAN arccosine intrinsic function. . . . . acos(3F)
aint, dint: FORTRAN integer part intrinsic function. . . . . aint(3F)
erf, erfc: error function and complementary error function. . . . . erf(3M)
asin, dasin: FORTRAN arcsine intrinsic function. . . . . asin(3F)
atan2, datan2: FORTRAN arctangent intrinsic function. . . . . atan2(3F)
atan, datan: FORTRAN arctangent intrinsic function. . . . . atan(3F)
conjg, dconjg: FORTRAN complex conjugate intrinsic function. . . . . conjg(3F)

cos, dcos, ccos: FORTRAN cosine intrinsic function. . . . . cos(3F)
cosh, dcosh: FORTRAN hyperbolic cosine intrinsic function. . . . . cosh(3F)
dprod: double precision product intrinsic function. . . . . dprod(3F)
erf, erfc: error function and complementary error function. . . . . erf(3M)
exp, dexp, cexp: FORTRAN exponential intrinsic function. . . . . exp(3F)
gamma: log gamma function. . . . . gamma(3M)
hypot: Euclidean distance function. . . . . hypot(3M)
line number entries of a common object file function. ldlread, ldlinit, ldlitem: manipulate . . . ldlread(3X)
alog10, dlog10: FORTRAN common logarithm intrinsic function. log10, ... log10(3F)
dlog, clog: FORTRAN natural logarithm intrinsic function. log, alog, ... log(3F)
matherr: error-handling function. . . . . matherr(3M)
prof: profile within a function. . . . . prof(5)
isign, dsign: FORTRAN transfer-of-sign intrinsic function. sign, ... sign(3F)
sin, dsin, csin: FORTRAN sine intrinsic function. . . . . sin(3F)
sinh, dsinh: FORTRAN hyperbolic sine intrinsic function. . . . . sinh(3F)
sqrt, dsqrt, csqrt: FORTRAN square root intrinsic function. . . . . sqrt(3F)
tan, dtan: FORTRAN tangent intrinsic function. . . . . tan(3F)
tanh, dtanh: FORTRAN hyperbolic tangent intrinsic function. . . . . tanh(3F)
math: math functions and constants. . . . . math(5)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
xor, not, lshift, rshift: FORTRAN bitwise Boolean functions. and, or, ... bool(3F)
dim, ddim, idim: positive difference intrinsic functions. . . . . dim(3F)
sqrt: exponential, logarithm, power, square root functions. exp, log, log10, pow, ... exp(3M)
fabs: floor, ceiling, remainder, absolute value functions. floor, ceil, fmod, ... floor(3M)
amax0, max1, amax1, dmax1: FORTRAN maximum-value functions. max, max0, ... max(3F)
amin0, min1, amin1, dmin1: FORTRAN minimum-value functions. min, min0, ... min(3F)
mod, amod, dmod: FORTRAN remaindering intrinsic functions. . . . . mod(3F)
300, 300s: handle special functions of DASI 300 and 300s terminals. . . . . 300(1)
hp: handle special functions of HP 2640 and 2621-series terminals. . . . hp(1)
450: handle special functions of the DASI 450 terminal. . . . . 450(1)
anint, dnint, nint, idnint: FORTRAN nearest integer functions. . . . . round(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)
lge, lgt, lle, llt: string comparision intrinsic functions. . . . . strcmp(3F)
cos, tan, asin, acos, atan, atan2: trigonometric functions. sin, ... trig(3M)
structure. fuser: identify processes using a file or file . . . . . fuser(1M)
fread, fwrite: binary input/output. . . . . fread(3S)
records. fwtmp, wtmpfix: manipulate connect accounting . . . fwtmp(1M)
jotto: secret word game. . . . . jotto(6)
moo: guessing game. . . . . moo(6)
back: the game of backgammon. . . . . back(6)
bj: the game of blackjack. . . . . bj(6)
chess: the game of chess. . . . . chess(6)
craps: the game of craps. . . . . craps(6)
reversi: a game of dramatic reversals. . . . . reversi(6)
wump: the game of hunt-the-wumpus. . . . . wump(6)
intro: introduction to games. . . . . intro(6)
gamma: log gamma function. . . . . gamma(3M)
gamma: log gamma function. . . . . gamma(3M)
ecvt, fcvt, gcvt: convert floating-point number to string. . . . . ecvt(3C)
ged: graphical editor. . . . . ged(1G)
maze: generate a maze. . . . . maze(6)
abort: generate an IOT fault. . . . . abort(3C)
cflow: generate C flow graph. . . . . cflow(1)
cxref: generate C program cross-reference. . . . . cxref(1)
diskusg: generate disk accounting data by user ID. . . . . diskusg(1M)
makekey: generate encryption key. . . . . makekey(1)
ctermid: generate filename for terminal. . . . . ctermid(3S)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
lex: generate programs for simple lexical tasks. . . . . lex(1)
/mrand48, jrand48, srand48, seed48, lcong48: generate uniformly distributed pseudo-random/ . . . drand48(3C)
rand, srand: simple random-number generator. . . . . rand(3C)
irand, srand, rand: random number generator. . . . . rand(3F)
gets, fgets: get a string from a stream. . . . . gets(3S)
get: get a version of an SCCS file. . . . . get(1)
ulimit: get and set user limits. . . . . ulimit(2)
cuserid: get character login name of the user. . . . . cuserid(3S)
getc, getchar, fgetc, getw: get character or word from stream. . . . . getc(3S)
nlist: get entries from name list. . . . . nlist(3C)
umask: set and get file creation mask. . . . . umask(2)
stat, fstat: get file status. . . . . stat(2)
ustat: get file system statistics. . . . . ustat(2)
get: get a version of an SCCS file. . . . . get(1)
getlogin: get login name. . . . . getlogin(3C)
logname: get login name. . . . . logname(1)
msgget: get message queue. . . . . msgget(2)

getpw: get name from UID. . . . . getpw(3C)
uname: get name of current operating system. . . . . uname(2)
unget: undo a previous get of an SCCS file. . . . . unget(1)
getopt: get option letter from argument vector. . . . . getopt(3C)
getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get password file entry. getpwent, ... getpwent(3C)
getcwd: get pathname of current working directory. . . . . getcwd(3C)
times: get process and child process times. . . . . times(2)
getpid, getpgrp, getppid: get process, process group, and parent process IDs. . . . getpid(2)
effective group/ getuid, geteuid, getgid, getegid: get real user, effective user, real group, and . . . . . getuid(2)
semget: get set of semaphores. . . . . semget(2)
shmget: get shared memory segment. . . . . shmget(2)
tty: get the terminal's name. . . . . tty(1)
time: get time. . . . . time(2)
getarg: return FORTRAN command-line argument. . . getarg(3F)
from stream. getc, getchar, fgetc, getw: get character or word . . . . getc(3S)
stream. getc, getchar, fgetc, getw: get character or word from . . . getc(3S)
getcwd: get pathname of current working directory. . getcwd(3C)
and effective group IDs. getuid, geteuid, getgid, getegid: get real user, effective user, real group, ... getuid(2)
getenv: return FORTRAN environment variable. . . . getenv(3F)
getenv: return value for environment name. . . . . getenv(3C)
user, real group, and effective group IDs. getuid, geteuid, getgid, getegid: get real user, effective . . . . getuid(2)
real group, and effective group/ getuid, geteuid, getgid, getegid: get real user, effective user, ... getuid(2)
obtain. getgrent, getgrgid, getgrnam, setgrent, endgrent: . . . . getgrent(3C)
getgrent, getgrgid, getgrnam, setgrent, endgrent: obtain. . . . getgrent(3C)
getgrent, getgrgid, getgrnam, setgrent, endgrent: obtain. . . . . getgrent(3C)
getlogin: get login name. . . . . getlogin(3C)
getopt: get option letter from argument vector. . . . getopt(3C)
getopt: parse command options. . . . . getopt(1)
getpass: read a password. . . . . getpass(3C)
parent process IDs. getpid, getpgrp, getppid: get process, process group, and . . . . getpid(2)
group, and parent process IDs. getpid, getpgrp, getppid: get process, process . . . . . getpid(2)
process IDs. getpid, getpgrp, getppid: get process, process group, and parent . . . getpid(2)
getpw: get name from UID. . . . . getpw(3C)
fgetpwent: get password file entry. getpwent, getpwuid, getpwnam, setpwent, endpwent, . getpwent(3C)
password file entry. getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get . . . getpwent(3C)
get password file entry. getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: getpwent(3C)
gets, fgets: get a string from a stream. . . . . gets(3S)
gettydefs: speed and terminal settings used by getty. . . . . gettydefs(4)
discipline. getty: set terminal type, modes, speed, and line . . . . getty(1M)
ct: spawn getty to a remote terminal. . . . . ct(1C)
getty. gettydefs: speed and terminal settings used by . . . . gettydefs(4)
effective user, real group, and effective group/ getuid, geteuid, getgid, getegid: get real user, ... getuid(2)
endutent, utmpname: access utmp file entry. getutent, getutid, getutline, pututline, setutent, ... getut(3C)
utmpname: access utmp file entry. getutent, getutid, getutline, pututline, setutent, endutent, . . . getut(3C)
access utmp file entry. getutent, getutid, getutline, pututline, setutent, endutent, utmpname: . . getut(3C)
getc, getchar, fgetc, getw: get character or word from stream. . . . . getc(3S)
string. ctime, localtime, gmtime, asctime, tzset: convert date and time to . . . ctime(3C)
setjmp, longjmp: non-local goto. . . . . setjmp(3C)
graphical files. gps: graphical primitive string, format of . . . . . gps(4)
cflow: generate C flow graph. . . . . cflow(1)
graph: draw a graph. . . . . graph(1G)
graph: draw a graph. . . . . graph(1G)
sag: system activity graph. . . . . sag(1G)
graphics: access graphical and numerical commands. . . . . graphics(1G)
stat: statistical network useful with graphical commands. . . . . stat(1G)
hpd, erase, hardcopy, tekset, td: graphical device routines and filters. . . . . gdev(1G)
ged: graphical editor. . . . . ged(1G)
gps: graphical primitive string, format of graphical files. . . . . gps(4)
files. gps: graphical primitive string, format of graphical . . . . gps(4)
toc: graphical table of contents routines. . . . . toc(1G)
gutil: graphical utilities. . . . . gutil(1G)
graphics: access graphical and numerical commands. . . graphics(1G)
tplot: graphics filters. . . . . tplot(1G)
plot: graphics interface. . . . . plot(4)
plot: graphics interface subroutines. . . . . plot(3X)
v10graph - VME/10 graphics subsystem interface. . . . . v10graph(7)
greek: select terminal filter. . . . . greek(1)
grep, egrep, fgrep: search a file for a pattern. . . . . grep(1)
getegid: get real user, effective user, real group, and effective group IDs. /geteuid, getgid, . . . getuid(2)
getpid, getpgrp, getppid: get process, process group, and parent process IDs. . . . . getpid(2)
chown, chgrp: change owner or group. . . . . chown(1)
group: group file. . . . . group(4)
group: group file. . . . . group(4)
setpgrp: set process group ID. . . . . setpgrp(2)
id: print user and group IDs and names. . . . . id(1)

user, effective user, real group, and effective — group IDs. /geteuid, getgid, getegid: get real . . . . . getuid(2)
setuid, setgid: set user and — group IDs. . . . . setuid(2)
newgrp: log in to a new — group. . . . . newgrp(1)
chown: change owner and — group of a file. . . . . chown(2)
kill: send a signal to a process or a — group of processes. . . . . kill(2)
make: maintain, update, and regenerate — groups of programs. . . . . make(1)
pwck, — grpck: password/group file checkers. . . . . pwck(1M)
ssignal, — gsignal: software signals. . . . . ssignal(3C)
hangman: — guess the word. . . . . hangman(6)
moo: — guessing game. . . . . moo(6)
— gutil: graphical utilities. . . . . gutil(1G)
terminals. 300, 300s: — handle special functions of DASI 300 and 300s . . . . 300(1)
terminals. hp: — handle special functions of HP 2640 and 2621-series . hp(1)
450: — handle special functions of the DASI 450 terminal. . . 450(1)
varargs: — handle variable argument list. . . . . varargs(5)
curses: CRT screen — handling and optimization package. . . . . curses(3X)
— hangman: guess the word. . . . . hangman(6)
nohup: run a command immune to — hangups and quits. . . . . nohup(1)
filters. hpd, erase, — hardcopy, tekset, td: graphical device routines and . . gdev(1G)
hsearch, hcreate, hdestroy: manage — hash search tables. . . . . hsearch(3C)
spell, hashmake, spellin, — hashcheck: find spelling errors. . . . . spell(1)
crypt, encrypt: a one way — hashing encryption algorithm. . . . . crypt(3C)
spell, — hashmake, spellin, hashcheck: find spelling errors. . . . spell(1)
hsearch, — hcreate, hdestroy: manage hash search tables. . . . . hsearch(3C)
hsearch, hcreate, — hdestroy: manage hash search tables. . . . . hsearch(3C)
aouthdr: optional aout — header. . . . . aouthdr(4)
scnhdr: section — header for a common object file. . . . . scnhdr(4)
filehdr: file — header for common object files. . . . . filehdr(4)
ldfhread: read the file — header of a common object file. . . . . ldfhread(3X)
ldohseek: seek to the optional file — header of a common object file. . . . . ldohseek(3X)
ldshread, ldnshread: read an indexed/named section — header of a common object file. . . . . ldshread(3X)
ldahread: read the archive — header of a member of an archive file. . . . . ldahread(3X)
— help: ask for help. . . . . help(1)
help: ask for — help. . . . . help(1)
hp: handle special functions of — HP 2640 and 2621-series terminals. . . . . hp(1)
2621-series terminals. — hp: handle special functions of HP 2640 and . . . . hp(1)
routines and filters. — hpd, erase, hardcopy, tekset, td: graphical device . . . gdev(1G)
tables. — hsearch, hcreate, hdestroy: manage hash search . . . hsearch(3C)
wump: the game of — hunt-the-wumpus. . . . . wump(6)
cosh, dcosh: FORTRAN — hyperbolic cosine intrinsic function. . . . . cosh(3F)
sinh, cosh, tanh: — hyperbolic functions. . . . . sinh(3M)
sinh, dsinh: FORTRAN — hyperbolic sine intrinsic function. . . . . sinh(3F)
tanh, dtanh: FORTRAN — hyperbolic tangent intrinsic function. . . . . tanh(3F)
— hyphen: find hyphenated words. . . . . hyphen(1)
hyphen: find — hyphenated words. . . . . hyphen(1)
— hypot: Euclidean distance function. . . . . hypot(3M)
abs, — iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . abs(3F)
— iargc. . . . . iargc(3F)
/idint, real, float, sngl, dble, cmplx, dcmplx, — ichar, char: explicit FORTRAN type conversion. . . . ftype(3F)
diskusg: generate disk accounting data by user — ID. . . . . diskusg(1M)
a message queue, semaphore set or shared memory — id. ipcrm: remove . . . . . ipcrm(1)
— id: print user and group IDs and names. . . . . id(1)
setpgrp: set process group — ID. . . . . setpgrp(2)
issue: issue — identification file. . . . . issue(4)
fuser: — identify processes using a file or file structure. . . . fuser(1M)
what: — identify SCCS files. . . . . what(1)
dim, ddim, — idim: positive difference intrinsic functions. . . . . dim(3F)
ichar, char: explicit FORTRAN type/ int, ifix, — idint, real, float, sngl, dble, cmplx, dcmplx, ... ftype(3F)
anint, dnint, nint, — idnint: FORTRAN nearest integer functions. . . . . round(3F)
id: print user and group — IDs and names. . . . . id(1)
get process, process group, and parent process — IDs. getpid, getpgrp, getppid: . . . . . getpid(2)
effective user, real group, and effective group — IDs. /geteuid, getgid, getegid: get real user, ... getuid(2)
setuid, setgid: set user and group — IDs. . . . . setuid(2)
dcmplx, ichar, char: explicit FORTRAN type/ int, — ifix, idint, real, float, sngl, dble, cmplx, ... ftype(3F)
core: format of core — image file. . . . . core(4)
crash: examine system — images. . . . . crash(1M)
pnch: file format for card — images. . . . . pnch(4)
aimag, dimag: FORTRAN — imaginary part of complex argument. . . . . aimag(3F)
nohup: run a command — immune to hangups and quits. . . . . nohup(1)
finc: fast — incremental backup. . . . . finc(1M)
sputl, sgetl: access long integer data in a machine — independent fashion.. . . . . sputl(3X)
file. ldtbindex: compute the — index of a symbol table entry of a common object . . . ldtbindex(3X)
ptx: permuted — index. . . . . ptx(1)
— index: return location of FORTRAN substring. . . . . index(3F)
ldtbread: read an — indexed symbol table entry of a common object file. . . ldtbread(3X)

file. ldshread, ldnshread: read an — indexed/named section header of a common object . . ldshread(3X)
ldsseek, ldnsseek: seek to an — indexed/named section of a common object file. . . . . ldsseek(3X)
inittab: script for the — init process. . . . . inittab(4)
— init, telinit: process control initialization. . . . . init(1M)
init, telinit: process control — initialization. . . . . init(1M)
brc, bcheckrc, rc, powerfail: system — initialization shell scripts. . . . . brc(1M)
dinit: disk — initializer. . . . . dinit(1M)
popen, pclose: — initiate pipe to/from a process. . . . . popen(3S)
— inittab: script for the init process. . . . . inittab(4)
clri: clear — inode. . . . . clri(1M)
— inode: format of an inode. . . . . inode(4)
inode: format of an — inode. . . . . inode(4)
scanf, fscanf, sscanf: convert formatted — input. . . . . scanf(3S)
ungetc: push character back into — input stream. . . . . ungetc(3S)
fread, fwrite: binary — input/output. . . . . fread(3S)
stdio: standard buffered — input/output package. . . . . stdio(3S)
ferror, feof, clearerr, fileno: stream status — inquiries. . . . . ferror(3S)
uustat: uucp status — inquiry and job control. . . . . uustat(1C)
install: — install commands. . . . . install(1M)
— install: install commands. . . . . install(1M)
cpset: — install object files in binary directories. . . . . cpset(1M)
dcmplx, ichar, char: explicit FORTRAN type/ — int, ifix, idint, real, float, sngl, dble, cmplx, ... ftype(3F)
abs: return — integer absolute value. . . . . abs(3C)
a64l, l64a: convert between long — integer and base-64 ASCII string. . . . . a64l(3C)
sputl, sgetl: access long — integer data in a machine independent fashion. . . . . sputl(3X)
anint, dnint, nint, idnint: FORTRAN nearest — integer functions. . . . . round(3F)
aint, dint: FORTRAN — integer part intrinsic function. . . . . aint(3F)
strtol, atol, atoi: convert string to — integer. . . . . strtol(3C)
l3tol, ltol3: convert between 3-byte — integers and long integers. . . . . l3tol(3C)
ltol3: convert between 3-byte integers and long — integers. l3tol, ... l3tol(3C)
bcopy: — interactive block copy. . . . . bcopy(1M)
mailx: — interactive message processing system. . . . . mailx(1)
fsck, dfsck: file system consistency check and — interactive repair. . . . . fsck(1M)
rjestat: RJE status report and — interactive status console. . . . . rjestat(1C)
acia: Asynchronous Communications — Interface Adapter. . . . . acia(7)
err: error-logging — interface. . . . . err(7)
lp: MVME410 line printer — interface. . . . . lp(7)
plot: graphics — interface. . . . . plot(4)
plot: graphics — interface subroutines. . . . . plot(3X)
termio: general terminal — interface. . . . . termio(7)
tty: controlling terminal — interface. . . . . tty(7)
v10graph - VME/10 graphics subsystem — interface. . . . . v10graph(7)
spline: — interpolate smooth curve. . . . . spline(1G)
asa: — interpret ASA carriage control characters. . . . . asa(1)
sno: SNOBOL — interpreter. . . . . sno(1)
pipe: create an — interprocess channel. . . . . pipe(2)
ipcs: report — inter-process communication facilities status. . . . . ipcs(1)
stdipc: standard — interprocess communication package. . . . . stdipc(3C)
sleep: suspend execution for an — interval. . . . . sleep(1)
sleep: suspend execution for — interval. . . . . sleep(3C)
acos, dacos: FORTRAN arccosine — intrinsic function. . . . . acos(3F)
aint, dint: FORTRAN integer part — intrinsic function. . . . . aint(3F)
asin, dasin: FORTRAN arcsine — intrinsic function. . . . . asin(3F)
atan2, datan2: FORTRAN arctangent — intrinsic function. . . . . atan2(3F)
atan, datan: FORTRAN arctangent — intrinsic function. . . . . atan(3F)
conjg, dconjg: FORTRAN complex conjugate — intrinsic function. . . . . conjg(3F)
cos, dcos, ccos: FORTRAN cosine — intrinsic function. . . . . cos(3F)
cosh, dcosh: FORTRAN hyperbolic cosine — intrinsic function. . . . . cosh(3F)
dprod: double precision product — intrinsic function. . . . . dprod(3F)
exp, dexp, cexp: FORTRAN exponential — intrinsic function. . . . . exp(3F)
log10, alog10, dlog10: FORTRAN common logarithm — intrinsic function. . . . . log10(3F)
log, alog, dlog, clog: FORTRAN natural logarithm — intrinsic function. . . . . log(3F)
sign, isign, dsign: FORTRAN transfer-of-sign — intrinsic function. . . . . sign(3F)
sin, dsin, csin: FORTRAN sine — intrinsic function. . . . . sin(3F)
sinh, dsinh: FORTRAN hyperbolic sine — intrinsic function. . . . . sinh(3F)
sqrt, dsqrt, csqrt: FORTRAN square root — intrinsic function. . . . . sqrt(3F)
tan, dtan: FORTRAN tangent — intrinsic function. . . . . tan(3F)
tanh, dtanh: FORTRAN hyperbolic tangent — intrinsic function. . . . . tanh(3F)
dim, ddim, idim: positive difference — intrinsic functions. . . . . dim(3F)
mod, amod, dmod: FORTRAN remaindering — intrinsic functions. . . . . mod(3F)
lge, lgt, lle, llt: string comparision — intrinsic functions. . . . . strcmp(3F)
programs. — intro: introduction to commands and application . . . intro(1)
— intro: introduction to file formats. . . . . intro(4)
— intro: introduction to games. . . . . intro(6)
— intro: introduction to miscellaneous facilities. . . . . intro(5)

intro: introduction to special files. . . . . intro(7)
intro: introduction to subroutines and libraries. . . . . intro(3)
numbers. intro: introduction to system calls and error . . . . . intro(2)
and application programs. intro: introduction to system maintenance commands . intro(1M)
procedures. intro: introduction to system maintenance . . . . . intro(8)
intro: introduction to commands and application programs. . intro(1)
intro: introduction to file formats. . . . . intro(4)
intro: introduction to games. . . . . intro(6)
intro: introduction to miscellaneous facilities. . . . . intro(5)
intro: introduction to special files. . . . . intro(7)
intro: introduction to subroutines and libraries. . . . . intro(3)
intro: introduction to system calls and error numbers. . . . intro(2)
application programs. intro: introduction to system maintenance commands and . . intro(1M)
intro: introduction to system maintenance procedures. . . . intro(8)
ncheck: generate names from i-numbers. . . . . ncheck(1M)
ioctl: control device. . . . . ioctl(2)
abort: generate an IOT fault. . . . . abort(3C)
shared memory id. ipcrm: remove a message queue, semaphore set or . . . ipcrm(1)
status. ipcs: report inter-process communication facilities . . . ipcs(1)
irand, srand, rand: random number generator. . . . rand(3F)
isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph,/ . . . . . ctype(3C)
isalnum, isspace, ispunct, isprint, isgraph,/ isalpha, isupper, islower, isdigit, isxdigit, ... ctype(3C)
isspace, ispunct, isprint, isgraph, iscntrl, isascii: classify characters. /isxdigit, isalnum, ... ctype(3C)
ttyname, isatty: find name of a terminal. . . . . ttyname(3C)
isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii: classify characters. /isxdigit, ... ctype(3C)
isprint, isgraph,/ isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, ... ctype(3C)
/isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii: classify characters. . . . . ctype(3C)
function. sign, isign, dsign: FORTRAN transfer-of-sign intrinsic . . . sign(3F)
ispunct, isprint, isgraph,/ isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ... ctype(3C)
/isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii: classify/ . . . . . ctype(3C)
/islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl, isascii:/ . . . . . ctype(3C)
/isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, isgraph, iscntrl,/ . . . . . ctype(3C)
system: issue a shell command from FORTRAN . . . . . system(3F)
system: issue a shell command. . . . . system(3S)
issue: issue identification file. . . . . issue(4)
issue: issue identification file. . . . . issue(4)
isspace, ispunct, isprint, isgraph,/ isalpha, isupper, islower, isdigit, isxdigit, isalnum, ... ctype(3C)
isgraph,/ isalpha, isupper, islower, isdigit, isxdigit, isalnum, isspace, ispunct, isprint, ... ctype(3C)
news: print news items. . . . . news(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
join: relational database operator. . . . . join(1)
jotto: secret word game. . . . . jotto(6)
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate/ . . . . . drand48(3C)
makekey: generate encryption key. . . . . makekey(1)
killall: kill all active processes. . . . . killall(1M)
processes. kill: send a signal to a process or a group of . . . . . kill(2)
kill: terminate a process. . . . . kill(1)
killall: kill all active processes. . . . . killall(1M)
mem, kmem: core memory. . . . . mem(7)
quiz: test your knowledge. . . . . quiz(6)
long integers. l3tol, ltol3: convert between 3-byte integers and . . . l3tol(3C)
ASCII string. a64l, 164a: convert between long integer and base-64 . . . . a64l(3C)
volcopy, labelit: copy file systems with label checking. . . . . volcopy(1M)
volcopy, labelit: copy file systems with label checking. . . . . volcopy(1M)
awk: pattern scanning and processing language. . . . . awk(1)
bc: arbitrary-precision arithmetic language. . . . . bc(1)
efl: Extended FORTRAN Language. . . . . efl(1)
cpp: the C language preprocessor. . . . . cpp(1)
shell, the standard/restricted command programming language. sh, rsh: . . . . . sh(1)
lrk25: 25Mb LARK Module Drive for Universal Disk Driver. . . . lrk25(7)
lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 Driver. lark25(7)
lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 Driver. lark8(7)
VM22 Driver. lark25: 50Mb LARK Module Drive for VM21 Driver and lark25(7)
VM22 Driver. lark8: 16Mb LARK Module Drive for VM21 Driver and lark8(7)
prtacct, runacct,/ chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, ... acctsh(1M)
shl: shell layer manager. . . . . shl(1)
nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate uniformly distributed/ /lrand48, . . drand48(3C)
ld: link editor for common object files. . . . . ld(1)
ldclose, ldaclose: close a common object file. . . . . ldclose(3X)
archive file. ldahread: read the archive header of a member of an . ldahread(3X)
ldopen, ldaopen: open a common object file for reading. . . . . ldopen(3X)
ldclose, ldaclose: close a common object file. . . . . ldclose(3X)
numbers. frexp, ldexp, modf: manipulate parts of floating-point . . . . frexp(3C)

ldfcn: common object file access routines. . . . . ldfcn(4)
file. ldfhread: read the file header of a common object . . . ldfhread(3X)
ldgetname: retrieve symbol name for object file. . . . . ldgetname(3X)
a common object file function. ldlread, ldlinit, ldlitem: manipulate line number entries of . . ldlread(3X)
object file function. ldlread, ldlinit, ldlitem: manipulate line number entries of a common . ldlread(3X)
entries of a common object file function. ldlread, ldlinit, ldlitem: manipulate line number . . . ldlread(3X)
section of a common object file. ldlseek, ldnlseek: seek to line number entries of a . . . ldlseek(3X)
of a common object file. ldlseek, ldnlseek: seek to line number entries of a section . . . ldlseek(3X)
of a common object file. ldrseek, ldnrseek: seek to relocation entries of a section . . . . ldrseek(3X)
a common object file. ldshread, ldnshread: read an indexed/named section header of . . ldshread(3X)
common object file. ldsseek, ldnsseek: seek to an indexed/named section of a . . . . ldsseek(3X)
common object file. ldohseek: seek to the optional file header of a . . . . . ldohseek(3X)
reading. ldopen, ldaopen: open a common object file for . . . . ldopen(3X)
section of a common object file. ldrseek, ldnrseek: seek to relocation entries of a . . . . ldrseek(3X)
header of a common object file. ldshread, ldnshread: read an indexed/named section . . ldshread(3X)
section of a common object file. ldsseek, ldnsseek: seek to an indexed/named . . . . . ldsseek(3X)
entry of a common object file. ldtbindex: compute the index of a symbol table . . . . ldtbindex(3X)
common object file. ldtbread: read an indexed symbol table entry of a . . . ldtbread(3X)
object file. ldtbseek: seek to the symbol table of a common . . . . ldtbseek(3X)
len: return length of FORTRAN string. . . . . len(3F)
len: return length of FORTRAN string. . . . . len(3F)
getopt: get option letter from argument vector. . . . . getopt(3C)
lex: generate programs for simple lexical tasks. . . . . lex(1)
lex: generate programs for simple lexical tasks. . . . . lex(1)
lsearch, lfind: linear search and update. . . . . lsearch(3C)
functions. lge, lgt, lle, llt: string comparision intrinsic . . . . . strcmp(3F)
functions. lge, lgt, lle, llt: string comparision intrinsic . . . . . strcmp(3F)
intro: introduction to subroutines and libraries. . . . . intro(3)
lorder: find ordering relation for an object library. . . . . lorder(1)
ar: archive and library maintainer for portable archives. . . . . ar(1)
ulimit: get and set user limits. . . . . ulimit(2)
dial: establish an out-going terminal line connection. . . . . dial(3C)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
col: filter reverse line feeds. . . . . col(1)
line: read one line. . . . . line(1)
linenum: line number entries in a common object file. . . . . linenum(4)
function. ldlread, ldlinit, ldlitem: manipulate line number entries of a common object file . . . . . ldlread(3X)
file. ldlseek, ldnlseek: seek to line number entries of a section of a common object . . ldlseek(3X)
strip: strip symbol and line number information from an object file. . . . . strip(1)
nl: line numbering filter. . . . . nl(1)
cut: cut out selected fields of each line of a file. . . . . cut(1)
lpd: line printer daemon. . . . . lpd(1C)
lp: MVME410 line printer interface. . . . . lp(7)
lp, cancel: send/cancel requests to an LP line printer. . . . . lp(1)
line: read one line. . . . . line(1)
lsearch, lfind: linear search and update. . . . . lsearch(3C)
file. linenum: line number entries in a common object . . . linenum(4)
comm: select or reject lines common to two sorted files. . . . . comm(1)
uniq: report repeated lines in a file. . . . . uniq(1)
merge same lines of several files or subsequent lines of one file. paste: . . . . . paste(1)
file. paste: merge same lines of several files or subsequent lines of one . . . . paste(1)
link, unlink: exercise link and unlink system calls. . . . . link(1M)
ld: link editor for common object files. . . . . ld(1)
a.out: common assembler and link editor output. . . . . a.out(4)
link: link to a file. . . . . link(2)
cp, ln, mv: copy, link or move files. . . . . cp(1)
link: link to a file. . . . . link(2)
calls. link, unlink: exercise link and unlink system . . . . . link(1M)
lint: a C program checker. . . . . lint(1)
ls: list contents of directories. . . . . ls(1)
ff: list filenames and statistics for a file system. . . . . ff(1M)
nlist: get entries from name list. . . . . nlist(3C)
nm: print name list of common object file. . . . . nm(1)
checklist: list of file systems processed by fsck. . . . . checklist(4)
varargs: handle variable argument list. . . . . varargs(5)
print formatted output of a varargs argument list. vprintf, vfprintf, vsprintf: . . . . . vprintf(3S)
xargs: construct argument list(s) and execute command. . . . . xargs(1)
lge, lgt, lle, llt: string comparision intrinsic functions. . . . . strcmp(3F)
lge, lgt, lle, llt: string comparision intrinsic functions. . . . . strcmp(3F)
cp, ln, mv: copy, link or move files. . . . . cp(1)
time to string. ctime, localtime, gmtime, asctime, tzset: convert date and . . ctime(3C)
index: return location of FORTRAN substring. . . . . index(3F)
end, etext, edata: last locations in program. . . . . end(3C)
plock: lock process, text, or data in memory. . . . . plock(2)
intrinsic function. log, alog, dlog, clog: FORTRAN natural logarithm . . . log(3F)

gamma: log gamma function. . . . . gamma(3M)
newgrp: log in to a new group. . . . . newgrp(1)
power, square root functions. exp, log, log10, pow, sqrt: exponential, logarithm, ... exp(3M)
intrinsic function. log10, alog10, dlog10: FORTRAN common logarithm . log10(3F)
square root functions. exp, log, log10, pow, sqrt: exponential, logarithm, power, . . . exp(3M)
log10, alog10, dlog10: FORTRAN common logarithm intrinsic function. . . . . log10(3F)
log, alog, dlog, clog: FORTRAN natural logarithm intrinsic function. . . . . log(3F)
exp, log, log10, pow, sqrt: exponential, logarithm, power, square root functions. . . . . exp(3M)
errpt: process a report of logged errors. . . . . errpt(1M)
getlogin: get login name. . . . . getlogin(3C)
logname: get login name. . . . . logname(1)
cuserid: get character login name of the user. . . . . cuserid(3S)
logname: return login name of user. . . . . logname(3X)
passwd: change login password. . . . . passwd(1)
login: sign on. . . . . login(1)
profile: setting up an environment at login time. . . . . profile(4)
logname: get login name. . . . . logname(1)
logname: return login name of user. . . . . logname(3X)
a64l, l64a: convert between long integer and base-64 ASCII string. . . . . a64l(3C)
fashion.. sputl, sgetl: access long integer data in a machine independent . . . . . sputl(3X)
l3tol, ltol3: convert between 3-byte integers and long integers. . . . . l3tol(3C)
setjmp, longjmp: non-local goto. . . . . setjmp(3C)
library. lorder: find ordering relation for an object . . . . . lorder(1)
nice: run a command at low priority. . . . . nice(1)
printer. lp, cancel: send/cancel requests to an LP line . . . . . lp(1)
lp, cancel: send/cancel requests to an LP line printer. . . . . lp(1)
lp: MVME410 line printer interface. . . . . lp(7)
enable, disable: enable/disable LP printers. . . . . enable(1)
lpsched, lpshut, lpmove: start/stop the LP request scheduler and move requests. . . . . lpsched(1M)
accept, reject: allow/prevent LP requests. . . . . accept(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
lpstat: print LP status information. . . . . lpstat(1)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
lpd: line printer daemon. . . . . lpd(1C)
move requests. lpsched, lpshut, lpmove: start/stop the LP request scheduler and . . . lpsched(1M)
scheduler and move requests. lpsched, lpshut, lpmove: start/stop the LP request . . . lpsched(1M)
and move requests. lpsched, lpshut, lpmove: start/stop the LP request scheduler . . lpsched(1M)
lpstat: print LP status information. . . . . lpstat(1)
seed48, lcong48: generate/ drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, ... drand48(3C)
Driver. lrk25: 25Mb LARK Module Drive for Universal Disk . lrk25(7)
ls: list contents of directories. . . . . ls(1)
lsearch, lfind: linear search and update. . . . . lsearch(3C)
lseek: move read/write file pointer. . . . . lseek(2)
and, or, xor, not, lshift, rshift: FORTRAN bitwise Boolean functions. . . bool(3F)
integers. l3tol, ltol3: convert between 3-byte integers and long . . . . l3tol(3C)
m4: macro processor. . . . . m4(1)
m400: MVME400 Dual RS-232C Serial Port Module. . m400(7)
fscv: convert files between M68000 and VAX-11/780 processors. . . . . fscv(1M)
type. pdp11, u3b, vax, m68k: provide truth value about your processor . . . machid(1)
general driver for all disk units supported by the M68KVM21 disk controller. ud: . . . . . ud(7)
general driver for all disk units supported by the M68KVM21 disk controller. vm21: default . . . . . vm21(7)
general driver for all disk units supported by the M68KVM22 disk controller. vm22: default . . . . . vm22(7)
sputl, sgetl: access long integer data in a machine independent fashion.. . . . . sputl(3X)
values: machine-dependent values. . . . . values(5)
m4: macro processor. . . . . m4(1)
mail, rmail: send mail to users or read mail. . . . . mail(1)
mail, rmail: send mail to users or read mail. . . . . mail(1)
mail, rmail: send mail to users or read mail. . . . . mail(1)
mailx: interactive message processing system. . . . . mailx(1)
malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)
free, realloc, calloc, mallopt, mallinfo: fast main memory allocator. malloc, ... malloc(3X)
programs. make: maintain, update, and regenerate groups of . . . . . make(1)
ar: archive and library maintainer for portable archives. . . . . ar(1)
intro: introduction to system maintenance commands and application programs. . . intro(1M)
intro: introduction to system maintenance procedures. . . . . intro(8)
delta: make a delta (change) to an SCCS file. . . . . delta(1)
mkdir: make a directory. . . . . mkdir(1)
mknod: make a directory, or a special or ordinary file. . . . . mknod(2)
mktemp: make a unique filename. . . . . mktemp(3C)
programs. make: maintain, update, and regenerate groups of . . . make(1)
banner: make posters. . . . . banner(1)
makekey: generate encryption key. . . . . makekey(1)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory allocator. . . . . malloc(3X)
allocator. malloc, free, realloc, calloc: main memory . . . . . malloc(3C)
fast main memory allocator. malloc, free, realloc, calloc, mallopt, mallinfo: . . . . malloc(3X)

malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory allocator. . . . . malloc(3X)
man, manprog: print entries in this manual. . . . . man(1)
tsearch, tfind, tdelete, twalk: manage binary search trees. . . . . tsearch(3C)
hsearch, hcreate, hdestroy: manage hash search tables. . . . . hsearch(3C)
shl: shell layer manager. . . . . shl(1)
fwtmp, wtmpfix: manipulate connect accounting records. . . . . fwtmp(1M)
file function. ldlread, ldlinit, ldlitem: manipulate line number entries of a common object . . ldlread(3X)
frexp, ldexp, modf: manipulate parts of floating-point numbers. . . . . frexp(3C)
man, manprog: print entries in this manual. . . . . man(1)
man, manprog: print entries in this manual. . . . . man(1)
ascii: map of ASCII character set. . . . . ascii(5)
diffmk: mark differences between files. . . . . diffmk(1)
umask: set file-creation mode mask. . . . . umask(1)
umask: set and get file creation mask. . . . . umask(2)
master: master device information table. . . . . master.dec(4)
master: master device information table. . . . . master.dec(4)
regexp: regular expression compile and match routines. . . . . regexp(5)
math: math functions and constants. . . . . math(5)
math: math functions and constants. . . . . math(5)
matherr: error-handling function. . . . . matherr(3M)
maximum-value functions. max, max0, amax0, max1, amax1, dmax1: FORTRAN . max(3F)
maximum-value functions. max, max0, amax0, max1, amax1, dmax1: FORTRAN . . . . max(3F)
functions. max, max0, amax0, max1, amax1, dmax1: FORTRAN maximum-value . . max(3F)
max, max0, amax0, max1, amax1, dmax1: FORTRAN maximum-value functions. . . . . max(3F)
maze: generate a maze. . . . . maze(6)
maze: generate a maze. . . . . maze(6)
mclock: return FORTRAN time accounting. . . . . mclock(3F)
mem, kmem: core memory. . . . . mem(7)
operations. memccpy, memchr, memcmp, memcpy, memset: memory memory(3C)
memccpy, memchr, memcmp, memcpy, memset: memory operations. memory(3C)
memccpy, memchr, memcmp, memcpy, memset: memory operations. . . . memory(3C)
memccpy, memchr, memcmp, memcpy, memset: memory operations. . . . . memory(3C)
malloc, free, realloc, calloc: main memory allocator. . . . . malloc(3C)
free, realloc, calloc, mallopt, mallinfo: fast main memory allocator. malloc, ... malloc(3X)
shmctl: shared memory control operations. . . . . shmctl(2)
remove a message queue, semaphore set or shared memory id. ipcrm: . . . . . ipcrm(1)
mem, kmem: core memory. . . . . mem(7)
memccpy, memchr, memcmp, memcpy, memset: memory operations. . . . . memory(3C)
shmat, shmdt: shared memory operations. . . . . shmop(2)
plock: lock process, text, or data in memory. . . . . plock(2)
shmget: get shared memory segment. . . . . shmget(2)
memccpy, memchr, memcmp, memcpy, memset: memory operations. . . . . memory(3C)
sort: sort and/or merge files. . . . . sort(1)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
lines of one file. paste: merge same lines of several files or subsequent . . . . paste(1)
mesg: permit or deny messages. . . . . mesg(1)
msgctl: message control operations. . . . . msgctl(2)
msgsnd, msgrcv: message operations. . . . . msgop(2)
mailx: interactive message processing system. . . . . mailx(1)
msgget: get message queue. . . . . msgget(2)
ipcrm: remove a message queue, semaphore set or shared memory id. . . ipcrm(1)
mesg: permit or deny messages. . . . . mesg(1)
perror, errno, sys_errlist, sys_nerr: system error messages. . . . . perror(3C)
minimum-value functions. min, min0, amin0, min1, amin1, dmin1: FORTRAN . . min(3F)
minimum-value functions. min, min0, amin0, min1, amin1, dmin1: FORTRAN . . . . min(3F)
functions. min, min0, amin0, min1, amin1, dmin1: FORTRAN minimum-value . . . min(3F)
min, min0, amin0, min1, amin1, dmin1: FORTRAN minimum-value functions. . . . . min(3F)
mk: how to remake the system and commands. . . . . mk(8)
mkdir: make a directory. . . . . mkdir(1)
mkfs: construct a file system. . . . . mkfs(1M)
mknod: build special file. . . . . mknod(1M)
file. mknod: make a directory, or a special or ordinary . . . mknod(2)
mktemp: make a unique filename. . . . . mktemp(3C)
mnttab: mounted file system table. . . . . mnttab(4)
functions. mod, amod, dmod: FORTRAN remaindering intrinsic . mod(3F)
chmod: change mode. . . . . chmod(1)
umask: set file-creation mode mask. . . . . umask(1)
chmod: change mode of file. . . . . chmod(2)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
bs: a compiler/interpreter for modest-sized programs. . . . . bs(1)
frexp, ldexp, modf: manipulate parts of floating-point numbers. . . frexp(3C)
touch: update access and modification times of a file. . . . . touch(1)
utime: set file access and modification times. . . . . utime(2)
cm16: 16Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm16(7)
cm80: 80Mb Cartridge Module Drive for Universal Disk Driver. . . . . cm80(7)

lrk25: 25Mb LARK — Module Drive for Universal Disk Driver. . . . . lrk25(7)
cmd16: 16Mb Cartridge — Module Drive for VM21 Driver and VM22 Driver. . . cmd16(7)
cmd80: 80Mb Cartridge — Module Drive for VM21 Driver and VM22 Driver. . . cmd80(7)
lark25: 50Mb LARK — Module Drive for VM21 Driver and VM22 Driver. . . lark25(7)
lark8: 16Mb LARK — Module Drive for VM21 Driver and VM22 Driver. . . lark8(7)
m400: MVME400 Dual RS-232C Serial Port — Module. . . . . m400(7)
runacct,/ chargefee, ckpacct, dodisk, lastlogin, — monacct, nulladm, prctmp, prdaily, prtacct, ... acctsh(1M)
— monitor: prepare execution profile. . . . . monitor(3C)
uusub: — monitor uucp network. . . . . uusub(1M)
— moo: guessing game. . . . . moo(6)
mount: — mount a file system. . . . . mount(2)
mount, umount: — mount and dismount file system. . . . . mount(1M)
— mount: mount a file system. . . . . mount(2)
setmnt: establish — mount table. . . . . setmnt(1M)
— mount, umount: mount and dismount file system. . . . mount(1M)
mnttab: — mounted file system table. . . . . mnttab(4)
mvdir: — move a directory. . . . . mvdir(1M)
cp, ln, mv: copy, link or — move files. . . . . cp(1)
lseek: — move read/write file pointer. . . . . lseek(2)
lpmove: start/stop the LP request scheduler and — move requests. lpsched, lpshut, ... lpsched(1M)
generate/ drand48, erand48, lrand48, nrand48, — mrand48, jrand48, srand48, seed48, lcong48: . . . . drand48(3C)
— msgctl: message control operations. . . . . msgctl(2)
— msgget: get message queue. . . . . msgget(2)
msgsnd, — msgrcv: message operations. . . . . msgop(2)
— msgsnd, msgrcv: message operations. . . . . msgop(2)
cp, ln, — mv: copy, link or move files. . . . . cp(1)
— mvdir: move a directory. . . . . mvdir(1M)
m400: — MVME400 Dual RS-232C Serial Port Module. . . . . m400(7)
lp: — MVME410 line printer interface. . . . . lp(7)
log, alog, dlog, clog: FORTRAN — natural logarithm intrinsic function. . . . . log(3F)
— ncheck: generate names from i-numbers. . . . . ncheck(1M)
anint, dnint, nint, idnint: FORTRAN — nearest integer functions. . . . . round(3F)
stat: statistical — network useful with graphical commands. . . . . stat(1G)
uusub: monitor uucp — network. . . . . uusub(1M)
— newform: change the format of a text file. . . . . newform(1)
— newgrp: log in to a new group. . . . . newgrp(1)
news: print — news items. . . . . news(1)
— news: print news items. . . . . news(1)
— nice: change priority of a process. . . . . nice(2)
— nice: run a command at low priority. . . . . nice(1)
anint, dnint, — nint, idnint: FORTRAN nearest integer functions. . . . round(3F)
— nl: line numbering filter. . . . . nl(1)
— nlist: get entries from name list. . . . . nlist(3C)
— nm: print name list of common object file. . . . . nm(1)
— nohup: run a command immune to hangups and quits. nohup(1)
setjmp, longjmp: — non-local goto. . . . . setjmp(3C)
functions. and, or, xor, — not, lshift, rshift: FORTRAN bitwise Boolean . . . . . bool(3F)
lcong48: generate/ drand48, erand48, lrand48, — nrand48, mrand48, jrand48, srand48, seed48, ... drand48(3C)
null: the — null file. . . . . null(7)
— null: the null file. . . . . null(7)
chargefee, ckpacct, dodisk, lastlogin, monacct, — nulladm, prctmp, prdaily, prtacct, runacct,/ . . . . . acctsh(1M)
nl: line — numbering filter. . . . . nl(1)
graphics: access graphical and — numerical commands. . . . . graphics(1G)
ldfcn: common — object file access routines. . . . . ldfcn(4)
conv: — object file converter. . . . . conv(1)
dump: dump selected parts of an — object file. . . . . dump(1)
ldopen, ldaopen: open a common — object file for reading. . . . . ldopen(3X)
ldlitem: manipulate line number entries of a common — object file function. ldlread, ldlinit, ... ldlread(3X)
ldclose, ldaclose: close a common — object file. . . . . ldclose(3X)
ldfhread: read the file header of a common — object file. . . . . ldfhread(3X)
ldgetname: retrieve symbol name for — object file. . . . . ldgetname(3X)
to line number entries of a section of a common — object file. ldlseek, ldnlseek: seek . . . . . ldlseek(3X)
seek to the optional file header of a common — object file. ldohseek: . . . . . ldohseek(3X)
seek to relocation entries of a section of a common — object file. ldrseek, ldnrseek: . . . . . ldrseek(3X)
read an indexed/named section header of a common — object file. ldshread, ldnshread: . . . . . ldshread(3X)
seek to an indexed/named section of a common — object file. ldsseek, ldnsseek: . . . . . ldsseek(3X)
the index of a symbol table entry of a common — object file. ldtbindex: compute . . . . . ldtbindex(3X)
read an indexed symbol table entry of a common — object file. ldtbread: . . . . . ldtbread(3X)
ldtbseek: seek to the symbol table of a common — object file. . . . . ldtbseek(3X)
linenum: line number entries in a common — object file. . . . . linenum(4)
nm: print name list of common — object file. . . . . nm(1)
reloc: relocation information for a common — object file. . . . . reloc(4)
scnhdr: section header for a common — object file. . . . . scnhdr(4)
strip symbol and line number information from an — object file. strip: . . . . . strip(1)
syms: common — object file symbol table format. . . . . syms(4)

filehdr: file header for common — object files. . . . filehdr(4)
cpset: install — object files in binary directories. . . . cpset(1M)
ld: link editor for common — object files. . . . ld(1)
size: print section sizes of common — object files. . . . size(1)
lorder: find ordering relation for an — object library. . . . lorder(1)
sky: — obtain ephemerides. . . . sky(6)
getgrent, getgrgid, getgrnam, setgrent, endgrent: — obtain. . . . getgrent(3C)
od: — octal dump. . . . od(1)
— od: octal dump. . . . od(1)
ldopen, ldaopen: — open a common object file for reading. . . . ldopen(3X)
fopen, freopen, fdopen: — open a stream. . . . fopen(3S)
dup: duplicate an — open file descriptor. . . . dup(2)
open: — open for reading or writing. . . . open(2)
— open: open for reading or writing. . . . open(2)
bo.macs: bootstrap — operating procedure for system restart on EXORmacs. . . bo.macs(8)
bo.vme: bootstrap — operating procedure for system restart on VME/10. . . bo.vme(8)
prf: — operating system profiler. . . . prf(7)
prfld, prfstat, prfdc, prfsnap, prfpr: — operating system profiler. . . . profiler(1M)
uname: get name of current — operating system. . . . uname(2)
memccpy, memchr, memcmp, memcpy, memset: memory — operations. . . . memory(3C)
msgctl: message control — operations. . . . msgctl(2)
msgsnd, msgrcv: message — operations. . . . msgop(2)
ops.macs: EXORmacs — operations. . . . ops.macs(8)
semctl: semaphore control — operations. . . . semctl(2)
semop: semaphore — operations. . . . semop(2)
shmctl: shared memory control — operations. . . . shmctl(2)
shmat, shmdt: shared memory — operations. . . . shmop(2)
strrchr, strpbrk, strspn, strcspn, strtok: string — operations. /strcpy, strncpy, strlen, strchr, ... string(3C)
join: relational database — operator. . . . join(1)
— ops.macs: EXORmacs operations. . . . ops.macs(8)
dcopy: copy file systems for — optimal access time. . . . dcopy(1M)
curses: CRT screen handling and — optimization package. . . . curses(3X)
getopt: get — option letter from argument vector. . . . getopt(3C)
aouthdr: — optional aout header. . . . aouthdr(4)
ldohseek: seek to the — optional file header of a common object file. . . . ldohseek(3X)
fcntl: file control — options. . . . fcntl(5)
stty: set the — options for a terminal. . . . stty(1)
getopt: parse command — options. . . . getopt(1)
Boolean functions. and, — or, xor, not, lshift, rshift: FORTRAN bitwise . . . bool(3F)
lorder: find — ordering relation for an object library. . . . lorder(1)
mknod: make a directory, or a special or — ordinary file. . . . mknod(2)
dial: establish an — out-going terminal line connection. . . . dial(3C)
a.out: common assembler and link editor — output. . . . a.out(4)
vprintf, vfprintf, vsprintf: print formatted — output of a varargs argument list. . . . vprintf(3S)
printf, fprintf, sprintf: print formatted — output. . . . printf(3S)
commands. acctdisk, acctdusg, accton, acctwtmp: — overview of accounting and miscellaneous accounting . acct(1M)
chown: change — owner and group of a file. . . . chown(2)
chown, chgrp: change — owner or group. . . . chown(1)
— pack, pcat, unpack: compress and expand files. . . . pack(1)
curses: CRT screen handling and optimization — package. . . . curses(3X)
sa1, sa2, sadc: system activity report — package. . . . sar(1M)
stdio: standard buffered input/output — package. . . . stdio(3S)
stdipc: standard interprocess communication — package. . . . stdipc(3C)
4014: — paginator for the Tektronix 4014 terminal. . . . 4014(1)
getpgrp, getppid: get process, process group, and — parent process IDs. getpid, . . . getpid(2)
getopt: — parse command options. . . . getopt(1)
— passwd: change login password. . . . passwd(1)
— passwd: password file. . . . passwd(4)
getpwnam, setpwent, endpwent, fgetpwent: get — password file entry. getpwent, getpwuid, . . . getpwent(3C)
putpwent: write — password file entry. . . . putpwent(3C)
passwd: — password file. . . . passwd(4)
getpass: read a — password. . . . getpass(3C)
passwd: change login — password. . . . passwd(1)
pwck, grpck: — password/group file checkers. . . . pwck(1M)
subsequent lines of one file. — paste: merge same lines of several files or . . . paste(1)
getcwd: get — pathname of current working directory. . . . getcwd(3C)
basename, dirname: deliver portions of — pathnames. . . . basename(1)
grep, egrep, fgrep: search a file for a — pattern. . . . grep(1)
awk: — pattern scanning and processing language. . . . awk(1)
— pause: suspend process until signal. . . . pause(2)
pack, — pcat, unpack: compress and expand files. . . . pack(1)
popen, — pclose: initiate pipe to/from a process. . . . popen(3S)
your processor type. — pdp11, u3b, vax, m68k: provide truth value about . . machid(1)
mesg: — permit or deny messages. . . . mesg(1)
ptx: — permuted index. . . . ptx(1)

acct:	per-process accounting file format.	acct(4)
acctcms: command summary from	per-process accounting records.	acctcms(1M)
messages.	perror, errno, sys_errlist, sys_nerr: system error	perror(3C)
pg: file	perusal filter for soft-copy terminals.	pg(1)
	pg: file perusal filter for soft-copy terminals.	pg(1)
split: split a file into	pieces.	split(1)
	pipe: create an interprocess channel.	pipe(2)
tee:	pipe fitting.	tee(1)
popen, pclose: initiate	pipe to/from a process.	popen(3S)
	plock: lock process, text, or data in memory.	plock(2)
	plot: graphics interface.	plot(4)
	plot: graphics interface subroutines.	plot(3X)
	pnch: file format for card images.	pnch(4)
fseek, rewind, ftell: reposition a file	pointer in a stream.	fseek(3S)
lseek: move read/write file	pointer.	lseek(2)
	popen, pclose: initiate pipe to/from a process.	popen(3S)
m400: MVME400 Dual RS-232C Serial	Port Module.	m400(7)
ar: archive and library maintainer for	portable archives.	ar(1)
basename, dirname: deliver	portions of pathnames.	basename(1)
dim, ddim, idim:	positive difference intrinsic functions.	dim(3F)
banner: make	posters.	banner(1)
root functions. exp, log, log10,	pow, sqrt: exponential, logarithm, power, square	exp(3M)
exp, log, log10, pow, sqrt: exponential, logarithm,	power, square root functions.	exp(3M)
brc, bcheckrc, rc,	powerfail: system initialization shell scripts.	brc(1M)
	pr: print files.	pr(1)
/ckpacct, dodisk, lastlogin, monacct, nulladm,	prctmp, prdaily, prtacct, runacct, shutacct,/	acctsh(1M)
/dodisk, lastlogin, monacct, nulladm, prctmp,	prdaily, prtacct, runacct, shutacct, startup,/	acctsh(1M)
dprod: double	precision product intrinsic function.	dprod(3F)
monitor:	prepare execution profile.	monitor(3C)
cpp: the C language	preprocessor.	cpp(1)
unget: undo a	previous get of an SCCS file.	unget(1)
	prf: operating system profiler.	prf(7)
prfld, prfstat,	prfdc, prfsnap, prfpr: operating system profiler.	profiler(1M)
system profiler.	prfld, prfstat, prfdc, prfsnap, prfpr: operating	profiler(1M)
prfld, prfstat, prfdc, prfsnap,	prfpr: operating system profiler.	profiler(1M)
prfld, prfstat, prfdc,	prfsnap, prfpr: operating system profiler.	profiler(1M)
profiler. prfld,	prfstat, prfdc, prfsnap, prfpr: operating system	profiler(1M)
gps: graphical	primitive string, format of graphical files.	gps(4)
types:	primitive system data types.	types(5)
prs:	print an SCCS file.	prs(1)
date:	print and set the date.	date(1)
cal:	print calendar.	cal(1)
sum:	print checksum and block count of a file.	sum(1)
sact:	print current SCCS file editing activity.	sact(1)
man, manprog:	print entries in this manual.	man(1)
cat: concatenate and	print files.	cat(1)
pr:	print files.	pr(1)
vprintf, vfprintf, vsprintf:	print formatted output of a varargs argument list.	vprintf(3S)
printf, fprintf, sprintf:	print formatted output.	printf(3S)
lpstat:	print LP status information.	lpstat(1)
nm:	print name list of common object file.	nm(1)
uname:	print name of current UNIX System.	uname(1)
news:	print news items.	news(1)
acctcom: search and	print process accounting file(s).	acctcom(1)
size:	print section sizes of common object files.	size(1)
id:	print user and group IDs and names.	id(1)
lpd: line	printer daemon.	lpd(1C)
lp: MVME410 line	printer interface.	lp(7)
lp, cancel: send/cancel requests to an LP line	printer.	lp(1)
enable, disable: enable/disable LP	printers.	enable(1)
	printf, fprintf, sprintf: print formatted output.	printf(3S)
nice: run a command at low	priority.	nice(1)
nice: change	priority of a process.	nice(2)
errpt:	process a report of logged errors.	errpt(1M)
acct: enable or disable	process accounting.	acct(2)
acctprc1, acctprc2:	process accounting.	acctprc(1M)
acctcom: search and print	process accounting file(s).	acctcom(1)
times: get	process and child process times.	times(2)
init, telinit:	process control initialization.	init(1M)
timex: time a command; report	process data and system activity.	timex(1)
exit, _exit: terminate	process.	exit(2)
fork: create a new	process.	fork(2)
getpid, getpgrp, getppid: get process,	process group, and parent process IDs.	getpid(2)
setpgrp: set	process group ID.	setpgrp(2)
getppid: get process, process group, and parent	process IDs. getpid, getpgrp,	getpid(2)

inittab: script for the init process. . . . . inittab(4)
kill: terminate a process. . . . . kill(1)
nice: change priority of a process. . . . . nice(2)
kill: send a signal to a process or a group of processes. . . . . kill(2)
popen, pclose: initiate pipe to/from a process. . . . . popen(3S)
getpid, getpgrp, getppid: get process, process group, and parent process IDs. . . . . getpid(2)
ps: report process status. . . . . ps(1)
plock: lock process, text, or data in memory. . . . . plock(2)
times: get process and child process times. . . . . times(2)
wait: wait for child process to stop or terminate. . . . . wait(2)
ptrace: process trace. . . . . ptrace(2)
pause: suspend process until signal. . . . . pause(2)
wait: await completion of process. . . . . wait(1)
checklist: list of file systems processed by fsck. . . . . checklist(4)
kill: send a signal to a process or a group of processes. . . . . kill(2)
killall: kill all active processes. . . . . killall(1M)
fuser: identify processes using a file or file structure. . . . . fuser(1M)
awk: pattern scanning and processing language. . . . . awk(1)
shutdown: terminate all processing. . . . . shutdown(1M)
mailx: interactive message processing system. . . . . mailx(1)
m4: macro processor. . . . . m4(1)
u3b, vax, m68k: provide truth value about your processor type. pdp11, ... machid(1)
wffmt: format floppies for the VME/10 processor. . . . . wffmt(1M)
fscv: convert files between M68000 and VAX-11/780 processors. . . . . fscv(1M)
alarm: set a process's alarm clock. . . . . alarm(2)
dprod: double precision product intrinsic function. . . . . dprod(3F)
prof: display profile data. . . . . prof(1)
prof: profile within a function. . . . . prof(5)
profil: execution time profile. . . . . profil(2)
prof: display profile data. . . . . prof(1)
monitor: prepare execution profile. . . . . monitor(3C)
profil: execution time profile. . . . . profil(2)
profile: setting up an environment at login time. . . . profile(4)
prof: profile within a function. . . . . prof(5)
prf: operating system profiler. . . . . prf(7)
prfstat, prfdc, prfsnap, prfpr: operating system profiler. prfld, ... profiler(1M)
sadp: disk access profiler. . . . . sadp(1)
sh, rsh: shell, the standard/restricted command programming language. . . . . sh(1)
arithmetic: provide drill in number facts. . . . . arithmetic(6)
pdp11, u3b, vax, m68k: provide truth value about your processor type. . . . . machid(1)
true, false: provide truth values. . . . . true(1)
prs: print an SCCS file. . . . . prs(1)
/lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct:/ . . . . . acctsh(1M)
ps: report process status. . . . . ps(1)
sxt: pseudo-device driver. . . . . sxt(7)
seed48, lcong48: generate uniformly distributed pseudo-random numbers. /mrand48, jrand48, srand48, drand48(3C)
ptrace: process trace. . . . . ptrace(2)
ptx: permuted index. . . . . ptx(1)
ungetc: push character back into input stream. . . . . ungetc(3S)
on a stream. putc, putchar, fputc, putw: put character or word . . putc(3S)
stream. putc, putchar, fputc, putw: put character or word on a . . . putc(3S)
putenv: change or add value to environment. . . . . putenv(3C)
putpwent: write password file entry. . . . . putpwent(3C)
puts, fputs: put a string on a stream. . . . . puts(3S)
utmp file entry. getutent, getutid, getutline, pututline, setutent, endutent, utmpname: access . . . getut(3C)
putc, putchar, fputc, putw: put character or word on a stream. . . . . putc(3S)
pwck, grpck: password/group file checkers. . . . . pwck(1M)
pwd: working directory name. . . . . pwd(1)
qsort: quicker sort. . . . . qsort(3C)
tput: query terminfo database. . . . . tput(1)
msgget: get message queue. . . . . msgget(2)
ipcrm: remove a message queue, semaphore set or shared memory id. . . . . ipcrm(1)
qsort: quicker sort. . . . . qsort(3C)
nohup: run a command immune to hangups and quits. . . . . nohup(1)
quiz: test your knowledge. . . . . quiz(6)
irand, srand, rand: rand: random number generator. . . . . rand(3F)
rand, srand: simple random-number generator. . . . . rand(3C)
irand, srand, rand: random number generator. . . . . rand(3F)
rand, srand: simple random-number generator. . . . . rand(3C)
fsplit: split f77, ratfor, or efl files. . . . . fsplit(1)
ratfor: rational FORTRAN dialect. . . . . ratfor(1)
ratfor: rational FORTRAN dialect. . . . . ratfor(1)
brc, bcheckrc, rc, powerfail: system initialization shell scripts. . . . brc(1M)
getpass: read a password. . . . . getpass(3C)
object file. ldtbread: read an indexed symbol table entry of a common . . . ldtbread(3X)

object file. ldshread, ldnshread:	read an indexed/named section header of a common	ldshread(3X)
read:	read from file.	read(2)
mail, rmail: send mail to users or	read mail.	mail(1)
line:	read one line.	line(1)
	read: read from file.	read(2)
file. ldahread:	read the archive header of a member of an archive	ldahread(3X)
ldfhread:	read the file header of a common object file.	ldfhread(3X)
ldopen, ldaopen: open a common object file for	reading.	ldopen(3X)
open: open for	reading or writing.	open(2)
lseek: move	read/write file pointer.	lseek(2)
char: explicit FORTRAN type/ int, ifix, idint,	real, float, sngl, dble, cmplx, dcmplx, ichar,	ftype(3F)
malloc, free,	realloc, calloc: main memory allocator.	malloc(3C)
memory allocator. malloc, free,	realloc, calloc, mallopt, mallinfo: fast main	malloc(3X)
signal: specify what to do upon	receipt of a signal.	signal(2)
signal: specify FORTRAN action on	receipt of a system signal.	signal(3F)
command summary from per-process accounting	records. acctcms:	acctcms(1M)
errdead: extract error	records from dump.	errdead(1M)
fwtmp, wtmpfix: manipulate connect accounting	records.	fwtmp(1M)
frec:	recover files from a backup tape.	frec(1M)
ed,	red: text editor.	ed(1)
expression.	regcmp, regex: compile and execute a regular	regcmp(3X)
	regcmp: regular expression compile.	regcmp(1)
make: maintain, update, and	regenerate groups of programs.	make(1)
regcmp,	regex: compile and execute a regular expression.	regcmp(3X)
routines.	regexp: regular expression compile and match	regexp(5)
regexp:	regular expression compile and match routines.	regexp(5)
regcmp:	regular expression compile.	regcmp(1)
regcmp, regex: compile and execute a	regular expression.	regcmp(3X)
accept,	reject: allow/prevent LP requests.	accept(1M)
comm: select or	reject lines common to two sorted files.	comm(1)
lorder: find ordering	relation for an object library.	lorder(1)
join:	relational database operator.	join(1)
file.	reloc: relocation information for a common object	reloc(4)
file. ldrseek, ldnrseek: seek to	relocation entries of a section of a common object	ldrseek(3X)
reloc:	relocation information for a common object file.	reloc(4)
floor, ceil, fmod, fabs: floor, ceiling,	remainder, absolute value functions.	floor(3M)
mod, amod, dmod: FORTRAN	remaindering intrinsic functions.	mod(3F)
mk: how to	remake the system and commands.	mk(8)
calendar:	reminder service.	calendar(1)
ct: spawn getty to a	remote terminal.	ct(1C)
rmdel:	remove a delta from an SCCS file.	rmdel(1)
memory id. ipcrm:	remove a message queue, semaphore set or shared	ipcrm(1)
unlink:	remove directory entry.	unlink(2)
rm, rmdir:	remove files or directories.	rm(1)
file system consistency check and interactive	repair. fsck, dfsck:	fsck(1M)
uniq: report	repeated lines in a file.	uniq(1)
rjestat: RJE status	report and interactive status console.	rjestat(1C)
clock:	report CPU time used.	clock(3C)
status. ipcs:	report inter-process communication facilities	ipcs(1)
df:	report number of free disk blocks.	df(1M)
errpt: process a	report of logged errors.	errpt(1M)
sa1, sa2, sadc: system activity	report package.	sar(1M)
timex: time a command;	report process data and system activity.	timex(1)
ps:	report process status.	ps(1)
uniq:	report repeated lines in a file.	uniq(1)
trenter: enter a trouble	report.	trenter(1M)
sar: system activity	reporter.	sar(1)
fseek, rewind, ftell:	reposition a file pointer in a stream.	fseek(3S)
lpsched, lpshut, lpmove: start/stop the LP	request scheduler and move requests.	lpsched(1M)
accept, reject: allow/prevent LP	requests.	accept(1M)
start/stop the LP request scheduler and move	requests. lpsched, lpshut, lpmove:	lpsched(1M)
lp, cancel: send/cancel	requests to an LP line printer.	lp(1)
bo.macs: bootstrap operating procedure for system	restart on EXORmacs.	bo.macs(8)
bo.vme: bootstrap operating procedure for system	restart on VME/10.	bo.vme(8)
ldgetname:	retrieve symbol name for object file.	ldgetname(3X)
getarg:	return FORTRAN command-line argument.	getarg(3F)
getenv:	return FORTRAN environment variable.	getenv(3F)
mclock:	return FORTRAN time accounting.	mclock(3F)
abs:	return integer absolute value.	abs(3C)
len:	return length of FORTRAN string.	len(3F)
index:	return location of FORTRAN substring.	index(3F)
logname:	return login name of user.	logname(3X)
getenv:	return value for environment name.	getenv(3C)
stat: data	returned by stat system call.	stat(5)
reversi: a game of dramatic	reversals.	reversi(6)

col: filter — reverse line feeds. . . . col(1)
reversi: a game of dramatic reversals. . . . reversi(6)
stream. fseek, — rewind, ftell: reposition a file pointer in a . . . fseek(3S)
creat: create a new file or — rewrite an existing one. . . . creat(2)
rjestat: — RJE status report and interactive status console. . . . rjestat(1C)
console. — rjestat: RJE status report and interactive status . . . rjestat(1C)
rm, rmdir: remove files or directories. . . . rm(1)
mail, — rmail: send mail to users or read mail. . . . mail(1)
rmdel: remove a delta from an SCCS file. . . . rmdel(1)
rm, — rmdir: remove files or directories. . . . rm(1)
chroot: change — root directory. . . . chroot(2)
chroot: change — root directory for a command. . . . chroot(1M)
pow, sqrt: exponential, logarithm, power, square — root functions. exp, log, log10, . . . exp(3M)
sqrt, dsqrt, csqrt: FORTRAN square — root intrinsic function. . . . sqrt(3F)
hpd, erase, hardcopy, tekset, td: graphical device — routines and filters. . . . gdev(1G)
ldfcn: common object file access — routines. . . . ldfcn(4)
regexp: regular expression compile and match — routines. . . . regexp(5)
toc: graphical table of contents — routines. . . . toc(1G)
m400: MVME400 Dual — RS-232C Serial Port Module. . . . m400(7)
programming language. sh, — rsh: shell, the standard/restricted command . . . sh(1)
and, or, xor, not, lshift, — rshift: FORTRAN bitwise Boolean functions. . . . bool(3F)
nice: — run a command at low priority. . . . nice(1)
nohup: — run a command immune to hangups and quits. . . . nohup(1)
runacct: — run daily accounting. . . . runacct(1M)
runacct: run daily accounting. . . . runacct(1M)
/monacct, nulladm, prctmp, prdaily, prtacct, — runacct, shutacct, startup, turnacct: shell/ . . . acctsh(1M)
sa1, sa2, sadc: system activity report package. . . . sar(1M)
sa1, — sa2, sadc: system activity report package. . . . sar(1M)
Driver. — sa400fl22: 5¼-inch Floppy Disk Drive for VM22 . . . sa400fl22(7)
Winchester Disk Driver. — sa400flwd: 5¼-inch Floppy Disk Drive for the . . . sa400flwd(7)
Driver. — sa800fl21: 8-inch Floppy Disk Drive for VM21 . . . sa800fl21(7)
Driver. — sa800fl22: 8-inch Floppy Disk Drive for VM22 . . . sa800fl22(7)
sact: print current SCCS file editing activity. . . . sact(1)
sa1, sa2, — sadc: system activity report package. . . . sar(1M)
sadp: disk access profiler. . . . sadp(1)
sag: system activity graph. . . . sag(1G)
sar: system activity reporter. . . . sar(1)
brk, — sbrk: change data segment space allocation. . . . brk(2)
scanf, fscanf, sscanf: convert formatted input. . . . scanf(3S)
bfs: big file — scanner. . . . bfs(1)
awk: pattern — scanning and processing language. . . . awk(1)
scc: C compiler for stand-alone programs. . . . scc(1)
cdc: change the delta commentary of an — SCCS delta. . . . cdc(1)
comb: combine — SCCS deltas. . . . comb(1)
delta: make a delta (change) to an — SCCS file. . . . delta(1)
sact: print current — SCCS file editing activity. . . . sact(1)
get: get a version of an — SCCS file. . . . get(1)
prs: print an — SCCS file. . . . prs(1)
rmdel: remove a delta from an — SCCS file. . . . rmdel(1)
sccsdiff: compare two versions of an — SCCS file. . . . sccsdiff(1)
sccsfile: format of — SCCS file. . . . sccsfile(4)
unget: undo a previous get of an — SCCS file. . . . unget(1)
val: validate — SCCS file. . . . val(1)
admin: create and administer — SCCS files. . . . admin(1)
what: identify — SCCS files. . . . what(1)
sccsdiff: compare two versions of an SCCS file. . . . sccsdiff(1)
sccsfile: format of SCCS file. . . . sccsfile(4)
lpsched, lpshut, lpmove: start/stop the LP request — scheduler and move requests. . . . lpsched(1M)
scnhdr: section header for a common object file. . . . scnhdr(4)
curses: CRT — screen handling and optimization package. . . . curses(3X)
ex. vi: — screen-oriented (visual) display editor based on . . . vi(1)
inittab: — script for the init process. . . . inittab(4)
rc, powerfail: system initialization shell — scripts. brc, bcheckrc, . . . brc(1M)
sdb: symbolic debugger. . . . sdb(1)
sdiff: side-by-side difference program. . . . sdiff(1)
grep, egrep, fgrep: — search a file for a pattern. . . . grep(1)
acctcom: — search and print process accounting file(s). . . . acctcom(1)
lsearch, lfind: linear — search and update. . . . lsearch(3C)
bsearch: binary — search. . . . bsearch(3C)
hsearch, hcreate, hdestroy: manage hash — search tables. . . . hsearch(3C)
tsearch, tfind, tdelete, twalk: manage binary — search trees. . . . tsearch(3C)
jotto: — secret word game. . . . jotto(6)
scnhdr: — section header for a common object file. . . . scnhdr(4)
ldshread, ldnshread: read an indexed/named — section header of a common object file. . . . ldshread(3X)
ldlseek, ldnlseek: seek to line number entries of a — section of a common object file. . . . ldlseek(3X)

ldrseek, ldnrseek: seek to relocation entries of a section of a common object file. . . . . ldrseek(3X)
ldsseek, ldnsseek: seek to an indexed/named section of a common object file. . . . . ldsseek(3X)
size: print section sizes of common object files. . . . . size(1)
sed: stream editor. . . . . sed(1)
/lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate uniformly distributed/ . . . drand48(3C)
object file. ldsseek, ldnsseek: seek to an indexed/named section of a common . . . . ldsseek(3X)
common object file. ldlseek, ldnlseek: seek to line number entries of a section of a . . . . ldlseek(3X)
object file. ldrseek, ldnrseek: seek to relocation entries of a section of a common . . ldrseek(3X)
file. ldohseek: seek to the optional file header of a common object . . ldohseek(3X)
ldtbseek: seek to the symbol table of a common object file. . . . ldtbseek(3X)
shmget: get shared memory segment. . . . . shmget(2)
brk, sbrk: change data segment space allocation. . . . . brk(2)
comm: select or reject lines common to two sorted files. . . . comm(1)
greek: select terminal filter. . . . . greek(1)
cut: cut out selected fields of each line of a file. . . . . cut(1)
dump: dump selected parts of an object file. . . . . dump(1)
semctl: semaphore control operations. . . . . semctl(2)
semop: semaphore operations. . . . . semop(2)
ipcrm: remove a message queue, semaphore set or shared memory id. . . . . ipcrm(1)
semget: get set of semaphores. . . . . semget(2)
semctl: semaphore control operations. . . . . semctl(2)
semget: get set of semaphores. . . . . semget(2)
semop: semaphore operations. . . . . semop(2)
kill: send a signal to a process or a group of processes. . . . kill(2)
mail, rmail: send mail to users or read mail. . . . . mail(1)
lp, cancel: send/cancel requests to an LP line printer. . . . . lp(1)
m400: MVME400 Dual RS-232C Serial Port Module. . . . . m400(7)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
setuid, setgid: set user and group IDs. . . . . setuid(2)
getgrent, getgrgid, getgrnam, setgrent, endgrent: obtain. . . . . getgrent(3C)
setjmp, longjmp: non-local goto. . . . . setjmp(3C)
setmnt: establish mount table. . . . . setmnt(1M)
setpgrp: set process group ID. . . . . setpgrp(2)
entry. getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get password file . . getpwent(3C)
profile: setting up an environment at login time. . . . . profile(4)
gettydefs: speed and terminal settings used by getty. . . . . gettydefs(4)
setuid, setgid: set user and group IDs. . . . . setuid(2)
entry. getutent, getutid, getutline, pututline, setutent, endutent, utmpname: access utmp file . . . . getut(3C)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
independent fashion.. sputl, sgetl: access long integer data in a machine . . . . . sputl(3X)
programming language. sh, rsh: shell, the standard/restricted command . . . . sh(1)
shmctl: shared memory control operations. . . . . shmctl(2)
ipcrm: remove a message queue, semaphore set or shared memory id. . . . . ipcrm(1)
shmat, shmdt: shared memory operations. . . . . shmop(2)
shmget: get shared memory segment. . . . . shmget(2)
system: issue a shell command from FORTRAN. . . . . system(3F)
system: issue a shell command. . . . . system(3S)
shl: shell layer manager. . . . . shl(1)
prtacct, runacct, shutacct, startup, turnacct: shell procedures for accounting. /prctmp, prdaily, . . acctsh(1M)
brc, bcheckrc, rc, powerfail: system initialization shell scripts. . . . . brc(1M)
language. sh, rsh: shell, the standard/restricted command programming . sh(1)
shl: shell layer manager. . . . . shl(1)
shmat, shmdt: shared memory operations. . . . . shmop(2)
shmctl: shared memory control operations. . . . . shmctl(2)
shmat, shmdt: shared memory operations. . . . . shmop(2)
shmget: get shared memory segment. . . . . shmget(2)
/nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct: shell procedures for/ . . . acctsh(1M)
shutdown: terminate all processing. . . . . shutdown(1M)
sdiff: side-by-side difference program. . . . . sdiff(1)
intrinsic function. sign, isign, dsign: FORTRAN transfer-of-sign . . . . . sign(3F)
login: sign on. . . . . login(1)
pause: suspend process until signal. . . . . pause(2)
signal: specify what to do upon receipt of a signal. . . . . signal(2)
specify FORTRAN action on receipt of a system signal. signal: . . . . . signal(3F)
system signal. signal: specify FORTRAN action on receipt of a . . . . signal(3F)
signal. signal: specify what to do upon receipt of a . . . . . signal(2)
kill: send a signal to a process or a group of processes. . . . . kill(2)
ssignal, gsignal: software signals. . . . . ssignal(3C)
lex: generate programs for simple lexical tasks. . . . . lex(1)
rand, srand: simple random-number generator. . . . . rand(3C)
trigonometric functions. sin, cos, tan, asin, acos, atan, atan2: . . . . . trig(3M)
sin, dsin, csin: FORTRAN sine intrinsic function. . . . sin(3F)
sin, dsin, csin: FORTRAN sine intrinsic function. . . . . sin(3F)
sinh, dsinh: FORTRAN hyperbolic sine intrinsic function. . . . . sinh(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)

function. sinh, dsinh: FORTRAN hyperbolic sine intrinsic . . . . sinh(3F)
size: print section sizes of common object files. . . . . size(1)
size: print section sizes of common object files. . . . . size(1)
sky: obtain ephemerides. . . . . sky(6)
sleep: suspend execution for an interval. . . . . sleep(1)
sleep: suspend execution for interval. . . . . sleep(3C)
ttyslot: find the slot in the utmp file of the current user. . . . . ttyslot(3C)
spline: interpolate smooth curve. . . . . spline(1G)
FORTRAN type/ int, ifix, idint, real, float, sngl, dble, cmplx, dcmplx, ichar, char: explicit . . . . ftype(3F)
sno: SNOBOL interpreter. . . . . sno(1)
sno: SNOBOL interpreter. . . . . sno(1)
pg: file perusal filter for soft-copy terminals. . . . . pg(1)
ssignal, gsignal: software signals. . . . . ssignal(3C)
sort: sort and/or merge files. . . . . sort(1)
qsort: quicker sort. . . . . qsort(3C)
sort: sort and/or merge files. . . . . sort(1)
tsort: topological sort. . . . . tsort(1)
comm: select or reject lines common to two sorted files. . . . . comm(1)
brk, sbrk: change data segment space allocation. . . . . brk(2)
ct: spawn getty to a remote terminal. . . . . ct(1C)
fspec: format specification in text files. . . . . fspec(4)
signal. signal: specify FORTRAN action on receipt of a system . . . signal(3F)
signal: specify what to do upon receipt of a signal. . . . . signal(2)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
gettydefs: speed and terminal settings used by getty. . . . . gettydefs(4)
errors. spell, hashmake, spellin, hashcheck: find spelling . . . spell(1)
spell, hashmake, spellin, hashcheck: find spellin, hashcheck: find spelling errors. . . . . spell(1)
spell, hashmake, spellin, hashcheck: find spelling errors. . . . . spell(1)
spline: interpolate smooth curve. . . . . spline(1G)
split: split a file into pieces. . . . . split(1)
csplit: context split. . . . . csplit(1)
fsplit: split f77, ratfor, or efl files. . . . . fsplit(1)
split: split a file into pieces. . . . . split(1)
uuclean: uucp spool directory clean-up. . . . . uuclean(1M)
lpadmin: configure the LP spooling system. . . . . lpadmin(1M)
printf, fprintf, sprintf: print formatted output. . . . . printf(3S)
independent fashion.. sputl, sgetl: access long integer data in a machine . . . sputl(3X)
function. sqrt, dsqrt, csqrt: FORTRAN square root intrinsic . . . sqrt(3F)
functions. exp, log, log10, pow, sqrt: exponential, logarithm, power, square root . . . . exp(3M)
log10, pow, sqrt: exponential, logarithm, power, square root functions. exp, log, ... exp(3M)
sqrt, dsqrt, csqrt: FORTRAN square root intrinsic function. . . . . sqrt(3F)
irand, srand, rand: random number generator. . . . . rand(3F)
rand, srand: simple random-number generator. . . . . rand(3C)
/erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ . . . . drand48(3C)
scanf, fscanf, sscanf: convert formatted input. . . . . scanf(3S)
ssignal, gsignal: software signals. . . . . ssignal(3C)
scc: C compiler for stand-alone programs. . . . . scc(1)
stdio: standard buffered input/output package. . . . . stdio(3S)
stdipc: standard interprocess communication package. . . . stdipc(3C)
sh, rsh: shell, the standard/restricted command programming language. . sh(1)
requests. lpsched, lpshut, lpmove: start/stop the LP request scheduler and move . . . . . lpsched(1M)
/prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct: shell procedures for accounting. . . . acctsh(1M)
stat: data returned by stat system call. . . . . stat(5)
stat, fstat: get file status. . . . . stat(2)
commands. stat: statistical network useful with graphical . . . . stat(1G)
stat: data returned by stat system call. . . . . stat(5)
stat: statistical network useful with graphical commands. . stat(1G)
ff: list filenames and statistics for a file system. . . . . ff(1M)
ustat: get file system statistics. . . . . ustat(2)
rjestat: RJE status report and interactive status console. . . . . rjestat(1C)
lpstat: print LP status information. . . . . lpstat(1)
ferror, feof, clearerr, fileno: stream status inquiries. . . . . ferror(3S)
uustat: uucp status inquiry and job control. . . . . uustat(1C)
ipcs: report inter-process communication facilities status. . . . . ipcs(1)
ps: report process status. . . . . ps(1)
rjestat: RJE status report and interactive status console. . . . . rjestat(1C)
stat, fstat: get file status. . . . . stat(2)
stdio: standard buffered input/output package. . . . . stdio(3S)
package. stdipc: standard interprocess communication . . . . . stdipc(3C)
stime: set time. . . . . stime(2)
wait: wait for child process to stop or terminate. . . . . wait(2)
strlen, strchr, strrchr, strpbrk, strspn, strcspn,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, . . . string(3C)
/strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok:/ . . . . string(3C)
strrchr, strpbrk, strspn,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, ... string(3C)
strspn, strcspn,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, ... string(3C)

strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations. /strcpy, ... string(3C)
sed: stream editor. . . . . sed(1)
fclose, fflush: close or flush a stream. . . . . fclose(3S)
fopen, freopen, fdopen: open a stream. . . . . fopen(3S)
rewind, ftell: reposition a file pointer in a stream. fseek, ... fseek(3S)
getchar, fgetc, getw: get character or word from stream. getc, ... getc(3S)
gets, fgets: get a string from a stream. . . . . gets(3S)
putchar, fputc, putw: put character or word on a stream. putc, ... putc(3S)
puts, fputs: put a string on a stream. . . . . puts(3S)
setbuf, setvbuf: assign buffering to a stream. . . . . setbuf(3S)
ferror, feof, clearerr, fileno: stream status inquiries. . . . . ferror(3S)
ungetc: push character back into input stream. . . . . ungetc(3S)
convert between long integer and base-64 ASCII string. a64l, l64a: . . . . . a64l(3C)
lge, lgt, lle, llt: string comparision intrinsic functions. . . . . strcmp(3F)
gmtime, asctime, tzset: convert date and time to string. ctime, localtime, ... ctime(3C)
ecvt, fcvt, gcvt: convert floating-point number to string. . . . . ecvt(3C)
gps: graphical primitive string, format of graphical files. . . . . gps(4)
gets, fgets: get a string from a stream. . . . . gets(3S)
len: return length of FORTRAN string. . . . . len(3F)
puts, fputs: put a string on a stream. . . . . puts(3S)
strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations. /strcpy, strncpy, strlen, ... string(3C)
strtod, atof: convert string to double-precision number. . . . . strtod(3C)
atof: convert ASCII string to floating-point number. . . . . atof(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
from an object file. strip: strip symbol and line number information . . . strip(1)
object file. strip: strip symbol and line number information from an . . strip(1)
strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn,/ . . . . string(3C)
strchr, strrchr, strpbrk, strspn, strcspn,/ strcat, strncat, strcmp, strncpy, strlen, . . . string(3C)
strpbrk, strspn, strcspn,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, ... string(3C)
strcspn,/ strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, ... string(3C)
strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/ /strcmp, . . . string(3C)
/strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/ . . . . string(3C)
/strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations. . . . . string(3C)
number. strtod, atof: convert string to double-precision . . . . strtod(3C)
strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations. /strcpy, strncpy, ... string(3C)
strtol, atol, atoi: convert string to integer. . . . . strtol(3C)
fuser: identify processes using a file or file structure. . . . . fuser(1M)
stty: set the options for a terminal. . . . . stty(1)
su: become superuser or another user. . . . . su(1)
intro: introduction to subroutines and libraries. . . . . intro(3)
plot: graphics interface subroutines. . . . . plot(3X)
paste: merge same lines of several files or subsequent lines of one file. . . . . paste(1)
index: return location of FORTRAN substring. . . . . index(3F)
v10graph - VME/10 graphics subsystem interface. . . . . v10graph(7)
sum: print checksum and block count of a file. . . . . sum(1)
du: summarize disk usage. . . . . du(1)
acctcms: command summary from per-process accounting records. . . . . acctcms(1M)
sync: update the super block. . . . . sync(1)
sync: update super-block. . . . . sync(2)
su: become superuser or another user. . . . . su(1)
ud: general driver for all disk units supported by the M68KVM21 disk controller. . . . . ud(7)
vm21: default general driver for all disk units supported by the M68KVM21 disk controller. . . . . vm21(7)
vm22: default general driver for all disk units supported by the M68KVM22 disk controller. . . . . vm22(7)
sleep: suspend execution for an interval. . . . . sleep(1)
sleep: suspend execution for interval. . . . . sleep(3C)
pause: suspend process until signal. . . . . pause(2)
swab: swap bytes. . . . . swab(3C)
swab: swap bytes. . . . . swab(3C)
sxt: pseudo-device driver. . . . . sxt(7)
file. strip: strip symbol and line number information from an object . strip(1)
ldgetname: retrieve symbol name for object file. . . . . ldgetname(3X)
ldtbindex: compute the index of a symbol table entry of a common object file. . . . . ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object file. . . . . ldtbread(3X)
syms: common object file symbol table format. . . . . syms(4)
ldtbseek: seek to the symbol table of a common object file. . . . . ldtbseek(3X)
sdb: symbolic debugger. . . . . sdb(1)
syms: common object file symbol table format. . . . . syms(4)
sync: update super-block. . . . . sync(2)
sync: update the super block. . . . . sync(1)
sysdef: system definition. . . . . sysdef(1M)
perror, errno, sys_errlist, sys_nerr: system error messages. . . . . perror(3C)
perror, errno, sys_errlist, sys_nerr: system error messages. . . . . perror(3C)
uuto, uupick: public UNIX System-to-UNIX System file copy. . . . . uuto(1C)
ldtbindex: compute the index of a symbol table entry of a common object file. . . . . ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object file. . . . . ldtbread(3X)

syms: common object file symbol table format. . . . . syms(4)
master: master device information table. . . . . master.dec(4)
mnttab: mounted file system table. . . . . mnttab(4)
ldtbseek: seek to the symbol table of a common object file. . . . . ldtbseek(3X)
toc: graphical table of contents routines. . . . . toc(1G)
setmnt: establish mount table. . . . . setmnt(1M)
hsearch, hcreate, hdestroy: manage hash search tables. . . . . hsearch(3C)
tabs: set tabs on a terminal. . . . . tabs(1)
tabs: set tabs on a terminal. . . . . tabs(1)
tail: deliver the last part of a file. . . . . tail(1)
functions. sin, cos, tan, asin, acos, atan, atan2: trigonometric . . . . . trig(3M)
tan, dtan: FORTRAN tangent intrinsic function. . . . tan(3F)
tan, dtan: FORTRAN tangent intrinsic function. . . . . tan(3F)
tanh, dtanh: FORTRAN hyperbolic tangent intrinsic function. . . . . tanh(3F)
tanh, dtanh: FORTRAN hyperbolic tangent intrinsic . . tanh(3F)
sinh, cosh, tanh: hyperbolic functions. . . . . sinh(3M)
tar: tape file archiver. . . . . tar(1)
frec: recover files from a backup tape. . . . . frec(1M)
filesave, tapesave: daily/weekly SYSTEM V/68 file system backup. filesave(1M)
tar: tape file archiver. . . . . tar(1)
lex: generate programs for simple lexical tasks. . . . . lex(1)
hpd, erase, hardcopy, tekset, td: graphical device routines and filters. . . . . gdev(1G)
tsearch, tfind, tdelete, twalk: manage binary search trees. . . . . tsearch(3C)
tee: pipe fitting. . . . . tee(1)
hpd, erase, hardcopy, tekset, td: graphical device routines and filters. . . . . gdev(1G)
4014: paginator for the Tektronix 4014 terminal. . . . . 4014(1)
init, telinit: process control initialization. . . . . init(1M)
tmpnam, tempnam: create a name for a temporary file. . . . . tmpnam(3S)
tmpfile: create a temporary file. . . . . tmpfile(3S)
tmpnam, tempnam: create a name for a temporary file. . . . . tmpnam(3S)
term: conventional names for terminals. . . . . term(5)
term: format of compiled term file. . . . . term(4)
term: format of compiled term file. . . . . term(4)
4014: paginator for the Tektronix 4014 terminal. . . . . 4014(1)
450: handle special functions of the DASI 450 terminal. . . . . 450(1)
terminfo: terminal capability data base. . . . . terminfo(4)
ct: spawn getty to a remote terminal. . . . . ct(1C)
ctermid: generate filename for terminal. . . . . ctermid(3S)
greek: select terminal filter. . . . . greek(1)
termio: general terminal interface. . . . . termio(7)
tty: controlling terminal interface. . . . . tty(7)
dial: establish an out-going terminal line connection. . . . . dial(3C)
gettydefs: speed and terminal settings used by getty. . . . . gettydefs(4)
stty: set the options for a terminal. . . . . stty(1)
tabs: set tabs on a terminal. . . . . tabs(1)
ttyname, isatty: find name of a terminal. . . . . ttyname(3C)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
300s: handle special functions of DASI 300 and 300s terminals. 300, ... 300(1)
handle special functions of HP 2640 and 2621-series terminals. hp: . . . . . hp(1)
tty: get the terminal's name. . . . . tty(1)
pg: file perusal filter for soft-copy terminals. . . . . pg(1)
term: conventional names for terminals. . . . . term(5)
kill: terminate a process. . . . . kill(1)
shutdown: terminate all processing. . . . . shutdown(1M)
abort: terminate FORTRAN program. . . . . abort(3F)
exit, _exit: terminate process. . . . . exit(2)
errstop: terminate the error-logging daemon. . . . . errstop(1M)
wait: wait for child process to stop or terminate. . . . . wait(2)
tic: terminfo compiler. . . . . tic(1M)
tput: query terminfo database. . . . . tput(1)
terminfo: terminal capability data base. . . . . terminfo(4)
termio: general terminal interface. . . . . termio(7)
test: condition evaluation command. . . . . test(1)
quiz: test your knowledge. . . . . quiz(6)
ed, red: text editor. . . . . ed(1)
ex: text editor. . . . . ex(1)
edit: text editor (variant of ex for casual users). . . . . edit(1)
newform: change the format of a text file. . . . . newform(1)
fspec: format specification in text files. . . . . fspec(4)
plock: lock process, text, or data in memory. . . . . plock(2)
tsearch, tfind, tdelete, twalk: manage binary search trees. . . . tsearch(3C)
tic: terminfo compiler. . . . . tic(1M)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
activity. timex: time a command; report process data and system . . . timex(1)
time: time a command. . . . . time(1)

mclock: return FORTRAN time accounting. . . . . mclock(3F)
at, batch: execute commands at a later time. . . . . at(1)
dcopy: copy file systems for optimal access time. . . . . dcopy(1M)
time: get time. . . . . time(2)
profil: execution time profile. . . . . profil(2)
profile: setting up an environment at login time. . . . . profile(4)
stime: set time. . . . . stime(2)
time: time a command. . . . . time(1)
time: get time. . . . . time(2)
localtime, gmtime, asctime, tzset: convert date and time to string. ctime, ... ctime(3C)
clock: report CPU time used. . . . . clock(3C)
times: get process and child process times. . . . . times(2)
touch: update access and modification times of a file. . . . . touch(1)
times: get process and child process times. . . . . times(2)
utime: set file access and modification times. . . . . utime(2)
system activity. timex: time a command; report process data and . . . timex(1)
tmpfile: create a temporary file. . . . . tmpfile(3S)
file. tmpnam, tempnam: create a name for a temporary . . tmpnam(3S)
toupper, tolower, _toupper, _tolower, toascii: translate characters. . . . . conv(3C)
toc: graphical table of contents routines. . . . . toc(1G)
popen, pclose: initiate pipe to/from a process. . . . . popen(3S)
toupper, tolower, _toupper, _tolower, toascii: translate characters. . . . . conv(3C)
characters. toupper, tolower, _toupper, _tolower, toascii: translate . . . . . conv(3C)
tsort: topological sort. . . . . tsort(1)
acctmerg: merge or add total accounting files. . . . . acctmerg(1M)
file. touch: update access and modification times of a . . . touch(1)
toupper, tolower, _toupper, _tolower, toascii: translate characters. . . . . conv(3C)
translate characters. toupper, tolower, _toupper, _tolower, toascii: . . . . . conv(3C)
tplot: graphics filters. . . . . tplot(1G)
tput: query terminfo database. . . . . tput(1)
tr: translate characters. . . . . tr(1)
ptrace: process trace. . . . . ptrace(2)
sign, isign, dsign: FORTRAN transfer-of-sign intrinsic function. . . . . sign(3F)
toupper, tolower, _toupper, _tolower, toascii: translate characters. . . . . conv(3C)
tr: translate characters. . . . . tr(1)
ftw: walk a file tree. . . . . ftw(3C)
tfind, tdelete, twalk: manage binary search trees. tsearch, ... tsearch(3C)
trenter: enter a trouble report. . . . . trenter(1M)
sin, cos, tan, asin, acos, atan, atan2: trigonometric functions. . . . . trig(3M)
trenter: enter a trouble report. . . . . trenter(1M)
true, false: provide truth values. . . . . true(1)
pdp11, u3b, vax, m68k: provide truth value about your processor type. . . . . machid(1)
true, false: provide truth values. . . . . true(1)
search trees. tsearch, tfind, tdelete, twalk: manage binary . . . . . tsearch(3C)
tsort: topological sort. . . . . tsort(1)
ttt, cubic: tic-tac-toe. . . . . ttt(6)
tty: controlling terminal interface. . . . . tty(7)
tty: get the terminal's name. . . . . tty(1)
ttyname, isatty: find name of a terminal. . . . . ttyname(3C)
current user. ttyslot: find the slot in the utmp file of the . . . . . ttyslot(3C)
prdaily, prtacct, runacct, shutacct, startup, turnacct: shell procedures for accounting. /prctmp, . . acctsh(1M)
tsearch, tfind, tdelete, twalk: manage binary search trees. . . . . tsearch(3C)
dble, cmplx, dcmplx, ichar, char: explicit FORTRAN type conversion. /ifix, idint, real, float, sngl, ... ftype(3F)
file: determine file type. . . . . file(1)
vax, m68k: provide truth value about your processor type. pdp11, u3b, ... machid(1)
getty: set terminal type, modes, speed, and line discipline. . . . . getty(1M)
types: primitive system data types. . . . . types(5)
types: primitive system data types. . . . . types(5)
ctime, localtime, gmtime, asctime, tzset: convert date and time to string. . . . . ctime(3C)
processor type. pdp11, u3b, vax, m68k: provide truth value about your . . . machid(1)
the M68KVM21 disk controller ud: general driver for all disk units supported by . . . ud(7)
getpw: get name from UID. . . . . getpw(3C)
ulimit: get and set user limits. . . . . ulimit(2)
umask: set and get file creation mask. . . . . umask(2)
umask: set file-creation mode mask. . . . . umask(1)
mount, umount: mount and dismount file system. . . . . mount(1M)
umount: unmount a file system. . . . . umount(2)
uname: get name of current operating system. . . . . uname(2)
uname: print name of current UNIX System. . . . . uname(1)
unget: undo a previous get of an SCCS file. . . . . unget(1)
unget: undo a previous get of an SCCS file. . . . . unget(1)
ungetc: push character back into input stream. . . . . ungetc(3S)
/jrand48, srand48, seed48, lcong48: generate uniformly distributed pseudo-random numbers. . . . . drand48(3C)
uniq: report repeated lines in a file. . . . . uniq(1)
mktemp: make a unique filename. . . . . mktemp(3C)

	units: conversion program.	units(1)
ud: general driver for all disk	units supported by the M68KVM21 disk controller.	ud(7)
vm21: default general driver for all disk	units supported by the M68KVM21 disk controller.	vm21(7)
vm22: default general driver for all disk	units supported by the M68KVM22 disk controller.	vm22(7)
cm16: 16Mb Cartridge Module Drive for	Universal Disk Driver.	cm16(7)
cm80: 80Mb Cartridge Module Drive for	Universal Disk Driver.	cm80(7)
fl8: 8-inch Floppy Disk Drive for	Universal Disk Driver.	fl8(7)
lrk25: 25Mb LARK Module Drive for	Universal Disk Driver.	lrk25(7)
uux:	UNIX-to-UNIX system command execution.	uux(1C)
link,	unlink: exercise link and unlink system calls.	link(1M)
	unlink: remove directory entry.	unlink(2)
link, unlink: exercise link and	unlink system calls.	link(1M)
umount:	unmount a file system.	umount(2)
pack, pcat,	unpack: compress and expand files.	pack(1)
touch:	update access and modification times of a file.	touch(1)
make: maintain,	update, and regenerate groups of programs.	make(1)
lsearch, lfind: linear search and	update.	lsearch(3C)
sync:	update super-block.	sync(2)
sync:	update the super block.	sync(1)
du: summarize disk	usage.	du(1)
stat: statistical network	useful with graphical commands.	stat(1G)
id: print	user and group IDs and names.	id(1)
setuid, setgid: set	user and group IDs.	setuid(2)
crontab:	user crontab file.	crontab(1)
cuserid: get character login name of the	user.	cuserid(3S)
group/ getuid, geteuid, getgid, getegid: get real	user, effective user, real group, and effective	getuid(2)
environ:	user environment.	environ(5)
diskusg: generate disk accounting data by	user ID.	diskusg(1M)
ulimit: get and set	user limits.	ulimit(2)
logname: return login name of	user.	logname(3X)
geteuid, getgid, getegid: get real user, effective	user, real group, and effective group IDs. getuid,	getuid(2)
su: become superuser or another	user.	su(1)
find the slot in the utmp file of the current	user. ttyslot:	ttyslot(3C)
write: write to another	user.	write(1)
edit: text editor (variant of ex for casual	users).	edit(1)
mail, rmail: send mail to	users or read mail.	mail(1)
wall: write to all	users.	wall(1M)
fuser: identify processes	using a file or file structure.	fuser(1M)
	ustat: get file system statistics.	ustat(2)
gutil: graphical	utilities.	gutil(1G)
	utime: set file access and modification times.	utime(2)
utmp, wtmp:	utmp and wtmp entry formats.	utmp(4)
pututline, setutent, endutent, utmpname: access	utmp file entry. getutent, getutid, getutline,	getut(3C)
ttyslot: find the slot in the	utmp file of the current user.	ttyslot(3C)
	utmp, wtmp: utmp and wtmp entry formats.	utmp(4)
getutid, getutline, pututline, setutent, endutent,	utmpname: access utmp file entry. getutent,	getut(3C)
	uuclean: uucp spool directory clean-up.	uuclean(1M)
uusub: monitor	uucp network.	uusub(1M)
uuclean:	uucp spool directory clean-up.	uuclean(1M)
uustat:	uucp status inquiry and job control.	uustat(1C)
	uucp, uulog, uuname: unix to unix copy.	uucp(1C)
uucp,	uulog, uuname: unix to unix copy.	uucp(1C)
uucp, uulog,	uuname: unix to unix copy.	uucp(1C)
copy. uuto,	uupick: public UNIX System-to-UNIX System file	uuto(1C)
	uustat: uucp status inquiry and job control.	uustat(1C)
	uusub: monitor uucp network.	uusub(1M)
file copy.	uuto, uupick: public UNIX System-to-UNIX System	uuto(1C)
	uux: UNIX-to-UNIX system command execution.	uux(1C)
	v10graph - VME/10 graphics subsystem interface.	v10graph(7)
	val: validate SCCS file.	val(1)
val:	validate SCCS file.	val(1)
pdp11, u3b, vax, m68k: provide truth	value about your processor type.	machid(1)
abs: return integer absolute	value.	abs(3C)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute	value.	abs(3F)
getenv: return	value for environment name.	getenv(3C)
fmod, fabs: floor, ceiling, remainder, absolute	value functions. floor, ceil,	floor(3M)
putenv: change or add	value to environment.	putenv(3C)
	values: machine-dependent values.	values(5)
true, false: provide truth	values.	true(1)
values: machine-dependent	values.	values(5)
vfprintf, vsprintf: print formatted output of a	varargs argument list. vprintf,	vprintf(3S)
	varargs: handle variable argument list.	varargs(5)
varargs: handle	variable argument list.	varargs(5)
getenv: return FORTRAN environment	variable.	getenv(3F)
edit: text editor	(variant of ex for casual users).	edit(1)

type. pdp11, u3b, vax, m68k: provide truth value about your processor . machid(1)
fscv: convert files between M68000 and VAX-11/780 processors. . . . . fscv(1M)
vc: version control. . . . . vc(1)
getopt: get option letter from argument vector. . . . . getopt(3C)
assert: verify program assertion. . . . . assert(3X)
vc: version control. . . . . vc(1)
get: get a version of an SCCS file. . . . . get(1)
sccsdiff: compare two versions of an SCCS file. . . . . sccsdiff(1)
varargs argument list. vprintf, vfprintf, vsprintf: print formatted output of a . . . . vprintf(3S)
on ex. vi: screen-oriented (visual) display editor based . . . . vi(1)
vi: screen-oriented (visual) display editor based on ex. . . . . vi(1)
supported by the M68KVM21 disk controller. vm21: default general driver for all disk units . . . . vm21(7)
cmd16: 16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd16(7)
cmd80: 80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. . . . . cmd80(7)
lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark25(7)
lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark8(7)
sa800fl21: 8-inch Floppy Disk Drive for VM21 Driver. . . . . sa800fl21(7)
supported by the M68KVM22 disk controller. vm22: default general driver for all disk units . . . . vm22(7)
vm22fmt: format disks on the VM22 disk controller. . . . . vm22fmt(1M)
16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. cmd16: . . . . . cmd16(7)
80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver. cmd80: . . . . . cmd80(7)
lark25: 50Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark25(7)
lark8: 16Mb LARK Module Drive for VM21 Driver and VM22 Driver. . . . . lark8(7)
sa400fl22: 5¼-inch Floppy Disk Drive for VM22 Driver. . . . . sa400fl22(7)
sa800fl22: 8-inch Floppy Disk Drive for VM22 Driver. . . . . sa800fl22(7)
vm22fmt: format disks on the VM22 disk controller. . vm22fmt(1M)
bootstrap operating procedure for system restart on VME/10. bo.vme: . . . . . bo.vme(8)
v10graph - VME/10 graphics subsystem interface. . . . . v10graph(7)
wffmt: format floppies for the VME/10 processor. . . . . wffmt(1M)
checking. volcopy, labelit: copy file systems with label . . . . . volcopy(1M)
file system: format of system volume. . . . . fs(4)
of a varargs argument list. vprintf, vfprintf, vsprintf: print formatted output . . vprintf(3S)
argument list. vprintf, vfprintf, vsprintf: print formatted output of a varargs . . . . . vprintf(3S)
wait: await completion of process. . . . . wait(1)
wait: wait for child process to stop or terminate. . . . . wait(2)
wait: wait for child process to stop or terminate. . . . wait(2)
ftw: walk a file tree. . . . . ftw(3C)
wall: write to all users. . . . . wall(1M)
wc: word count. . . . . wc(1)
wd15: 15Mb Winchester Disk Drive. . . . . wd15(7)
wd40: 40Mb Winchester Disk Drive. . . . . wd40(7)
wffmt: format floppies for the VME/10 processor. . . . wffmt(1M)
what: identify SCCS files. . . . . what(1)
signal: specify what to do upon receipt of a signal. . . . . signal(2)
crash: what to do when the system crashes. . . . . crash.macs(8)
whodo: who is doing what. . . . . whodo(1M)
who: who is on the system. . . . . who(1)
who: who is on the system. . . . . who(1)
whodo: who is doing what. . . . . whodo(1M)
wd15: 15Mb Winchester Disk Drive. . . . . wd15(7)
wd40: 40Mb Winchester Disk Drive. . . . . wd40(7)
sa400flwd: 5¼-inch Floppy Disk Drive for the Winchester Disk Driver. . . . . sa400flwd(7)
cd: change working directory. . . . . cd(1)
chdir: change working directory. . . . . chdir(2)
getcwd: get pathname of current working directory. . . . . getcwd(3C)
pwd: working directory name. . . . . pwd(1)
write: write on a file. . . . . write(2)
putpwent: write password file entry. . . . . putpwent(3C)
wall: write to all users. . . . . wall(1M)
write: write to another user. . . . . write(1)
write: write on a file. . . . . write(2)
write: write to another user. . . . . write(1)
open: open for reading or writing. . . . . open(2)
utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)
utmp, wtmp: utmp and wtmp entry formats. . . . . utmp(4)
fwtmp, wtmpfix: manipulate connect accounting records. . . . fwtmp(1M)
wump: the game of hunt-the-wumpus. . . . . wump(6)
command. xargs: construct argument list(s) and execute . . . . . xargs(1)
functions. and, or, xor, not, lshift, rshift: FORTRAN bitwise Boolean . . . bool(3F)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
yacc: yet another compiler-compiler. . . . . yacc(1)
j0, j1, jn, y0, y1, yn: Bessel functions. . . . . bessel(3M)
abs, iabs, dabs, cabs, zabs: FORTRAN absolute value. . . . . abs(3F)

NAME

intro — introduction to system maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *SYSTEM V/68 User's Manual*. References to other manual entries not of the form *name*(1M), *name*(7) or *name*(8) refer to entries of that manual.

COMMAND SYNTAX

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

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

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

SEE ALSO

getopt(1), getopt(3C).
SYSTEM V/68 User's Manual.
SYSTEM V/68 Administrator's Guide.

DIAGNOSTICS

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

BUGS

Some commands do not adhere to the syntax mentioned above.

NAME

accept, reject — allow/prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [—r[reason]] destinations

DESCRIPTION

Accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with *reject*.

—r[*reason*] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next —r option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the —r option is not present or the —r option is given without a *reason*, then a default *reason* will be used.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1), lpadmin(1M), lpsched(1M), lpstat(1).

NAME

acctdisk, acctdusg, accton, acctwtmp — overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**—u** file] [**—p** file]

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. *Acctsh*(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into **/usr/adm/utmp**, as described in *utmp*(4). The programs described in *acctcon*(1M) convert this file into session and charging records, which are then summarized by *acctmerg*(1M).

Process accounting is performed by the SYSTEM V/68 kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

Process accounting and connect time accounting (or any accounting records in the format described in *acct*(4)) can be merged and summarized into total accounting records by *acctmerg* (see **tacct** format in *acct*(4)). *Prtacct* (see *acctsh*(1M)) is used to format any or all accounting records.

Acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

Acctdusg reads its standard input (usually from **find / —print**) and computes disk resource consumption (including indirect blocks) by login. If **—u** is given, records consisting of those filenames for which *acctdusg* charges no one are placed in *file* (a potential source for finding users trying to avoid disk charges). If **—p** is given, *file* is the name of the password file. This option is not needed if the password file is **/etc/passwd** (refer to *diskusg*(1M) for more details).

Accton alone turns process accounting off. If *file* is given, it must be the name of an existing file, to which the kernel appends process accounting records (see *acct*(2) and *acct*(4)).

Acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned (see *utmp*(4)). *Reason* must be a string of 11 or less characters, numbers, **$**, or spaces. For example, the following is a suggestion for use in shutdown procedures:

```
acctwtmp "file save" >> /etc/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

"Accounting" in the *SYSTEM V/68 Administrator's Guide*.

NAME

acctcms — command summary from per-process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [options] files

DESCRIPTION

Acctcms reads one or more *files*, normally in the form described in *acct*(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The *options* are:

- **—a** Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.
- **—c** Sort by total CPU time, rather than total kcore-minutes.
- **—j** Combine all commands invoked only once under "***other".
- **—n** Sort by number of command invocations.
- **—s** Any file names encountered hereafter are already in internal summary format.
- **—t** Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (i.e., UNIX System V) style *acctcms* internal summary format records.

The following options may be used only with the **—a** option:

- **—p** Output a prime-time-only command summary.
- **—o** Output a non-prime (offshift) time only command summary.

When **—p** and **—o** are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes, which will be split into prime and non-prime.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms file ... >today
cp total previoustotal
acctcms —s today previoustotal >total
acctcms —a —s today
```

SEE ALSO

acct(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

CAUTION

The current *acctcms* can still read, process, and generate command accounting records compatible with previous SYSTEM V/68 releases if the **—t** option is used. Note, however, that internal summary records generated with the **—t** option (or by the SYSTEM V/68 Release 1 version) are not compatible with those created by the Release 2 *acctcms* without this option.

NAME

acctcon1, acctcon2 — connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

Acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

- **—p** Print input only, showing line name, login name, and time (in both numeric and date/time formats).
- **—t** *Acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **—t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.
- **—l** *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of *login*(1) and termination of the login shell generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).
- **—o** *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in *acct*(4)).

EXAMPLES

These commands are typically used as shown below. The file **ctmp** is created only for the use of *acctprc*(1M) commands:

```
acctcon1 —t —l lineuse —o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

BUGS

The line usage report is confused by date changes. Use *wtmpfix* (see *fwtmp*(1M)) to correct this situation.

NAME

acctmerg — merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [options] [file] . . .

DESCRIPTION

Acctmerg reads its standard input and up to nine additional files in the **tacct** format (see *acct*(4)) or in an ASCII version. It merges these inputs by adding records whose keys (normally user ID and name) are identical and expects the inputs to be sorted on those keys. *Options* are:

- **—a** Produce output in ASCII version of **tacct.**
- **—i** Input files are in ASCII version of **tacct.**
- **—p** Print input with no processing.
- **—t** Produce a single record that totals all input.
- **—u** Summarize by user ID, rather than user ID and name.
- **—v** Produce output in verbose ASCII format, with more precise notation for floating point numbers.

The following sequence is useful for making repairs to any file kept in this format:

```
acctmerg —v <file1 >file2
```

Perform edit on *file2*, then enter:

```
acctmerg —i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

acctprc1, acctprc2 — process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in 64-byte units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

Acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

BUGS

Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from *cron*(1M), for example. More precise conversion can be done by faking login sessions on the console via the *acctwtmp* program in *acct*(1M).

CAUTION

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, on a PDP-11/70 this measure would be in 64-byte units; on a VAX11/780, EXORmacs, or VME/10 it would be in 512-byte units.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct — shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [—o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [—l] [—c] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | off | switch

DESCRIPTION

Chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

Ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* is invoked with argument *switch*. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* automatically turns off the collection of process accounting records via the **off** argument to *turnacct*. When at least this number of blocks is restored, accounting is reactivated. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

Dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/checklist**. If the —o flag is used, it will do a slower version of disk accounting by login directory. *Files* specify the one or more filesystem names where disk accounting will be done. If *files* are used, disk accounting will be done on these filesystems only. If the —o flag is used, *files* should be mount points of mounted filesystems. If omitted, they should be the special filenames of mountable filesystems.

Lastlogin is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

Monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01—12). This default is useful if *monacct* is to be executed via *cron*(1M) on the first day of each month. *Monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

Nulladm creates *file* with mode 664 and insures owner and group are **adm**. It is called by various accounting shell procedures.

Prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon1* (see *acctcon*(1M)).

Prdaily is invoked by *runacct* to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd*, where *mmdd* is the month and day of the

report. The current daily accounting reports may be printed by typing *prdaily*. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The —**l** flag prints a report of exceptional usage by login id for the specified date. Previous daily reports are cleaned up and, therefore, inaccessible after each invocation of *monacct*. The —**c** flag prints a report of exceptional resource usage by command and may be used on current day's accounting data only.

Prtacct can be used to format and print any total accounting (**tacct**) file.

Runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see *runacct*(1M).

Shutacct should be invoked during a system shutdown (usually in **/etc/shutdown**) to turn process accounting off and append a "reason" record to **/etc/wtmp**.

Startup should be called by **/etc/rc** to turn the accounting on whenever the system is brought up.

Turnacct is an interface to *accton* (see *acct*(1M)) to turn process accounting **on** or **off**. The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with **1** and incrementing by one for each additional **pacct** file), then turns accounting back on again. This procedure is called by *ckpacct* and thus can be taken care of by the *cron* and used to keep **pacct** to a reasonable size.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if **pacct** gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/lib/acct/ptelus.awk	contains the limits for exceptional usage by login id
/usr/lib/acct/ptecms.awk	contains the limits for exceptional usage by command name
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

bcopy — interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

Bcopy copies from and to files starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to)

offset: (you provide the starting "to" block number)

from: (you name the file or device to be copied from)

offset: (you provide the starting "from" block number)

count: (you reply with the number of blocks to be copied)

After **count** is exhausted, the **from** question is repeated (providing the ability to concatenate blocks at the **to+offset+count** location). If **from** is answered with a carriage return, everything starts over.

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

brc, bcheckrc, rc, powerfail — system initialization shell scripts

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

DESCRIPTION

Except for *powerfail*, these shell procedures are executed via entries in **/etc/inittab** by *init*(1M) when the system is changed out of single-user mode. *Powerfail* is executed whenever a system power failure is detected.

The *brc* procedure clears the mounted file system table **/etc/mnttab** (see *mnttab*(4)) and loads any programmable microprocessors with their appropriate scripts.

The *bcheckrc* procedure performs all the necessary consistency checks to prepare the system to change into multi-user mode. It will prompt to set the system date and to check the file systems with *fsck*(1M).

The *rc* procedure starts all system daemons before the terminal lines are enabled for multi-user mode. In addition, file systems are mounted and accounting, error logging, and system activity logging are activated in this procedure.

The *powerfail* procedure is invoked when the system detects a power failure condition. Its chief duty is to reload any programmable microprocessors with their appropriate scripts, if suitable. It also logs the fact that a power failure occurred.

These shell procedures, in particular *rc*, may be used for several run-level states. The *who*(1) command may be used to get the run-level information.

SEE ALSO

fsck(1M), init(1M), shutdown(1M), who(1), inittab(4), mnttab(4).

NAME

checkall — faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as an example:

```
# check the root file system by itself
fsck /dev/rdsk/cntrlr_0s0

# dual fsck of drives 0 and 1
dfsck /dev/rdsk/cntrlr_0s[12345] — /dev/rdsk/cntrlr_1s0
```

If **/dev/rdsk/*cntrlr*_1s0** is 320K blocks and **/dev/rdsk/*cntrlr*_0s[12345]** are each 65K or less, a previous sequential *fsck* took 19 minutes. The *checkall* procedure takes 11 minutes.

Dfsck is a program that permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Due to the file system load balancing required for dual checking, the *dfsck* command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided as follows:

```
dfsck file_systems_on_drive_0 — file_systems_on_drive_1
dfsck file_systems_on_drive_2 — file_systems_on_drive_3
  . . .
```

A three-drive system can be handled, as shown in the following example (assumes two large file systems per drive):

dfsck /dev/dsk/*cntrlr*_3s1 /dev/dsk/*cntrlr*_0s[14] — /dev/dsk/*cntrlr*_1s[14] /dev/dsk/*cntrlr*_3s4

Note that the first file system on drive 3 is first in the *filesystems1* list and is last in the *filesystems2* list, assuring that references to that drive will not overlap at execution time.

WARNINGS

1. Do not use *dfsck* to check the **root** file system.
2. On a check that requires a scratch file (refer to the **—t** option of *dfsck*), be careful not to use the same temporary file for the two groups (this is sure to scramble the file systems).
3. The *dfsck* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).

"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

AME

chk – check a file system

NOPSIS

/mot/bin/chk [*disk*] [*-y* *-n*]

ESCRIPTION

Chk(1M) checks a file system using *fsck*(1M). The argument *disk* and the *permissions* file are used to determine the device to check and whether it should be checked '-n' (NO WRITE), as when *Write* permission has not been granted. This device will be the first match of *disk* and the *real device* or *alias* entries in the *permissions* file. If the *disk* argument is not given then the first *alias* of **default** in the *permissions* file will be used.

Chk actually uses the raw version of the listed *real device* (by prepending an 'r' to the name).

The *-y* (always answer yes) and *-n* (always answer no; i.e. NO WRITE) options are passed to *fsck* if present. They are mutually exclusive.

LES

/etc/fsck
/mot/bin/fs
/mot/etc/perms permissions file

E ALSO

fs(4), fsck(1M), perms(4)
SYSTEM V/68 Administrator's Guide.

NAME

chroot — change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The given command is executed relative to the new root. The meaning of any initial slashes (/) in pathnames is changed for a command and any of its children to *newroot*. Furthermore, the initial working directory is *newroot*.

Notice that:

chroot newroot command >x

will create the file **x** relative to the original root, not the new one.

This command is restricted to the superuser.

The new root pathname is always relative to the current root; even if a *chroot* is currently in effect, the *newroot* argument is relative to the current root of the running process.

SEE ALSO

chdir(2).

BUGS

One should exercise extreme caution when referencing special files in the new root file system.

NAME

clri — clear inode

SYNOPSIS

/etc/clri file-system i-number ...

DESCRIPTION

Clri writes zeros on the 64 bytes occupied by the inode numbered *i-number*. *File-system* must be a special filename referring to a device containing a file system. After *clri* is executed, any blocks in the affected file will show up as missing in an *fsck*(1M) of the *file-system*. This command should only be used in emergencies, and extreme care should be exercised.

Read and write permission is required on the specified *file-system* device. The inode becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to delete an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point, removing the old entry will destroy the new file. The new entry will again point to an unallocated inode; therefore, the whole cycle is likely to be repeated again and again.

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M), fs(4).

BUGS

If the file is open, *clri* is likely to be ineffective.

NAME

config.68 — configure SYSTEM V/68

SYNOPSIS

/etc/config [**—t**] [**—v** file] [**—l** file] [**—c** file] [**—m** file] dfile

DESCRIPTION

Config is a program that takes a description of SYSTEM V/68 and generates three files. One file provides information regarding the interface between the hardware and device handlers (**low.s**). A second file is a C program defining the configuration tables for the various devices on the system (**conf.c**). A third file contains exception vector assignments (**m68kvec.s**).

The **—v** option specifies the name of the exception vector file; **m68kvec.s** is the default name.

The **—l** option specifies the name of the hardware interface file; **low.s** is the default name.

The **—c** option specifies the name of the configuration table file; **conf.c** is the default name.

The **—m** option specifies the name of the file that contains all the information regarding supported devices; **/etc/master** is the default name. This file is supplied with SYSTEM V/68 and should not be modified unless the user fully understands its construction.

The **—t** option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply **dfile**; it must contain device information for the user's system. This file is divided into three parts. The first part contains physical device specifications. The second part contains system-dependent information. The third part contains microprocessor-specific information. The first two parts are required, the third part is optional. The format and contents of the **dfile** are provided in *dfile*(4) in the *SYSTEM V/68 User's Manual.* To obtain an example configuration, the user can run the *sysdef*(1M) utility.

FILES

/etc/master	default input master device table
m68kvec.s	default output exception vector file for m68k
low.s	default output hardware interface file for m68k
conf.c	default output configuration table file

SEE ALSO

sysdef(1M), dfile(4), master(4).
"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

DIAGNOSTICS

Diagnostics are routed to the standard output and are self-explanatory.

NAME

cpset — install object files in binary directories

SYNOPSIS

cpset [—o] object directory [mode owner group]

DESCRIPTION

Cpset is used to install the specified *object* file in the given *directory*. The *mode*, *owner*, and *group* of the destination file may be specified on the command line. If this data is omitted, two results are possible:

1. If the user of *cpset* has administrative permissions (i.e., the user's numerical ID is less than 100), the following defaults are provided:
   mode — 0755
   owner — bin
   group — bin

2. If the user is not an administrator, the default owner and group of the destination file will be that of the invoker.

An optional argument of —o will force *cpset* to move *object* to **OLD***object* in the destination directory before installing the new object.

For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo

All the examples above have the same effect (assuming that the user is administrator). The file **echo** will be copied into **/bin** and will be given **0755**, **bin**, and **bin** as the mode, owner, and group, respectively.

Cpset utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of pathnames separated by spaces or tabs. The first name is the "official" destination (e.g., **/bin/echo**). The second name is the new destination. For example, if **echo** is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

/bin/echo /usr/bin/echo

When the actual installation is performed, *cpset* verifies that the "old" pathname does not exist. If a file exists at that location, *cpset* issues a warning and continues. This file does not exist on a distribution tape; it is used by sites to track local command movement. The procedures used to build the source will be responsible for defining the "official" locations of the source.

Cross Generation

The environment variable **ROOT** will be used to locate the destination file (in the form **$ROOT/usr/src/destinations**). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

install(1M), make(1), mk(8).

NAME

crash — examine system images

SYNOPSIS

/etc/crash [system] [namelist]

DESCRIPTION

Crash is an interactive utility for examining an operating system core image. It has facilities for interpreting and formatting the various control structures in the system and certain miscellaneous functions that are useful when perusing a dump.

The arguments to *crash* are the file name where the *system* image can be found and a *namelist* file to be used for symbol values.

The default values are **/dev/mem** and **/unix**; hence, *crash* with no arguments can be used to examine an active system. If a *system* image file is given, it is assumed to be a system core dump and the default process is set to be that of the process running at the time of the crash. This is determined by a value stored in a fixed location by the dump mechanism.

COMMANDS

Input to *crash* is typically of the form:

command [*options*] [*structures to be printed*]

When allowed, *options* modifies the format of the printout. If no specific structure elements are specified, all valid entries are used. As an example, **proc — 12 15 3** would print process table slots 12, 15 and 3 in a long format, while **proc** would print the entire process table in standard format.

In general, those commands that perform I/O with addresses assume hexadecimal on 32-bit machines and octal on 16-bit machines.

The current list of commands includes:

user [list of process table entries]

Aliases: **uarea, u_area, u.**

Print the user structure of the named process as determined by the information contained in the process table entry. If no entry number is given, the information of the last executing process is printed. Swapped processes produce an error message.

trace [—**r**] [list of process table entries]

Aliases: **t.**

Generate a kernel stack trace of the current process. If the —**r** option is used, the trace begins at the saved stack frame pointer in **kfp**. Otherwise the trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack. If no entry number is given, the information on the last executing process is printed.

kfp [stack frame pointer]

Aliases: **fp.**

Print the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given, or set the frame pointer to the supplied value.

stack [list of process table entries]

Aliases: **stk, s, kernel, k.**

Format a dump of the kernel stack of a process. The addresses shown are virtual system data addresses rather than true physical locations. If no entry number is given, the information on the last executing process is printed.

proc [—[**r**]] [list of process table entries]

Aliases: **ps, p.**

Format the process table. The —**r** option causes only runnable processes to be printed.

The — alone generates a longer listing.

inode [—] [list of inode table entries]
Aliases: **ino, i.**
Format the inode table. The — option also prints the inode data block addresses.

file [list of file table entries]
Aliases: **files, f.**
Format the file table.

mount [list of mount table entries]
Aliases: **mnt, m.**
Format the mount table.

text [list of text table entries]
Aliases: **txt, x.**
Format the text table.

tty [**type**] [—] [list of tty entries]
Aliases: **term, acia, m400.**
Print the tty structures. The *type* argument determines which structure is used (such as **acia**). No default *type* is provided; however, once specified, the last *type* is remembered. The — option prints the *stty*(1) parameters for the given line.

stat Print certain statistics found in the dump. These include the panic string (if a panic occurred), time of crash, system name, and the registers saved in low memory by the dump mechanism.

var Aliases: **tunables, tunable, tune, v.**
Print the tunable system parameters.

buf [list of buffer headers]
Aliases: **hdr, bufhdr.**
Format the system buffer headers.

buffer [format] [list of buffers]
Alias: **b.**
Print the data in a system buffer according to *format*. If *format* is omitted, the previous *format* is used. Valid formats include **decimal, octal, hex, character, byte, directory, inode,** and **write.** The last creates a file in the current directory (see "FILES") containing the buffer data.

callout
Aliases: **calls, call, c, timeout, time, tout.**
Print all entries in the callout table.

map [list of map names]
Format the named system map structures.

nm [list of symbols]
Print symbol value and type as found in the *namelist* file.

ts [list of text addresses]
Find the closest text symbols to the given addresses.

ds [list of data addresses]
Find the closest data symbols to the given addresses.

od [symbol name or address] [count] [format]
Aliases: **dump, rd.**
Dump *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **octal, longoct, decimal, longdec, character, hex,** or **byte.**

mmu [process slot number(s)]

The **mmu** command displays the contents of each entry in the **mmu_table[ ]**. When the optional argument is given, the command will display the **mmu_table** entry associated with the given process slot number. Multiple process slot numbers should be separated with blanks.

! Escape to shell.

q Exit from *crash*.

? Print synopsis of commands.

ALIASES

There are built-in aliases for many of the *formats* as well as those listed for the commands. Some of them are:

byte	b.
character	char, c.
decimal	dec, e.
directory	direct, dir, d.
hexadecimal	hexadec, hex, h, x.
inode	ino , i.
longdec	ld, D.
longoct	lo, O.
octal	oct, o.
write	w.

FILES

/usr/include/sys/*.h	header files for table and structure info
/dev/mem	default system image file
/unix	default namelist file
buf.#	files created containing buffer data

SEE ALSO

mount(1M), nm(1), ps(1), sh(1), stty(1), crash.macs(8).

BUGS

Most flags are abbreviated and have little meaning to the uninitiated user. A source listing of the system header files at hand would be most useful while using *crash.*

Stack tracing of the current process on a running system doesn't work.

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in *crontab* files; users can submit their own *crontab* file via the *crontab*(1) command. Commands that are to be executed only once may be submitted via the *at* command. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file **/etc/rc** .

Cron only examines *crontab* files and *at* command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/log	accounting information
/usr/spool/cron	spool area

SEE ALSO

at(1), crontab(1), sh(1).
"Administrative Guidelines" in the *SYSTEM V/68 Administrator's Guide.*

DIAGNOSTICS

A history of all actions taken by *cron* are recorded in **/usr/lib/cron/log.**

AME

crc - a tool to generate cyclic redundancy checksums (crc) of files

YNOPSIS

crc [-frcld] - | file_list

ESCRIPTION

The **crc** shell command utility is a versatile tool for use in generating 16-bit crc values of an input stream. The input stream can consist either of data or of names of files to be checked. There are four different display options available.

If the file to be checked is an object file, **crc** will ignore the compiler-generated time stamps embedded in the file.

The various options are defined as follows:

-f Selects *file mode* operation. The input stream is interpreted as a list of the names of the files to be processed rather than as the data itself.

-r Selects a *raw mode* of operation. This option is used mainly to determine if two versions of an executable file are *exactly* the same. This switch causes **crc** to include the compiler-generated time stamps in the *coff* file image when computing the crc.

-c Changes the output to include the byte count of each file processed.

-d Adds the time of the file's modification to the output.

-l Computes the crc in decimal for *each line of the input file* rather than for the whole file itself. Use of this option overrides all others!

Note that the first four options can be used in any combination.

There are three general forms of output. The first form is produced without the **-c** option:

$nnnn for *filename (time stamp)*

where *$nnnn* is the 16-bit checksum in hexadecimal representation; and *time stamp* is the time of the file's modification (displayed if the **-d** option is selected). The fields are separated by space (20h) characters.

The second output form is generated when the **-c** option is selected:

$nnnn length time stamp filename

where *length* is the true size of the file, regardless of whether or not *raw mode* (**-r**) is selected; and *time stamp* is the time of the file's modification (displayed if the **-d** option is selected). All fields of this second form are delimited by tab (\t) characters.

The third form of output is produced by the *line mode* option (**-l**). It replaces each line of input with its corresponding crc in the form *nnnn*.

DIAGNOSTICS

crc: bad option letter. - an invalid option letter was specified.

crc: argument count. - at least one file name (or '-') must be provided.

crc: can't open file for reading. - file cannot be opened for some reason.

crc: can't read file. - input file cannot be read for some reason.

EXAMPLES

Suppose that a **touch *; ls -log** command produces the following directory listing:

```
-rwxrwxrwx  1     23 Apr  8 12:39   apple
-rwxrwxrwx  1   8307 Apr  8 12:39   peaches
-rwxrwxrwx  1   1280 Apr  8 12:39   pears
-rwxrwxrwx  1    771 Apr  8 12:39   plums
```

Note that **ls | crc -fdc -** *is equivalent to* **crc -dc ***, and both would produce output similar to:

```
$8AC3     23   Apr  8 12:39:51 1986   apple
$FD06   8307   Apr  8 12:39:51 1986   peaches
$C3B0   1280   Apr  8 12:39:51 1986   pears
$02D2    771   Apr  8 12:39:51 1986   plums
```

A means of generating checksums for an entire directory hierarchy is:

find *root_path* **-type f -print | crc -fcd -**

Use of the '-type f' option on **find** is recommended because **crc** will generate the crc for the directory files themselves if presented with their names.

You can extract just the crc and length from a stream of crc's by using the **cut** command. When appended to the above command,

find *args* **-type f -print | crc -fcd - | cut -f1,2**

produces an output of two columns: the crc and the file's length.

FILES

/usr/bin/crc

AME

create - create master release media utility, R3.1

ESCRIPTION

The "create" program is designed to be used to create a set of master distribution media for application software products. The new version, R3.1, is similar to previous versions, although it has been completely re-designed and re-implemented to make it much more robust and tolerant of operator errors.

One new feature of particular interest is the automatic creation of a file containing the crc, length, and modification time for each file included on the distribution media. A copy of this file is put on the media and transferred to a customer's system, while another version is derived by upgrade for cross-checking purposes.

MEDIA CREATED WITH THIS RELEASE OF CREATE CAN ONLY BE READ BY R3.1 AND LATER VERSIONS OF UPGRADE! OLD VERSIONS OF UPGRADE WON'T WORK.

All user responses are limited to a specific length; some are limited to a specific set of valid responses (shown in the dialogue below in curly braces, eg. {Yes, No}). Any responses that exceed the valid internal lengths are truncated. The user is, however, now given the opportunity to verify what was entered before creating the media so as to correct any errors before they are committed to disk. (Previous versions didn't allow verification except by running Upgrade.)

Since create is primarily an interactive program, it's use is explained in the next section by examining a typical dialogue that a user would encounter. The last section describes the error messages that create produces, divided into most-likely and least-likely-to-be-encountered sections.

NTERACTING WITH CREATE

This section describes the interactive dialogue that a user encounters when using create. Lines shown in **boldface** typefont represent what the create program displays to the user. Text shown in *italics* is commentary intended to supplement the discussion; it never appears on the screen. User responses are underlined; they are examples of what a user might typically enter.

Create is invoked by typing "create" at the system prompt level.

$ create

The screen is now cleared via 'tput clear'

Create Master Release Media Utility, R3.1
Copyright 1984,86 by Motorola Computer Systems, Inc.

What type of media is this product distributed on?
{Floppy disk, 1.2MB floppy, Tape cartridge, Cartridge disk, 9-track tape}
--> xxx

Enter one of these media types here. An invalid input, eg. "xxx" shown here, produces the message:

You must enter the first letter of one of the choices and a RETURN,
or just a RETURN to select the first choice.

What type of media is this product distributed on?
{Floppy disk, 1.2MB floppy, Tape cartridge, Cartridge disk, 9-track tape}
--> c *(identifies a cartridge disk for use)*

If the media needs to be formatted before it can be used, the following questions are asked:

If you like, I will automatically format the media for you.
Would you like this?
{Yes, No}
--> y

Enter the shell command needed to format this type of media:
-->

The particular command needed here will vary, depending upon the device type, the system in use, its configuration, etc. The following virtual device names are normally linked to their respective raw device names in /dev, and should be allowable in the response:

/dev/FLOPPY	*for 640K floppy disk*
/dev/FLOPPY.MB	*for 1.2MB floppy disk*
/dev/IOMEGA	*for 5MB Iomega cartridge disks*

If specified, this command is issued to the shell each time a new volume of media is mounted.

Mount media volume #1, then hit RETURN...

If automatic formatting was requested, then the specified format command is displayed now (not shown) as it is invoked.

What is the name of this product? --> example

What is this product's new release identifier? --> EXAMPLE 3.0

Enter a file name (<11 chars; similar to the product name) for auditing needs
--> example (#2)

This name is used for auditing purposes and must conform to UNIX file system naming conventions. It is a good idea to use some variant of the product's name here, only without any embedded spaces or other funny characters. If illegal characters are entered, the following warning message is displayed and the prompt is re-issued:

This name must conform to UNIX file naming conventions!!!

Enter a file name (<11 chars; similar to the product name) for auditing needs
--> example

Enter the pathname for the directory where this product is found
(you're now in *<name-of-current-working-directory>*)
--> xyz

Any absolute or valid relative pathname is legal here. If you're already in the root directory for the product, just enter a " . " (dot). If an invalid directory name is entered, eg. the "xyz" shown, then the following message is displayed:

create: warning -- 'xyz' is an invalid directory path!!!
(errno = 2)

Enter the pathname for the directory where this product is found
(you're now in /usr/local/src/example/common)
--> . .
(The product's root pathname is /usr/local/src/example)

Into what directory (on the TARGET SYSTEM) should Upgrade copy this product?
-->

This is where the product will be installed in a user's system when he runs Upgrade. A null response is not allowed, and simply causes the prompt to be repeated. There are two other types of invalid responses:

Into what directory (on the TARGET SYSTEM) should Upgrade copy this product?
--> example *(invalid response type 1)*

create: warning -- The target directory name must begin with a '/'!

Into what directory (on the TARGET SYSTEM) should Upgrade copy this product?
--> / *(invalid response type 2)*

create: warning -- The target directory name cannot be just '/'!

Into what directory (on the TARGET SYSTEM) should Upgrade copy this product?
--> /d31/stuff/example

Note that if this directory path doesn't exist in the USER'S SYSTEM, Upgrade will create it, if possible.

Enter a command string (<128 chars) to be executed at START of upgrade:
--> date

This command, if specified, is simply submitted to the shell by Upgrade before any of the product's files are read from the distribution media. In this example, the date command is given, although any sequence of commands and/or scripts is legal.

Enter a command string (<128 chars) to be executed and END of upgrade:
-->

Similarly, this optional command string is executed by Upgrade after all of the product's files have been copied into the specified TARGET SYSTEM directory. For either command string, a null response is accepted, as in this example.

Please stand-by for a moment while I figure some things out...

(nnnn media blocks needed) . . .
(deriving an audit file now) . . .

As create computes how much space will occupy on the distribution media, the first line above is displayed and updated periodically to show its progress. After most of the media blocks have been counted, the second line above is displayed while create generates a crc file. When this is done, the first line is redisplayed showing the total block count for the media.

NOTE THAT THE AUDIT FILE DERIVATION CAN TAKE A MINUTE OR MORE! BE PATIENT!

*The audit files are placed into the /usr/AUDITS directory for use by Upgrade. Specifically, the *.cr0 file is copied to the distribution media for later installation into a user's system for auditing needs.*

When this phase is finished, the screen is cleared via 'tput clear'.

Create Master Release Media Utility, R3.1

This product's name is 'example'
It's release identifier is: EXAMPLE 3.0
(Auditing facilities will use the name: example)
It is located in the directory rooted at '/usr/local/src/example'.
It occupies about 292 x 1024-byte disk blocks in the file system,
and requires 268 (1024-byte) blocks of distribution media.

The distribution media was specified as Cartridge disk.
So, 1 volume(s) of this media will be needed to distribute this product.

Upgrade will execute this command string prior to installation:
date
It will then install the product into the directory named:
/d3/stuff/example
Finally, this command string will be invoked after installation:
--- none specified ---

Are these input parameters correct?
{Yes, No}
-->

At this point, a 'no' response will allow the user to re-define everything. A 'y' or null response will start the creation of the distribution media.

(Note that the byte count shown in the forth line above (i.e., "292 x 1024-byte blocks") will vary, depending upon whether the file system containing the product's files uses 512 or 1024 byte blocks -- both create and upgrade automatically detect and adjust for any variations here. The distribution media is always written with 1024 byte blocks.)

------ FILE COPY IN PROGRESS ------

nnnn blocks written to vol #1 from /usr/local/src/example.
nnnn blocks being verified on media . . .

The above two lines display create's progress as it copies the product's files to the media (top line), and then as it re-reads the media to verify its integrity.

In situations where more than one media volume is needed, the following prompt is issued after each volume has been filled:

Mount media volume #n, then hit RETURN...

Automatic formatting is performed now, if specified (not shown).

nnnn blocks written to volume #n from <source_directory>...
nnnn blocks being verified on media . . .

and so on, until the entire product has been written to media.

Now creating an audit file for this product...

Finished creating master(s) for 'example (EXAMPLE 3.0)'.

(A table-of-contents listing of this product with crc's is contained in:
'<audit_file_name>'

Do you wish to create a master copy of another product?
{No, Yes}
-->

If you're done, you can just hit RETURN now.

$ *This ends the example dialog!*

IAGNOSTICS

Most parameters are validated before being committed for use, either by the program or by the user via the status display. It is possible, however, for an internally executed shell command to die, in which case an error message may or may not be produced. Known error conditions fall into two categories: warnings and fatal errors. Warnings produce a message, but allow for continued execution. Fatal errors include any error that would prevent a complete

creation of the distribution media. The defined warning and fatal error messages are enumerated below.

Note that if the /usr/bin/crc program is missing, create should be aborted and re-run after crc is located and installed.

--- WARNING MESSAGES ---

Warnings likely to be encountered:

The target directory name must begin with a '/'
The target directory cannot be just a '/'
The media failed verification
'<pathname>' is an invalid directory path

Warnings unlikely to be encountered:

can't determine size of audit file
can't read/verify end-of-volume flag from media
can't read/verify checksum block from media

--- FATAL ERROR MESSAGES ---

Fatal errors likely to be encountered:

You must be logged in as root or have root's setuid permission set!

Fatal errors unlikely to be encountered:

can't open device {<dev_name>} for writing
error encountered while writing header info
error writing to output media
error reading from input data stream
error encountered while writing end-of-volume flag
error encountered while writing checksum to the media
unable to open output media for reading
error encountered while reading header
error occurred attempting to read data from media
The end-of-volume flag block is corrupted
cannot chdir to '<source_root_directory>'

FILES

tput
/usr/bin/crc
/usr/AUDITS/*

SEE ALSO

upgrade(1M), tput(1M), crc(1M), create(4)

NAME

dcopy — copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [—sX] [—an] [—d] [—v] [—ffsize:isize] inputfs outputfs

DESCRIPTION

Dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is an appropriately sized file system, to hold the reorganized result. For best results *inputfs* should be the raw device and *outputfs* should be the block device. *Dcopy* should be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are:

—s*X* supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the —s option of *fsck*(1M).

—a*n* place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

—d leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

—v reports how many files were processed, and how big the source and destination freelists are.

—f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If the option (or :*isize*) is not given, the values from the *inputfs* are used.

Dcopy catches interrupts and quits and reports on its progress. To terminate *dcopy*, send a quit signal, and *dcopy* will no longer catch interrupts or quits. *Dcopy* also attempts to modify its commandline arguments so its progress can be monitored with *ps*(1).

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

BUGS

If a non-zero length fifo file (named pipe) is present on the input (source) file system, the output (target) file system will be corrupted. To work around this problem, find the fifo file on the input file system and either delete it or zero it out. The error that *fsck* finds on the output file system is a duplicate block error, which will require elimination of files on that file system to correct. Under normal circumstances this error should not be encountered, because a fifo file goes to zero length if no processes have it open, which is the usual case if the input file system can be unmounted.

NAME

dcpy – copy file systems for optimal access time

SYNOPSIS

/mot/bin/dcpy *in-alias* *out-alias*

DESCRIPTION

Dcpy uses the *permissions* file to change *in-alias* (a block device) and *out-alias* (a block device) into *inputfs* and *outputfs*. *Dcpy* then copies file system *inputfs* to *outputfs*. *Inputfs* is the existing file system; *outputfs* is a file system appropriately sized to hold the reorganized result. *Dcpy* uses the *fsize* parameter in the *permissions* file for the size of the file system on *outputfs*. *Dcpy* should be run on unmounted file systems (in the case of the root file system, copy to a new pack). With no arguments, *dcpy* copies files from *inputfs*, compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. *Dcpy* places files not accessed in 7 days after the free blocks of *outputfs*. *Dcpy* moves subdirectories to the beginning of directories. *Dcpy* catches interrupts and quits and reports on its progress. To terminate *dcpy*, send a quit signal, and *dcpy* will no longer catch interrupts or quits.

FILES

/etc/dcopy
/mot/bin/dcpy
/mot/etc/perms permission file

SEE ALSO

dcopy(1M), fsck(1M), mkfs(1M), ps(1).

BUGS

If a non-zero length fifo file (named pipe) is present on the input (source) file system, the output (target) file system will be corrupted. To work around this problem, find the fifo file on the input file system and either delete it or zero it out. The error that *fsck*(1M) finds on the output file system is a duplicate block error, which will require elimination of files on that file system to correct. Under normal circumstances this error should not be encountered, because a fifo file goes to zero length if no processes have it open, which is the usual case if the input file system can be unmounted.

NAME

devnm — device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

Devnm identifies the special file associated with the mounted file system where the argument *name* resides. (As a special case, both the block device name and the swap device name are printed for the argument name **/** if swapping is done on the same disk section as the **root** file system.) Argument names must be full pathnames.

This command is most commonly used by **/etc/rc** (see *bcheckrc*(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

/etc/devnm /usr

produces

dsk/*cntrlr***_1s0 /usr**

if **/usr** is mounted on **/dev/dsk/***cntrlr***_1s0.**

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

bcheckrc(1M), setmnt(1M).

NAME

df — report number of free disk blocks

SYNOPSIS

df [**—t**] [**—f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free inodes available for online file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/dsk/***cntrlr*_**0s1**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The **—t** flag causes the total allocated block figures to be reported as well.

If the **—f** flag is given, only an actual count of the blocks in the free list is made (free inodes are not reported). With this option, *df* reports on raw devices.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

NAME

dinit – disk initializer

SYNOPSIS

/etc/dinit [–force] [-T] [–d desc] [–b file] [–t file] type rdev

DESCRIPTION

Dinit can be used to initialize specified disk *types*. The *type* must be one from the file **/etc/diskdefs.** Current values are shown in the following tables:

VM21 CONTROLLER

Drive Name	*type* Value
50Mb Lark Module Drive	vm21lark25
16Mb Lark Module Drive	vm21lark8
80Mb Cartridge Module Drive	vm21cmd80
16Mb Cartridge Module Drive	vm21cmd16
Double-sided 8" Floppy Diskette	vm21dssd8
Single-sided 8" Floppy Diskette	vm21sssd8

VM22 CONTROLLER

Drive Name	*type* Value
Removable 25Mb Lark	vm22R25L
Fixed 25Mb Lark	vm22F25L
Removable 8Mb Lark	vm22R8L
Fixed 8Mb Lark	vm22F8L
Removable 16Mb Cmd	vm22R16C
Fixed 16Mb CMD	vm22F16C
Fixed 80Mb CMD	vm22F80C
*Double-sided Double Density 8" Floppy	vm22dsdd8
*Single-sided Double Density 8" Floppy	vm22ssdd8
*Double-sided Single Density 8" Floppy	vm22dssd8
*Single-sided Single Density 8" Floppy	vm22sssd8
**Double-sided Double Density 5 ¼" Floppy	vm22dsdd5
**Single-sided Double Density 5 ¼" Floppy	vm22ssdd5
**Double-sided Single Density 5 ¼" Floppy	vm22dssd5
**Single-sided Single Density 5 ¼" Floppy	vm22sssd5

* Motorola 8" Format
** IBM 5 ¼" Format

MVME319 CONTROLLER (ID CONTROLLER)

Drive Name	*type* Value
40Mb Micropolis Winchester	idwm40
**Double-sided Double Density 5 ¼" Floppy	iddsdd5
*Double-sided Single Density 8" Floppy	iddssd8
*Cipher Data Products CT525 FloppyTape	idftape

* Motorola 8" Format
** IBM 5 ¼" Format

MVME320 CONTROLLER

Drive Name	*type* Value
40Mb Vertex Winchester	m32040v
15Mb Computer Memories Winchester	m32015
40Mb Micropolis Winchester	m32040m
70Mb Micropolis Winchester	m32070m
40Mb Miniscribe Winchester	m32040s
40Mb Toshiba Winchester	m32040t
70Mb Toshiba Winchester	m32070t
70Mb Priam Winchester	m32070p
140Mb MAXTOR Winchester	m320140
**Double-sided Double Density 5 ¼" Floppy	m320dsdd5

** IBM 5 ¼" Format

MVME360 CONTROLLER

Drive Name	*type* Value
337Mb Fujitsu SMD	m360337

For disk types with software or hardware bad track handling, the alternate track numbers will be taken from the file **/etc/diskalts/***type*, where *type* is the type name given in **/etc/diskdefs**. If no file **/etc/diskalts/***type* exists, the user will be prompted to enter the alternate track numbers interactively. There is no software bad track support for floppy diskettes.

The *rdev* argument specifies a raw device, which must be of the form **/dev/r***string*. There must be a corresponding block device **/dev/***string* with the same minor device number as the character device. *Dinit* must be executed over slice 7 of the raw device.

The following options are provided for *dinit*:

–f Format disk. When formatting an unformatted disk, two read errors appear on the screen. These errors occur because the controller is trying to read configuration information from the disk. The messages can be ignored; the disk will be formatted as requested.

–o Override disk contents, including type and bad tracks.

–r Read the current bad track list from the disk. By default, it is printed in <*head*> <*cylinder*> format. If the **–T** option is in effect, track numbers are printed instead. The actual list is printed to standard output, the header to standard error.

–c Check for new bad tracks. A read/write pass is executed for each track on the device. If a read or write error occurs for a track, the track number is stored in a list of bad tracks for the device (if **–o** was NOT specified, it is added to the current list read from the device). After the pass, any new bad tracks found are printed. As with **–r** above, the format of the list is controlled by the **–T** option.

–e Use **EXORMACS** instead of **MOTOROLA** in sector 0, for compatibility with VM03 and EXORmacs bootloaders.

–d *desc* Use *desc* as description string in sector 0.

–b *file* Use *file* (a.out format) as the bootloader program.

–t *file* Take bad track information from *file*, instead of interactively. By default, *file* is assumed to contain some number of lines of the form:

head cylinder

indicating the location of manufacturer specified defects. If, however, the –T option is in effect, the file is assumed to contain track numbers.

Unless the **–f** or **–o** options are given, *dinit* will examine the disk to ensure the disk type is not being changed (i.e. from m32070m to m32070t). Also, it will read the previous bad track list. Therefore, it is not necessary to re-enter bad track numbers on subsequent use of *dinit* on a disk. This is useful for changing the bootloader, description string, etc. (For calculations of bad track numbers, refer to the specific format utility, e.g., *m320fmt*(1M).)

Whenever new bad tracks are given to *dinit*, the layout of the areas of the disk provided for file systems may be arbitrarily remapped. Therefore, all useful information from the disk should be copied to backup media before adding bad tracks and then copied back when *dinit* has finished.

XAMPLE

/etc/dinit -f -o -t /etc/badtracks/00 -b /stand/m68k/boots/vmeboot m32070m /dev/r00s7

This command formats the first 70MB disk attached to the first MVME320 controller using a disk defect list entered into /etc/badtracks/00. All data on the disk is destroyed. The disk contains a bootloader file that will boot the operating system after it is installed. Note: This particular command could only be run while booted on a floppy or winchester drive other than 00.

ILES

/etc/diskdefs	disk definition file
/etc/diskalts/*	alternate track numbers

EE ALSO

m320fmt(1M)
SYSTEM V/68 Administrator's Guide.

NAME

diskusg – generate disk accounting data by user ID

SYNOPSIS

/usr/lib/acct/diskusg [options] [files]

DESCRIPTION

Diskusg generates intermediate disk accounting information from data in *files*, or the standard input if the *files* argument is omitted. *Diskusg* outputs lines on the standard output, one per user, in the following format:

uid login #blocks

where

uid is the numerical user ID of the user; *login* is the login name of the user; and *#blocks* is the total number of disk blocks allocated to this user.

Diskusg normally reads only the inodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

Diskusg recognizes the following options:

–s The input data is already in *diskusg* output format. *Diskusg* combines all lines for a single user into a single line.

–v Verbose. Print a list on standard error of all files that are charged to no one.

–i*fnmlist* Ignore the data on those file systems whose file system name is in *fnmlist*. *Fnmlist* is a list of file system names, separated by commas or enclosed within quotes. *Diskusg* compares each name in this list with the file system name stored in the volume ID (refer to *labelit* in *volcopy*(1M)). For example, sites that want to account only for disk blocks allocated to non-administrative users can use the **–i** option to ignore certain file systems, e.g., **root** and **usr**. In this case, the administrator should use the *fsname* field reported by *labelit* as the arguments of the **–i** option to *diskusg* in the shell script *dodisk*. To ignore **root** and **usr**, line 31 in *dodisk* would be changed to:
diskusg –i root,usr $args > dtmp

–p*file* Use *file* as the name of the password file to generate login names. The file **/etc/passwd** is used by default.

–u*file* Write records to *file* of files that are charged to no one. Records consist of the special filename, the inode number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* (refer to *acct*(1M)) which generates total accounting reocrds that can be merged with other accounting records. *Diskusg* is normally run in the shell script *dodisk* (refer to *acctsh*(1M)). Note that in previous releases the disk blocks of a file were charged to the user whose login directory hierarchy contained the file. In the current release, *diskusg* charges disk blocks to the file's owner. This change may result in slightly different disk block usage reports when run on the same data. The previous method of disk accounting may be invoked with the **–o** option of *dodisk*.

EXAMPLES

The following will generate daily disk accounting information:

```
for i in /dev/rp00 /dev/rp01 /dev/rp10 /dev/rp11; do
 diskusg $i > dtmp.`basename $i`&
done
```

```
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

/etc/passwd used for conversions of user ID to login name

SEE ALSO

acct(1M), acctsh(1M), acct(4).

NAME

errdead — extract error records from dump

SYNOPSIS

/etc/errdead dumpfile [namelist]

DESCRIPTION

When hardware errors are detected by the system, an error record that contains information pertinent to the error is generated. If the error-logging daemon *errdemon*(1M) is not active or if the system crashes before the record can be placed in the error file, the error information is held by the system in a local buffer. *Errdead* examines a system dump (or memory), extracts such error records, and passes them to *errpt*(1M) for analysis.

The *dumpfile* specifies the file (or memory) that is to be examined. The system namelist is specified by *namelist*; if not given, **/unix** is used.

FILES

/unix	system namelist
/usr/bin/errpt	analysis program
/usr/tmp/errXXXXXX	temporary file

DIAGNOSTICS

Diagnostics may come from either *errdead* or *errpt*. In either case, they are self-explanatory.

SEE ALSO

errdemon(1M), errpt(1M).

NAME

errdemon – error-logging daemon

SYNOPSIS

/usr/lib/errdemon [file]

DESCRIPTION

The error logging daemon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by using errstop (see *errstop*(1M)). Only the superuser may start the daemon, and only one daemon may be active at any time.

FILES

/dev/error	source of error records
/usr/adm/errfile	repository for error records

DIAGNOSTICS

The diagnostics produced by *errdemon* are self-explanatory.

SEE ALSO

errpt(1M), errstop(1M), kill(1), err(7).

NAME

errpt — process a report of logged errors

SYNOPSIS

errpt [options] [files]

DESCRIPTION

Errpt processes data collected by the error logging mechanism (*errdemon*(1M)) and generates a report of that data. The default report is a summary of all errors posted in the files named. Options apply to all files and are described below. If no files are specified, *errpt* attempts to use **/usr/adm/errfile** as *file*.

A summary report notes the options that may limit its completeness, records the time stamped on the earliest and latest errors encountered, and gives the total number of errors of one or more types. Each device summary contains the total number of unrecovered errors, recovered errors, errors unabled to be logged, I/O operations on the device, and miscellaneous activities that occurred on the device. The number of times that *errpt* has difficulty reading input data is included as read errors.

Any detailed report contains, in addition to specific error information, all instances of the error logging process being started and stopped, and any time changes (via *date*(1)) that took place during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report may be limited to certain records in the following ways:

—s *date* Ignore all records posted earlier than *date*, where *date* has the form *mmddhhmmyy*, consistent in meaning with the *date*(1) command.

—e *date* Ignore all records posted later than *date*, whose form is as described above.

—a Produce a detailed report that includes all error types.

—d *devlist* A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces. *Errpt* is familiar with the common form of identifiers. For the EXORmacs, the device for which errors are logged is *ud*(7). For the VME/10, the device is *wd*(7). For 3B20S, the devices are DFC, IOP, and MT. For Digital Equipment Corporation machines, the (block) devices for which errors are logged are RP03, RP04, RP05, RP06, RP07, RS03, RS04, TS11, TU10, TU16, TU78, RK05, RK06, RK07, RM05, RM80, and RF11. Additional identifiers are **int** and **mem** which include detailed reports of stray-interrupt and memory-parity type errors respectively.

—p *n* Limit the size of a detailed report to *n* pages.

—f In a detailed report, limit the reporting of block device errors to unrecovered errors.

FILES

/usr/adm/errfile default error file

SEE ALSO

errdead(1M), errdemon(1M), errfile(4), date(1).

NAME

errstop — terminate the error-logging daemon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

The error-logging daemon *errdemon*(1M) is terminated by using *errstop*. This is accomplished by executing *ps*(1) to determine the daemon's identity and then sending it a software kill signal (see *signal*(2)); **/unix** is used as the system namelist if none is specified. Only the superuser may use *errstop*.

FILES

/unix default system namelist

DIAGNOSTICS

The diagnostics produced by *errstop* are self-explanatory.

SEE ALSO

errdemon(1M), ps(1), kill(2), signal(2).

NAME

ff — list filenames and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving inode data for files that match the selection criteria. Output consists of the pathname for each saved inode, plus any other file information requested (refer to the print options below). Output fields are positional. The output is produced in inode order; fields are separated by tabs. The default line produced by *ff* is:

pathname i-number

With all options enabled, output fields would be:

pathname i-number size uid

The argument *n* in the option descriptions that follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *—n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24-hour period.

—I	Do not print the inode number after each pathname.
—l	Generate a supplementary list of all pathnames for multiple linked files.
—p *prefix*	The specified *prefix* is added to each generated pathname. The default is ..
—s	Print the file size, in bytes, after each pathname.
—u	Print the owner's login name after each pathname.
—a *n*	Select if the inode has been accessed in *n* days.
—m *n*	Select if the inode has been modified in *n* days.
—c *n*	Select if the inode has been changed in *n* days.
—n *file*	Select if the inode has been modified more recently than the argument *file*.
—i *inode-list*	Generate names for only those inodes specified in *inode-list*.

EXAMPLES

To generate a list of the names of all files on a specified file system:

ff —I /dev/diskroot

To produce an index of files and i-numbers that are on a file system and have been modified in the last 24 hours:

ff —m —1 /dev/diskusr > /log/incbackup/usr/tuesday

To obtain the pathnames for inodes 451 and 76 on a specified file system:

ff —i 451,76 /dev/rdsk/*cntrlr*_**1s0**

SEE ALSO

finc(1M), find(1), frec(1M), ncheck(1M).

BUGS

Only a single pathname is generated for a multiply linked inode, unless the **—l** option is specified. When **—l** is specified, no selection criteria apply to the names generated; all possible names for every linked file on the file system are included in the output.

On very large file systems, memory may run out before *ff* does.

NAME

filesave, tapesave — daily/weekly SYSTEM V/68 file system backup

SYNOPSIS

/etc/filesave.?
/etc/tapesave

DESCRIPTION

These shell scripts are provided as models. They are designed to provide a simple, interactive operator environment for file backup. *Filesave.?* is for daily disk-to-disk backup, and *tapesave* is for weekly disk-to-tape.

The suffix **.?** can be used to name another system where two (or more) machines share disk drives (or tape drives) and one or the other of the systems is used to perform backup on both.

SEE ALSO

shutdown(1M), volcopy(1M).

AME

finc – fast incremental backup

YNOPSIS

finc [selection-criteria] file-system raw-tape

ESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-tape* . Mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit* (see *volcopy*(1M)). The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument **n** in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *–n* means less than *n*, and *n* means exactly *n*. A day is defined as 24 hours.

–a *n*	True if the file has been accessed in *n* days.
–m *n*	True if the file has been modified in *n* days.
–c *n*	True if the inode has been changed in *n* days.
–n *file*	True for any file which has been modified more recently than the argument *file*.

XAMPLES

To write a tape consisting of all files from file-system **/usr** modified in the last 48 hours:

finc –m –2 /dev/rdiskusr /dev/rmt/*cntrlr*_0m

E ALSO

cpio(1), ff(1M), frec(1M), volcopy(1M).

NAME

format – disk initializer

SYNOPSIS

/mot/bin/fmt [options] *alias*

DESCRIPTION

Fmt(1M) checks to see that *alias* is in the *permissions* file and that format permission is given. It then lists the action it is going to take (i.e. what program it is going to execute over what device) and asks for confirmation. Any character other than 'y' is taken as a negative response and no action is taken. Otherwise, *fmt*(1M) passes on any *options* given and slice 7 of the raw device found in the *permissions* file to the *format-program* specified in the *permissions* file.

FILES

/etc/diskdefs	disk definition file
/etc/diskalts/*	alternate track numbers
/etc/dinit	
/mot/bin/fs	
/mot/etc/perms	permissions file

SEE ALSO

dinit(1M),
SYSTEM V/68 Administrator's Guide.

⟨AME

frec – recover files from a backup tape

YNOPSIS

/etc/frec [**–p** path] [**–f** reqfile] raw-tape i-number:name ...

)ESCRIPTION

Frec recovers files from the specified *raw-tape* backup tape written by *volcopy*(1M) or *finc*(1M), given the *i-numbers*. The data for each recovery request is written into the file given by *name*.

The **–p** option allows specification of a default prefixing *path* different from the current working directory. This is prefixed to any *names* that are not fully qualified, i.e., that do not begin with / or ./. If any directories are missing in the paths of recovery *names*, they are created.

–p *path* Specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.

–f *reqfile* Specifies a file which contains recovery requests. Using only one entry per line, the format is: *i-number:newname*

XAMPLES

To recover file i-number 1216, when backed up into a file named **junk** in your current working directory:

frec /dev/rmt/*cntrl*_m0 1216:junk

To recover files with i-numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/drane/a.c**:

frec –p /usr/src/cmd /dev/rmt/*cntrlr*_m0 14156:a 1232:b 3141:/usr/drane/a.c

⟨E ALSO

cpio(1), ff(1M), finc(1M), volcopy(1M).

UGS

While creating the intermediate directories contained in a pathname, *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.

NAME

fs – construct a file system

SYNOPSIS

/mot/bin/fs [*disk* [*blocks*[:*inodes*]]]

DESCRIPTION

Fs(1M) builds a file system with a single empty directory on it. The argument *disk* and the *permissions* file are used to determine the device to build a file system on. This device will be the first match of *disk* and the *real device* or *alias* entries in the *permissions* file. If the *disk* argument is not given then the first *alias* of **default** in the *permissions* file will be used.

Fs actually uses the raw version of the listed *real device* (by prepending an 'r' to the name).

The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system occupies. This value may not be larger than the default value specified in the *permissions* file. If the number of blocks is not specified the default value in the *permissions* file is used. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of logical blocks divided by four.

FILES

/etc/mkfs
/mot/bin/fs
/mot/etc/perms permissions file

SEE ALSO

dir(4), fs(4), mkfs(1M), perms(4)
SYSTEM V/68 Administrator's Guide.

NAME

fsba — file system block analyzer

SYNOPSIS

fsba file-system ...

DESCRIPTION

Fsba determines the number of extra sectors (1 sector has 512 bytes) needed when the file system logical block size is increased from 512 bytes per block to 1024 bytes/block. *File-system* should be specified by device name (e.g., **/dev/dsk/***cntrlr*_**1s1**).

Fsba determines how many sectors are currently allocated for the 512 bytes/block file system, and how many sectors are required for the 1024 bytes/block converted file system. *Fsba* also prints out the number of allocated and free inodes for each *file-system*.

If the number of free sectors for the 1024 bytes/block file system is negative, the file system is too large to convert to 1024 bytes/block.

SEE ALSO

fs(4).

NAME

fsck, dfsck — file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [—**y**] [—**n**] [—s*X*] [—S*X*] [—**t** file] [—**q**] [—**D**] [—**f**] [file-systems]

/etc/dfsck [options1] filsys1 ... — [options2] filsys2 ...

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for SYSTEM V/68 files. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent, the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* defaults to a **—n** action.

Fsck has more consistency checks than its predecessors *check*, *dcheck*, *fcheck*, and *icheck* combined.

The following options are interpreted by *fsck*.

—y Assume a yes response to all questions asked by *fsck*.

—n Assume a no response to all questions asked by *fsck*; do not open the file system for writing.

—s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the superblock of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock does not continue to be used, or written on the file system.

The —s*X* option allows for creating an optimal free-list organization. The following forms of *X* are supported for the following devices:

—sBlocks-per-cylinder:Blocks-to-skip

If *X* is not given, the values used when the file system was created are used. If these values were not specified, then the value *400:7* is used.

—S*X* Conditionally reconstruct the free list. This option is like —s*X* above, except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **—S** forces a **no** response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

—t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **—t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **—t** flag, *fsck* prompts the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

—q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifos** are silently removed. If *fsck* requires it, counts in the superblock are automatically fixed and the free list salvaged.

—D Directories are checked for bad blocks (useful after system crashes).

—f Fast check. Check block and sizes (Phase 1) and check the free list (Phase 5). The free list is reconstructed (Phase 6) if it is necessary.

If no *file-systems* are specified, *fsck* reads a list of default file systems from the file **/etc/checklist.**

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated inode.
   - Inode number out of range.
8. Super Block checks:
   - More than 65536 inodes.
   - More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory, if the files are not empty. The user is notified if the file or directory is empty or not. If it is empty, *fsck* silently removes it. *Fsck* forces the reconnection of directories which are not empty. The name assigned is the inode number. The only restriction is that the directory **lost+found** must pre-exist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found,** copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *Options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A — is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* prints the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Do not use *dfsck* to check the *root* file system.

FILES

/etc/checklist	contains default list of file systems to check.
/etc/checkall	optimizing *dfsck* shell file.

SEE ALSO

checkall(1M), clri(1M), ncheck(1M), checklist(4), fs(4), crash.macs(8).
"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

BUGS

Inode numbers for . and .. in each directory should be checked for validity.
Unless explicitly called with the —**n** flag, *fsck* will automatically clear unreferenced inodes. This will damage a mounted file system.

DIAGNOSTICS

The diagnostics produced by *fsck* are self-explanatory.

NAME

fscv — convert files between M68000 and VAX-11/780 processors

SYNOPSIS

/etc/fscv —v ispecial [ospecial]
/etc/fscv —m ispecial [ospecial]

DESCRIPTION

Fscv converts file systems between M68000 and VAX-11/780 formats. The super block, free list, and inodes are converted to the format of the output file. *Fscv* may be executed on M68000 and VAX processors. The mandatory flag specifies the format of the converted file system:

—v Convert file system from M68000 to VAX format.

—m Convert file system from VAX to M68000 format.

Ispecial is the name of a special file containing a file system to be converted (e.g., **/dev/rdsk/***cntrlr***_1s0**). The optional *ospecial* is the name of the special file to receive the results of the conversion. If *ospecial* is specified, the entire contents of *ispecial* are copied to *ospecial* before the conversion is performed. If *ospecial* is not specified, an in-place conversion of *ispecial* is performed. The following items should be noted before executing *fscv*:

1. A file system consistency check (*fsck*(1M)) should be performed on *ispecial* immediately prior to executing *fscv*.
2. Neither *ispecial* nor the optional *ospecial* should contain a mounted file system during execution of *fscv*. Modification to either the input or the output file system while *fscv* is executing will probably corrupt the converted file system.
3. A backup of *ispecial* (see *volcopy*(1M)) is highly recommended if an in-place conversion is to be performed. System crashes, I/O errors, etc., during execution of *fscv* may destroy the file system contained in *ispecial*. Also, if the optional *ospecial* is specified, any data contained in that special file is over written.
4. If the optional *ospecial* is specified, this special file must be large enough to contain the entire contents of *ispecial*. See the appropriate special files in section 4.

EXAMPLES

Copy and convert a file system from M68000 to VAX format:

/etc/fscv —v /dev/rdsk/*cntrlr***_0s0 /dev/rdsk/***cntrlr***_1s0**

Perform an in-place conversion from VAX to M68000 format:

/etc/fscv —m /dev/rdsk/*cntrlr***_1s0**

BUGS

The boot block is not modified during conversion; the resulting file system is not bootable. No data contained in the files of the file system are modified.

SEE ALSO

fsck(1M), volcopy(1M).

NAME

fsdb — file system debugger

SYNOPSIS

/etc/fsdb special [—]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an inode. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error checking routines to verify inode and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional — argument or by the use of the **O** symbol. (*Fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and, therefore, works with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to inode address
b	convert to block address
d	directory slot offset
+,—	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=—	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows, since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as inodes
d	print as directories
o	print as octal words
e	print as decimal words

c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current inode. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs and spaces may be used as function delimiters but are not necessary. A line with just a newline character increments the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or inode, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for inode examination and refer to the current working inode:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 — 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an inode format. This now becomes the current working inode.
ln=4	changes the link count for the working inode to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working inode.
2i.fd	prints the first 32 directory entries for the root inode of this file system.
d5i.fc	changes the current inode to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current inode as directory entries.

SEE ALSO

fsck(1M), dir(4), fs(4).

NAME

fuser — identify processes using a file or file structure

SYNOPSIS

/etc/fuser [—**ku**] files [—] [[—**ku**] files]

DESCRIPTION

Fuser lists the process IDs of the processes using the files specified as arguments. For block special devices, all processes using any file on that device are listed. The process ID is followed by **c**, **p** or **r** if the process is using the file as its current directory, the parent of its current directory (only when in use by the system), or its root directory, respectively. If the —**u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the —**k** option is specified, the **SIGKILL** signal is sent to each process. Only the superuser can terminate another user's process (see *kill*(2)). Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

EXAMPLES

fuser —ku /dev/dsk/*cntrlr*_1s?
terminates all processes that are preventing disk drive one from being unmounted if typed by the superuser, listing the process ID and login name of each as it is killed.

fuser —u /etc/passwd
lists process IDs and login names of processes that have the password file open.

fuser —ku /dev/dsk/*cntrlr*_1s? —u /etc/passwd
does both of the above examples in a single command line.

FILES

/unix	for namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

NAME

fwtmp, wtmpfix — manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [—ic]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

Fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formated ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument —**ic** is used to denote that input is in ASCII form, and output is in binary form.

Wtmpfix

Wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A — can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon1* faults when it encounters certain date change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* checks the validity of the name field to ensure that it consists solely of alphanumeric characters, a **$**, or spaces. If it encounters a name that is considered invalid, it changes the login name to **INVALID** and writes a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance of *acctcon1* failure, when processing connect accounting records.

FILES

/etc/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), acct(2), acct(4), utmp(4).

NAME

getty — set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**—h**] [**—t** timeout] line [speed [type [linedisc]]]
/etc/getty **—c** file

DESCRIPTION

Getty is a program that is invoked by *init*(1M). It is the second process in the series (*init-getty-login-shell*) that ultimately connects a user with SYSTEM V/68. Initially *getty* prints the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **—h** flag, *getty* forces a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **—t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* what speed to initially run at, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character.) The default *speed* is 9600 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

none	default
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard HP45
c100	Concept 100

The default terminal is **none**, i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled, in the default condition. The optional fourth argument, *linedisc*, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system, but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, *getty* sets the *speed* of the interface to 9600 baud, specifies that raw mode is used (awaken on every character), echo is suppressed, either parity is allowed, newline characters are converted to carriage return-line feed, and tab expansion is performed on the standard output. It types the login message, then reads the user's name, a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pressing the "break" key. This causes *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a newline or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *termio*(7)).

The user's name is scanned to see if it contains any lowercase alphabetic characters; if not, and if the name is not empty, the system is told to map any future uppercase characters into the corresponding lowercase characters.

Finally, *login* is called with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which places them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **—c** option and ***file***, it scans the file as if it were scanning **/etc/gettydefs** and prints the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *termio*(7) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), login(1), termio(7), gettydefs(4), inittab(4), tty(7), "Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

BUGS

While *getty* does understand simple single-character quoting conventions, it is not possible to quote the special control characters that *getty* uses to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. Therefore, it is not possible to log in via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of the login name or arguments. They will always be interpreted as having their special meanings as described above.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/bin/telinit [0123456sSQqabc]

DESCRIPTION

Init

The primary role of *init* is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* generate *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a run-level at any given time. A run-level can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes generated by *init* for each of these run-levels is defined in the *inittab* file. *Init* can be in one of eight run-levels, 0-6, and *S* or *s*. The run-level is changed by having a privileged user run **/etc/init** (which is linked to **/bin/telinit** and **/etc/init8).** This user-generated *init* sends appropriate signals to the orginal *init* created by the operating system when the system was rebooted, telling it which run-level to change to.

Init is invoked inside the SYSTEM V/68 operating system as the last step in the boot procedure. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(4)). If there is, *init* uses the run-level specified in that entry as the initial run-level to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a run-level from the virtual system console, **/dev/syscon.** If an S (s) is entered, *init* goes into the SINGLE USER level. This is the only run-level that doesn't require the existence of a properly formatted *inittab* file. If **/etc/inittab** doesn't exist, then by default the only legal run-level that *init* can enter is the SINGLE USER level. In the SINGLE USER level, the virtual console terminal **/dev/syscon** is opened for reading and writing, and the command **/bin/su** is invoked immediately. To exit from the SINGLE USER run-level, one of two options can be selected. First, if the shell is terminated (via an end-of-file), *init* reprompts for a new run-level. Second, the *init* or *telinit* command can signal *init* and force it to change the run-level of the system.

When attempting to boot the system, failure of *init* to prompt for a new run-level may be due to the fact that the device **/dev/syscon** is linked to a device other than the physical system teletype (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** by typing a delete on the system teletype which is located with the processor.

When *init* prompts for the new run-level, the operator may only enter one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered, *init* operates as previously described in SINGLE USER mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, **/dev/systty**, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon.** This file is written by *init* whenever SINGLE USER mode is entered. If this file doesn't exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered, *init* enters the corresponding run-level. Any other input is rejected and the user is reprompted. If this is the first time *init* has entered a run-level other than SINGLE USER, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the run-level entered matches that of the entry before any normal processing of *inittab* takes place. In this way, any special initialization of the

operating system (such as mounting file systems) can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that run-level.

Run-level **2** is usually defined by the user to contain all of the terminal processes and daemons that are generated in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* creates a process for each terminal on the system.

For terminal processes, ultimately the shell terminates because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a signal telling it that a process it created has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes generated is kept in **/etc/wtmp** if such a file exists.

To create each process in the *inittab* file, *init* reads each entry and for each entry that should be regenerated, it creates a process. After it has generated all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's run-level. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **init Q** or **init q** command can wake *init* to re-examine the *inittab* file.

If *init* receives a *powerfail* signal (**SIGPWR**) and is not in SINGLE USER mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the run-levels permit) before any further processing takes place. In this way, *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure. It is important to note that the powerfail entries should not use devices that must first be initialized after a power failure has occurred.

When *init* is requested to change run-levels (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run-level. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

Telinit

Telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one character argument and signals *init* via the kill system call to perform the appropriate action. The following arguments serve as directives to *init*.

0–6	place the system in one of the run-levels 0-6.
a,b,c	process only those **/etc/inittab** file entries having the *a*, *b* or *c* run-level set.
Q,q	re-examine the **/etc/inittab** file.
s,S	enter the single-user environment. When this level change is effected, the virtual system teletype, **/dev/syscon**, is changed to the terminal from which the command was executed.

Telinit can only be run by someone who is superuser or a member of group **sys**.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/ioctl.syscon
/dev/syscon
/dev/systty

SEE ALSO

getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), utmp(4).
"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide* .

DIAGNOSTICS

If *init* finds that it is continuously regenerating an entry from **/etc/inittab** more than 10 times in 2 minutes, it assumes that there is an error in the command string, and generates an error message on the system console. It does not regenerate this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab.*

BUGS

When changes to the *init* state of the system are made, a condition could develop where one process opened the console for I/O while another process closed it. The process that had opened the console would be unable to proceed and the sytem would hang, necessitating a reboot. This problem would be encountered only when using single-user mode for other than normal startup and then switching to multi-user mode.

NAME

install — install commands

SYNOPSIS

/etc/install [—c dira] [—f dirb] [—i] [—n dirc] [—o] [—s] file [dirx ...]

DESCRIPTION

Install is a command most commonly used in "makefiles" (see *make*(1)) to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx ...*) are given, *install* searches a set of default directories (**/bin, /usr/bin, /etc, /lib,** and **/usr/lib,** in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If directories (*dirx ...*) are specified after *file*, they are searched before the directories specified in the default list.

The meanings of the options are:

—c *dira* — Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the —s option.

—f *dirb* — Forces *file* to be installed in a given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file is set to **755** and **bin**, respectively. If the file already exists, the mode and owner is that of the already existing file. May be used alone or with the —o or —s options.

—i — Ignores default directory list, searching only through the given directories (*dirx ...*). May be used alone or with options other than —c and —f.

—n *dirc* — If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file is set to **755** and **bin**, respectively. May be used alone or with options other than —c and —f.

—o — If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a normally text busy file such as **/bin/sh** or **/etc/getty**, where the existing file cannot be removed. May be used alone or with options other than —c.

—s — Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1), mk(8).

NAME

killall — kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is a procedure used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

Killall is chiefly used to terminate all processes with open files so that the mounted file systems can be unmounted.

Killall sends *signal* (see *kill*(1)) to all remaining processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), kill(1), ps(1), shutdown(1M), signal(2).

NAME

link, unlink — exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

Link and *unlink* perform system calls on their arguments, abandoning all error checking. These commands may only be executed by the superuser.

SEE ALSO

rm(1), link(2), unlink(2).

NAME

lpadmin — configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin —**p** printer [options]
/usr/lib/lpadmin —**x** dest
/usr/lib/lpadmin —**d**[dest]

DESCRIPTION

Lpadmin configures LP spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs, and change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

One of the —**p**, —**d** or —**x** options must be present for every legal invocation of *lpadmin*.

—**d**[dest] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other options are allowed with —**d**.

—**x***dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class is deleted, too. No other options are allowed with —**x**.

—**p***printer* names a *printer* to which all of the options below refer. If *printer* does not exist then it is created.

The following options are only useful with —**p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

—**c***class* inserts printer *P* into the specified *class*. *Class* is created if it does not already exist.

—**e***printer* copies an existing *printer's* interface program to be the new interface program for *P*.

—**h** indicates that the device associated with *P* is hardwired. This option is assumed when creating a new printer, unless the —**l** option is supplied.

—**i***interface* establishes a new interface program for *P*. *Interface* is the pathname of the new program.

—**l** indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

—**m***model* selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP software (see *Models* below).

—**r***class* removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* is removed.

—**v***device* associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from associating the same *device* with more than one *printer*. If only the —**p** and —**v** options are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the —**v** option and only one of the —**e**, —**i** or —**m** options must be supplied. The —**h** and —**l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and

_(underscore).

Models.

Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **—m**. Alternatively, LP administrators may modify copies of models and then use *lpadmin* **—i** to associate them with printers. The following list describes the *models* and lists the options which they may be given on the *lp* command line using the **—o** keyletter:

dumb interface for a line printer without special functions and protocol. Form feeds are assumed. This is a good model to copy and modify for printers which do not have models.

1640 Diablo 1640 terminal running at 1200 baud, using XON/XOFF protocol. Options:

—12 12-pitch (10-pitch is the default)
—f don't use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the nroff 450 driving table.

hp Hewlett Packard 2631A line printer at 2400 baud. Options:

—c compressed print
—e expanded print

prx Printronix P300 or P600 printer using XON/XOFF protocol at 1200 baud.

EXAMPLES

1. Assuming there is an existing Hewlett Packard 2631A line printer named *hp2*, it uses the **hp** model interface after the command:

   /usr/lib/lpadmin —php2 —mhp

2. To obtain compressed print on *hp2*, use the command:

   lp —dhp2 —o—c files

3. A Diablo 1640 printer called *st1* can be added to the LP configuration with the command:

   /usr/lib/lpadmin —pst1 —v/dev/tty20 —m1640

4. An *nroff* document may be printed on *st1* in any of the following ways:

   nroff —T450 files | lp —dst1 —of
   nroff —T450—12 files | lp —dst1 —of
   nroff —T37 files | col | lp —dst1

5. The following command prints the password file on *st1* in 12-pitch:

   lp —dst1 —o12 /etc/passwd

   NOTE: the **—12** option to the **1640** model should never be used in conjunction with *nroff*.

FILES

/usr/spool/lp/*

SEE ALSO

450(1), accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1), "LP Spooling System" in *SYSTEM V/68 Administrator's Guide*.

NAME

lpsched, lpshut, lpmove — start/stop the LP request scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers.

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked stop printing. Requests that were printing at the time a printer was shut down are reprinted in their entirety after *lpsched* is started again. All LP commands perform their functions even when *lpsched* is not running.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*. The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp* rejects requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1), "LP Spooling System" in *SYSTEM V/68 Administrator's Guide*.

NAME

m320fmt – format disks on the MVME320 disk controller

SYNOPSIS

m320fmt [hard_disk_enable –h heads **–c** cylinders] rawdev

DESCRIPTION

The *m320fmt* utility is used to format disks on the MVME320 controller. Winchester ("hard") disks are formatted in a continuous operation which will keep the controller busy until it completes. Floppy disks are formatted track-by-track, permitting other I/O operations to intervene. Support for bad track handling on the MVME320 controller is done in software only. Therefore the *m320fmt* utility should be used only for diskettes or for media that has no defects listed on the Winchester verification report. To format any media that contains imperfections or that is to be booted, use the *dinit*(1M) utility. Refer to *dinit*(1M). To perform the calculations needed to enter the media imperfections with the *dinit* utility, use the conversion procedures described below.

The following options are available:

The string **hard_disk_enable** must appear as shown to enable formatting of a hard disk. It may not appear if the target disk is a floppy.

–h The number of heads (surfaces per cylinder) on the target hard disk.

–c The number of cylinders on the target hard disk.

rawdev must be a raw device defined on the target unit (/dev/rdsk/m320_ ...) . The slice number is irrelevant for hard disks, but must be one which spans the entire volume for floppy disks. By convention, slice 7 is thus defined. *M320fmt* generates a warning message if it finds a floppy slice other than 7.

The *dinit*(1M) utility invokes *m320fmt*(1M), a disk formatter for MVME320 devices. *Dinit* enters an interactive mode and prompts the user for bad track entries. Check the Winchester verification report supplied by the disk manufacturer for a list of bad blocks or imperfections on the disk. If no bad blocks are listed, type a period (.) to terminate the bad track handling phase of disk initialization. The device is assumed to be perfect.

If imperfections are listed on the verification report, you must perform some calculations to convert the information on the report into a form recognized by the utility. The *dinit* utility expects a list of bad tracks. The Winchester verification report lists media imperfections in one of two ways: either the report gives the sector number of the first bad sector on a track, or the report identifies the problem area by head number, cylinder number, and byte offset. To calculate the bad tracks from the information provided on the verification report, use one of the following methods, whichever is appropriate to your disk:

METHOD 1: Calculate Bad Tracks from Sector Numbers

To obtain the bad track numbers, divide each sector number listed in the Winchester verification report by the number of sectors per track. Since all supported drives (Computer Memories, Micropolis, and Vertex) contain 32 sectors per track, the conversion equation becomes:

track number = (sector number) / 32

METHOD 2: Calculate Bad Tracks from Head and Cylinder Numbers

(cylinder number) x (total # of heads) + (head number) = track number

The 15Mb Computer Memories drive and the 40Mb Micropolis drive have 6 heads. The Vertex 40Mb drive has 5 heads.

After *m320fmt* has formatted the device, *dinit* sets up the volume-id and the configuration sectors, records the bad track information, and installs the boot loader on the drive.

(NOTE: If a Winchester disk is formatted without making the required bad track entries, proper operation cannot be guaranteed.)

BUGS

An error in specifying heads or cylinders for a hard disk may result in a disk which appears to be correctly formatted but generates physical I/O errors in high cylinders (bad precomp values) or seems to have "lost" some of its space (surface mapped out). *Dinit*(1M) references a Motorola-prepared file which contains accurate values for these parameters.

SEE ALSO

dinit(1M), m320(7).

NAME

m350ctl – MVME350 control program

SYNOPSIS

m350ctl [–retwg] [–fx] [-s[n[kb]]] [special]

DESCRIPTION

M350ctl controls function of the MVME350 streaming tape device. The following options are interpreted by *m350ctl:*

–r Rewind tape.

–e Erase tape.

–t Retension tape.

–w Open tape for writing (with filemark).

–fx Position tape at the front of file x.

–g Print DMA buffer size.

–sn Set DMA buffer size. The buffer size is set to n bytes, nb (or nB) blocks, or nk (or nK) Kbytes. If n is not specified, the buffer size set to default value of 128 Kbytes. If n is zero, then double buffering is turned off.

If the special file is not given, standard input will be used. For example, the command

```
# m350ctl -e /dev/rmt/m350_0a
```

is identical to

```
# m350ctl -e < /dev/rmt/m350_0a
```

If the –w option is used, the default special file is standard output. Thus, the command

```
# m350ctl -ew /dev/rmt/m350_0t
```

is identical to

```
# m350ctl -ew > /dev/rmt/m350_0t
```

See *mvme350*(7) for more information regarding special file naming conventions.

FILES

/dev/rmt/m350_*

SEE ALSO

mvme350(7).

NAME

mkfs — construct a file system

SYNOPSIS

/etc/mkfs special blocks[:inodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. The command waits 10 seconds before starting to construct the file system. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system occupies. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of *logical* blocks divided by 4.

If the second argument is a filename that can be opened, *mkfs* assumes it to be a prototype file *proto*, and takes its directions from that file. The prototype file contains tokens separated by spaces or newlines. The first token is the name of a file to be copied onto block zero as the bootstrap program (see *ops.macs*(8)). The second token is a number specifying the size of the created file system in *physical* disk blocks. Typically, it is the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of inodes in the file system. The maximum number of inodes configurable is 65500. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6-character string. The first character specifies the type of the file. (The characters **—bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or — to specify set-user-id mode or not. The third is **g** or — for the set-group-id mode. The rest of the mode is a three-digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token is a pathname from which the contents and size are copied. If the file is a block or character special file, two decimal number tokens follow, which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/stand/diskboot
4872 110
d——777 3 1
usr     d——777 3 1
        sh      ———755 3 1 /bin/sh
        ken     d——755 6 1
                $
        b0      b——644 3 1 0 0
        c0      c——644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *blocks* per *cycle* can be specified. Default values are 7 for gap size and 400 blocks per cycle. The default will be used if the supplied *gap* and *blocks/cycle* are considered illegal values or if a short argument count

occurs.

SEE ALSO

chmod(1), dir(4), fs(4), bo.macs(8), bo.vme(8), ops.macs(8).
"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

BUGS

If a prototype is used, it is not possible to initialize a file larger than 64Kb, nor is there a way to specify links.

NAME

mknod — build special file

SYNOPSIS

/etc/mknod name **c** | **b** major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding inode for a special file. The first argument is the *name* of the entry. In the first case, the second is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device, e.g., unit, drive, or line number, which may be either decimal or octal.

The assignment of major device numbers is specific to each system and found in the system source file: **conf.c.**

Mknod can also be used to create fifo's (pipes). (See the second case in the above *SYNOPSIS*.)

SEE ALSO

mknod(2).

NAME

mnt, umnt – mount and dismount file system

SYNOPSIS

/mot/bin/mnt [*name* [*directory*]] [**–r**]

/mot/bin/umnt [*name*]

DESCRIPTION

Mnt (umnt) has an optional argument, *name*. This argument is used to search the *permissions* file to determine the real device to mount (unmount). The file is searched and when *name* matches either the *real_device* or the *alias* entry on a line, the *real_device* entry is then used as the special device to be mounted (unmounted).

The default value for *alias* is **default**. The default value for *directory* is whatever directory is listed in the *mntpt* entry of the *permissions* file.

Mnt announces to the system that a removable file system is present on the special device. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

The optional last argument indicates that the file is to be mounted read-only. Write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted. If the user has not been granted *Write* permission for the alias (as controlled by the *perms* entry of the *permissions* file), the device will be mounted read-only.

Umnt announces to the system that the removable file system previously mounted on a special device is to be removed.

By convention *mount*(1M) and *umount*(1M) require root permission to execute. Normal users must use *mnt* when dealing with mountable media.

FILES

/etc/mnttab	mount table
/etc/mount	
/etc/umount	
/mot/bin/mnt	
/mot/bin/umnt	
/mot/etc/perms	permissions file

SEE ALSO

mount(1M), mnttab(4), perms(4).

DIAGNOSTICS

Mnt issues a warning if the file system to be mounted is currently mounted under another name.

Umnt complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or a user's working directory.

BUGS

The *permissions* file is not checked for read permission, before mounting a disk.

Some degree of validation is done on the file system; however, it is generally unwise to mount garbage file systems.

NAME

mount, umount – mount and dismount file system

SYNOPSIS

/etc/mount [special directory [**–r**]]

/etc/umount special

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special*. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

These commands maintain a table of mounted devices. If invoked with no arguments, *mount* prints the table.

The optional last argument indicates that the file is to be mounted read-only. Write-protected and magnetic tape file systems must be mounted in this way or errors will occur when access times are updated, whether or not any explicit write is attempted.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

FILES

/etc/mnttab mount table

SEE ALSO

setmnt(1M), mount(2), mnttab(4).

DIAGNOSTICS

Mount issues a warning if the file system to be mounted is currently mounted under another name.

Umount complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or a user's working directory.

BUGS

Some degree of validation is done on the file system; however, it is generally unwise to mount garbage file systems.

NAME

mvdir — move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

Mvdir renames directories within a file system. *Dirname* must be a directory; *name* must not exist. Neither name may be a sub-set of the other (**/x/y** cannot be moved to **/x/y/z**, nor vice versa).

Only superuser can use *mvdir*.

SEE ALSO

mkdir(1).

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/etc/ncheck [–i numbers **] [–a] [–s] [** file-system **]**

DESCRIPTION

Ncheck generates a list of pathnames and i-numbers of all files on a set of default file systems. Names of directory files are followed by /.. The **–i** option reduces the report to only those files whose i-numbers follow. The **–a** option allows printing of the names . and .., which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order. The user can pipe *ncheck* to the *sort*(1) utility to obtain the report in a specified order. For example,

ncheck *file_system* **| sort –n –onchecklist**

will produce a list which sorts the i-node numbers numerically; or

ncheck *file_system* **| sort +l –f –onchecklist**

will produce a list which sorts files (in the second column of output) alphabetically by name.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

When the file system structure is improper, ?? denotes the "parent" of a parentless file, and a pathname beginning with ... denotes a loop.

PROFILER(1M)

NAME

prfld, prfstat, prfdc, prfsnap, prfpr — operating system profiler

SYNOPSIS

/etc/prfld [namelist]
/etc/prfstat [**on**]
/etc/prfstat [**off**]
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [namelist]]

DESCRIPTION

Prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a group of programs to facilitate an activity study of the operating system.

Prfld initializes the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *namelist*.

Prfstat enables or disables the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* also reveals the number of text addresses being measured.

Prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* stores the counters in *file* every *period* minutes and turns off at *off_hour* (valid values for *off_hour* are **0**—**24**). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

Prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf	interface to profile data and text addresses
/unix	default for namelist file

SEE ALSO

prf(7).

NAME

pwck, grpck — password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and optional program name exist. The criteria for determining a valid login name are derived from "Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*. The default password file is **/etc/passwd.**

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group.**

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).
"Setting up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*.

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

runacct — run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

Runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *Runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes.

Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see *mail*(1)) is sent to **root** and **adm**, and *runacct* terminates. *Runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

Runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *Runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of **chargefee** into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Include any installation-dependent accounting programs here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* reruns the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*:

nohup runacct 2> /usr/adm/acct/nite/fd2log &

To restart *runacct*:

nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &

To restart *runacct* at a specific *state*:

nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(4), utmp(4).
"Accounting" in the *SYSTEM V/68 Administrator's Guide*.

DIAGNOSTICS

The accounting system starts complaining with *****RECOMPILE pnpsplit WITH NEW HOLIDAYS***** after the last holiday of the year. See "Accounting" in the *SYSTEM V/68 Administrator's Guide* for more on how to correct this condition. Other diagnostics are placed in various error and log files.

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it is not complete.

NAME

sadp — disk access profiler

SYNOPSIS

sadp [**—th**] [**—d** device[—drive]] s [n]

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of eight cylinders.

The only valid value of *device* is **disk.** *Drive* specifies the disk drives and it may be: a drive number in the range supported by *device*, two numbers separated by a minus (indicating an inclusive range), or a list of drive numbers separated by commas.

Up to eight disk drives may be reported. The **—d** option may be omitted, if only one *device* is present.

The **—t** flag causes the data to be reported in tabular form. The **—h** flag produces a histogram of the data on the printer. Default is **—t.**

EXAMPLE

The command:

sadp —d disk —0 900 4

generates 4 tabular reports, each describing cylinder usage and seek distance of disk drive 0 during a 15-minute interval.

FILES

/dev/kmem

NAME

sa1, sa2, sadc — system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [—**ubdycwaqvmA**] [—**s** time] [—**e** time] [—**i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for inter-process communications.

Sadc and shell procedures *sa1* and *sa2* are used to sample, save, and process this data.

Sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The **/etc/rc** entry:

```
su sys —c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **/usr/spool/cron/crontabs/sys** (see *cron*(1M)):

```
0 * * * 0,6 /usr/lib/sa/sa1
0 8—17 * * 1—5 /usr/lib/sa/sa1 1200 3
0 18—7 * * 1—5 /usr/lib/sa/sa1
```

produces records every 20 minutes during working hours; otherwise; it is on an hourly basis.

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The **/usr/spool/cron/crontabs/sys** entry

```
5 18 * * 1—5 /usr/lib/sa/sa2 —s 8:00 —e 18:01 —i 3600 —A
```

reports important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;      /* see /usr/include/sys/sysinfo.h */
        int  szinode;   /* current entries of inode table */
        int  szfile;    /* current entries of file table */
        int  sztext;    /* current entries of text table */
        int  szproc;    /* current entries of proc table */
        int  mszinode;  /* size of inode table */
        int  mszfile;   /* size of file table */
        int  msztext;   /* size of text table */
        int  mszproc;   /* size of proc table */
        long  inodeovf; /* cumul. overflows of inode table */
        long  inodeovf; /* cumul. overflows of file table */
        long  textovf;  /* cumul. overflows of text table */
        long  procovf;  /* cumul. overflows of proc table */
        time_t  ts;     /* time stamp, seconds */
        long  devio[NDEVS][4];  /* device info for up to NDEVS units */
#define IO_OPS          0       /* cumul. I/O requests */
#define IO_BCNT         1       /* cumul. blocks transferred */
#define IO_ACT          2       /* cumul. drive busy time in ticks */
#define IO_RESP         3       /* cumul. I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

cron(1M), sag(1G), sar(1), timex(1).

"System Activity Package" in *SYSTEM V/68 Admininstrator's Guide*.

NAME

setmnt — establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the **/etc/mnttab** table (see *mnttab*(4)), which is needed for both the *mount*(1M) and *umount* commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

filesys node

where *filesys* is the name of the file system's *special file* (e.g., "dsk/?s?") and *node* is the root name of that file system. Thus, *filesys* and *node* become the first two strings in the *mnttab*(4) entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M), mnttab(4).

BUGS

Filesys or *node* can be no longer than 10 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

shutdown — terminate all processing

SYNOPSIS

/etc/shutdown

DESCRIPTION

Shutdown is part of the SYSTEM V/68 operation procedures. Its primary function is to terminate all currently running processes in an orderly and cautious manner. The procedure is designed to interact with the operator, i.e., the person who invoked *shutdown*. *Shutdown* may instruct the operator to perform some specific tasks, or to supply certain responses, before execution can resume. *Shutdown* goes through the following steps:

All users logged on the system are notified to log off the system by a broadcasted message. The operator may display his/her own message at this time; otherwise, the standard file-save message is displayed.

If the operator wishes to run the file-save procedure, *shutdown* unmounts all file systems.

The super blocks of all file systems are updated before the system is to be stopped (see *sync*(1)). This must be done before re-booting the system, to insure file system integrity.

DIAGNOSTICS

The most common error diagnostic that occurs is **device busy.** This happens when a particular file system could not be unmounted.

SEE ALSO

mount(1M), sync(1).

NAME

sysdef — system definition

SYNOPSIS

/etc/sysdef [opsys [master]]

DESCRIPTION

Sysdef analyzes the named operating system file and extracts configuration information; this includes all hardware devices as well as system devices and all tunable parameters.

The output of *sysdef* can usually be used directly by *config.68*(1M) to regenerate the appropriate configuration files.

FILES

/unix	default operating system file
/etc/master	default table for hardware specifications

SEE ALSO

config.68(1M), master(4).

BUGS

For devices that have interrupt vectors but are not interrupt-driven, the output of *sysdef* cannot be used for *config*. Because information regarding *config* aliases is not preserved by the system, device names returned might not be accurate.

NAME

tic — terminfo compiler

SYNOPSIS

tic [—**v**[*n*]] file ...

DESCRIPTION

Tic translates *terminfo* files from the source format into the compiled format. The results are placed in the directory **/usr/lib/terminfo.**

The —**v** (verbose) option causes *tic* to output trace information showing its progress. If the optional integer is appended, the level of verbosity can be increased.

Tic compiles all *terminfo* descriptions in the given files. When a **use=** field is discovered, *tic* searches first the current file, then the master file, which is **./terminfo.src.**

If the environment variable **TERMINFO** is set, the results are placed there instead of in **/usr/lib/terminfo.**

Tic has the following limitations: total compiled entries cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/*/* compiled terminal capability data base

SEE ALSO

curses(3X), terminfo(4).

BUGS

Instead of searching **./terminfo.src**, *tic* should check for an existing compiled entry.

NAME

trenter — enter a trouble report

SYNOPSIS

trenter [—s]

DESCRIPTION

Trenter resides on any machine that must submit machine-readable trouble reports to Customer Support. It prompts the user for the data needed to enter the report, and allows for correction of previously entered data, either in-line, or by invoking a text editor. *Trenter* also allows users to specify (in a file) default values for fields that will likely remain constant across reports, such as name, address, and company name. In addition, facilities are provided to assist local administrators in handling trouble report flow on their systems.

Fields and Values

Trouble reports consist simply of fields and associated values. Each field has a *field name*, by which it may be referenced. When invoked, *trenter* prompts for values for the trouble report's fields. The following table lists the prompts that are issued, along with their corresponding *field names*. All fields accept one line of input, except for the problem description, which is a multi-line field, terminated with a line consisting of only a period. The items marked with an asterisk (*) are explained below.

The first nine fields identify the originator of the report.

- Name (NAME) (*)
- Company (CO) (*)
- Phone (PHONE) (*)
- Room Number (ROOM) (*)
- Address (ADDR) (*)
- City (CITY) (*)
- State (STATE) (*)
- Zip Code (ZIP) (*)
- Country (COUNTRY) (*)

The next two fields are AT&T-assigned numbers to identify the customer and the specific site.

- Customer ID (CID) (*)
- Site ID (SID) (*)

The next two fields identify the processor on which the problem occurred.

- CPU Serial Number (CPUNO) (*)
- Machine type (MACH)

The following fields identify the area in which the problem occurred.

- Trouble Report Type (TYPE)
  Valid responses: **doc** (documentation), **enh** (enhancement), **cs** (customer support),**fw** (firmware), **hdw** (hardware), **sw** (software), **unk** (unknown).
- WECo Product Name (PROD)
  Examples: UNIX, BASIC
- Operating System Release (OS_REL) (*)
  The release of SYSTEM V/68 on which the problem occurred.
- Product Release (PROD_REL)
  The release of the product given in reponse to the WECo product prompt. If product is

TRENTER (1M)

unix, this prompt is not issued.

The remaining fields define the body of the trouble report.

- Severity (SEV)
  The severity of the problem (1-4).
- Required Date (RDATE)
  If the severity of the report is 2, the required date for the fix is prompted. The date given must be at least one week from the date of the trouble report.
- Abstract (ABS)
  One-line description of the problem.
- Description (DESC)
  Full description of the problem. Note that description input will not be passed through *nroff*; however, *trenter* will recognize the macros **.ES** and **.EE** (example start, example end), indicating an indented example (these may be nested).
- Attachments (yes or no) (ATT)

If **?** is given in response to a prompt, a message explaining the field will be printed.

If *trenter* receives an interrupt during prompting, the trouble report will be aborted.

After a trouble report has been completed, the user is given an opportunity to edit any data that has been supplied. Next, a reprint of the trouble report just entered may be requested. Finally, the user is asked whether another report is to be entered. If so, the values for the starred items in the field table above will be carried over from the first report.

Editing Field Values

In order to provide editing while responding to prompts, the following *escapes* are recognized on input:

- *—field*
  Return to a field for which data has previously been supplied. If the field name is not specified, return to the previous field. The value already assigned to the field is printed and the user may enter either new data or another editing command.
- **!e**
  Invoke the editor *ed*(1) with any text already supplied for the current prompt in the edit buffer (an alternate editor can be specified; refer to "Specifying Default Values" below).
- **>**
  Move down to the first unfilled field. This is useful, for example, when the — command has been used to fix a single field near the top of the report and the user wishes to quickly return to the point where he/she left off.
- **=field**
  Print the value currently assigned to the given field.
- **??**
  Print a summary of editing functions.

Editing commands are only recognized when they appear at the beginning of the input line; they may be escaped using a backslash (\ ).

Specifying Default Values

Users may provide default values for any fields marked with an asterisk (*) above. . These values are specified in a file **.trdef** in the user's home directory. Entries in this file are of the form:

field=value

where *field* is a field name from the table above.

The editor to be used for field editing can be overridden with a **.trdef** entry be assigning the name of the desired program to the field **EDITOR.**

During prompting, *trenter* will print any values supplied for fields from a **.trdef** file. By default, it will stop at each such field and wait for either a carriage return (indicating confirmation) , an edit command, or new data. If invoked with a **—s** option, *trenter* will print the supplied values, but will not stop for confirmation.

Default values specified in **.trdef** files may be changed, on a per-report basis, using the editing functions described above.

FILES

.trdef	default value file
/usr/spool/trenter	spool directory

AME

upgrade - software product field upgrade utility

ESCRIPTION

The upgrade program is used to install application software products on a computer system running SYSTEM V/68, Release 2 or later.

THIS VERSION OF UPGRADE CANNOT BE USED TO INSTALL SOFTWARE DISTRIBUTED ON MEDIA CREATED WITH OLDER VERSIONS OF CREATE.

XAMPLE

This section describes the interactive dialogue between a user and the upgrade program. Indented text represents the program display screens; indented **boldface** text represents sample user input; *italicized* text represents variable items. Text shown in braces (for example, {Yes, No}) represents a choice of responses to the prompts. Any response that is too long is truncated; you are then prompted to verify what was entered. Text shown in square brackets ([ ]) is commentary that never appears on the screen.

The **upgrade** program can only be run by the superuser (root) or by setting the owner of the **upgrade** program to root and turning the setuid bit on. Put the system into single-user mode and mount the file systems before executing the **upgrade** program. If you choose to execute the **upgrade** program in multi-user mode, be sure that no other users are logged on.

To execute the **upgrade** program, type:

```
# upgrade
```

The screen displays:

```
Software Product Field Upgrade Utility, R3.1
Copyright 1984,86 by Motorola Computer Systems, Inc.


What type of media is this product distributed on?
{Floppy disk, 1.2MB floppy, Tape cartridge, Cartridge disk, 9-track tape}
--> xxx
```

Choose the appropriate device name from the list shown in braces; type the device name (or the first character of the name), followed by RETURN. Pressing only RETURN causes the program to select the first item in the list. An invalid input, such as xxx shown here, produces the following error message:

```
You must enter the first letter of one of the choices and a RETURN,
or just a RETURN to select the first choice.
```

The previous message is then redisplayed.

```
What type of media is this product distributed on?
{Floppy disk, 1.2MB floppy, Tape cartridge, Cartridge disk, 9-track tape}
--> c    [for a cartridge disk]

Mount distribution media volume #1, then hit RETURN...
```

Insert the media into the proper drive and press return.

A temporary directory is usually created within a file system with enough free space to hold the product; by default, **/tmp**, or **/usr/tmp**, is used. If there is not enough space, the **upgrade** program asks for another location. Refer to "Selecting a Temporary Directory," below for details.

If the target device does not have enough space, the following message is displayed, and the program terminates:

```
The file system containing the intended target directory
(target_directory_name) doesn't have room to hold the product!
Delete some files, then invoke Upgrade again.
```

If enough space is available, a subdirectory is created within the temporary directory, "_upgrade_dir," which in most cases already exists. The screen briefly displays:

```
mkdir /temp_dir_name/_upgrade_dir
```

followed by:

```
        Software Product Field Upgrade Utility, R3.1

[1]     This product's name is 'product_name'
        Its release identifier is: release_id
        (Auditing facilities use the name: audit_name)
        It will occupy about nnnn x [ 512 | 1024]-byte disk blocks in a file system.

[5]     The distribution media is a media_type,
        The product uses nnnn blocks of the media,
              and you should have nn volume(s) of this media.

        Upgrade will execute this command string prior to installation:
              command_string_1
[10]    The product will be copied into the temporary directory:
              temp_dir_name
        Then it will be moved into the target directory:
              target_dir_name
        Finally, this command string will be invoked after installation:
[15]          command_string_2

        Continue...
        {Yes, No}
        -->
```

NOTES:

1. The byte count shown in line 4 displays either 512- or 1024-byte blocks depending on which block size the system uses; the **upgrade** program automatically selects the appropriate number for the system.
2. The *media_type* in line 5 corresponds to the media type you selected at the first prompt.
3. The block count (*nnnn*) in line 6 is based on 1024-byte blocks used on the media. It usually does not equal the file system block count shown in line 4.
4. Lines 8 through 15 may vary from what is shown here. If either *command_string_1* or *command_string_2* is empty, nothing is shown. If there is not enough space for a temporary copy of the product, lines 10 through 13 are replaced by the statement:

   The product will be copied straight into the directory:
   target_directory_name

To continue with the **upgrade** program, type **y** or press RETURN in response to the "Continue... {Yes,No}" prompt. A no (**n**) response terminates the program. If you type **y** to continue, the screen displays:

------ PRODUCT UPGRADE IN PROGRESS ------

Any messages produced by pre-installation commands (*command_string_1*) are displayed, followed by:

nnnn blocks transferred from volume #1 to *temp_dir_name* . . .

If more than one volume is needed, the screen displays:

Mount media volume #*n* for '*product_name release_id*',
then hit RETURN . . .

nnnn blocks transferred from volume #2 to *temp_dir_name* . . .

and so on for each volume of distribution media.

When all files have been copied into the temporary directory, they are moved into the target directory.

Now moving files into the target directory.
The name of each file will be displayed as it is copied . . .

Individual file names are displayed on the screen:

file_name_1
:
:
:
:

[An *nnnn blocks* message is also displayed here but should be ignored!]

The temporary directory created earlier is removed, and the screen displays:

rm -rf */temp_dir_name/*_upgrade_dir

A table-of-contents listing for this product is contained in:
'audit_name'

Would you like me to generate audit files for this product installation now?
{Yes, No}
--> **n**

A yes (**y**) response causes the program to generate audit files; for details on auditing, refer to "Performing an Installation Audit," below.

A no (**n**) response produces the following message:

You can use 'upgrade -a' to do the audit later!

For details on using **upgrade -a**, refer to "Performing an Installation Audit," below.

When the upgrade is complete, the screen displays:

Finished upgrading '*product_name (release_id)*'.

Do you wish to upgrade another product?
{No, Yes}
--> [A RETURN is sufficient to terminate program execution.]

SELECTING A TEMPORARY DIRECTORY

This section describes how the **upgrade** program responds when there is not enough space in either **/tmp** or **/usr** (for **/usr/tmp**) to hold a temporary copy of the product. (NOTE: **/tmp** is selected as a temporary directory only if it is a SEPARATE FILE SYSTEM in the system! If it is a directory beneath root, it cannot be selected! Use the **df** command to display all file system names.)

1. This first dialogue occurs if none of the file systems have enough space:

   Does a previous version of this product already exist in your system?
   {No, Yes}
   --> y [If you answer no (**n**), the product cannot be loaded, and the program terminates.]

   In order to install this new version, I'll have to write over
   the old version. Do you want to do this?
   {Yes, No}
   --> y [A no (**n**) terminates the program.]

Is the old version located in the '*target_dir*' directory?
{Yes, No}
--> **n** [Type no (**n**) if the old version of the product's files was moved since it was installed.]

Enter the name of the directory where it is located:
--> **xyz** [Enter a directory pathname.]

This response is checked to be sure that a valid directory was specified. If not, an error message and the previous message are displayed:

'xyz' is not a valid directory!

Enter the name of the directory where it is located:
--> [A valid directory pathname must be entered.]

2. The following dialogue occurs if at least one file system (other than /**tmp** and /**usr**) has enough space to hold a temporary copy of the product.

These file systems have enough room to install this product
(>*blocks needed*):

file_sys_1	*blocks_free*
:	:
:	:

[Other names are shown if applicable.]

For each file system shown, you are prompted:

Do you want to use '*file_sys_n*' for holding a temporary copy of the product?
{No, Yes}
-->

A no (**n**) response or RETURN causes the next possible choice to be offered. If all choices are declined, the following prompt is displayed and the process repeats:

YOU MUST SELECT ONE OF THESE DEVICES!

If you select a file system other than root (/), you can specify a subpath within the selected filesystem.

Enter a subdirectory within this file system, if desired.
--> /*file_sys_name*/**xyz**
[The selected *file_sys_name* is shown as part of the prompt.

RETURN is usually a sufficient response because the name, '/_upgrade_dir,' is appended to the end of the resulting name.

If a nonexistent directory name, such as **xyz** is entered, the screen displays:

'xyz' is not a valid directory!

Enter a subdirectory within this file system, if desired.
--> */file_sys_name/*

PERFORMING AN INSTALLATION AUDIT

If the audit files have not been deleted, an audit can be performed by executing **upgrade -a** once the product has been installed. The **upgrade -a** program asks for the type of media the product is distributed on and for the first volume to be installed. The program then moves directly to the audit section and displays:

--- NOW EXTRACTING CRC'S FROM PRODUCT FILES ---

This may take a minute or two . . . please be patient!

The following files contain auditing data for this product:

The names of three files appear on the screen in **ls -l** format. The information in them will be needed if you ever call the Customer Support Operation for assistance. The files should be printed and saved in a convenient place (for example, with the distribution media), then deleted. They can be reconstructed by executing **upgrade** again.

The three files are named after the *audit_name* identified in the status display: *audit_name*.cr0, *audit_name*.crc, and *audit_name*.ls. For simplicity, they are referred to here as *.cr0, *.crc, and *.ls, respectively. (Note that the *audit_name* is different for each product.)

These files use two different formats. The *.ls file format is similar to that produced by an **ls** command. It contains the name of every file installed as part of the product. The *.cr0 and *.crc files use the following format:

$*crc* *length* *modification time-stamp* *file name*

for each of the files identified in the corresponding *.ls file. The *.cr0 file is created at the factory when the distribution media is created. The *.crc file is created by this part of the program when the **upgrade** program is run or when executing **upgrade -a.** It can also be created by typing:

crc -cdf /usr/AUDITS/*audit_name***.ls >/usr/AUDITS/***audit_name***.crc**

The first three fields of the *.cr0 and *.crc files should be identical for every corresponding file name (see NOTE below). The fourth columns should differ only in their path prefixes; that is, the *.cr0 file might display its file names as **/x/y/example**, while the *.crc file shows **/a/b/c/example** for every file **example**. This causes corresponding *.cr0 and *.crc files to have different lengths, even when the data in the first three columns is identical for each entry.

NOTE: Due to the operation of **cpio**, the *.ls and *.crc files may contain a few more entries than their corresponding *.cr0 file. If present, these entries represent DIRECTORY NAMES rather than file names.

A short shell function, **audit**, can be used to perform a simple audit of these files. To create **audit**, type the following lines after a shell prompt.

```
# audit() {
> sort /usr/AUDITS/$1.cr? |
> awk 'BEGIN { FS = "\t" }
> { print $4 "\t" $1 "\t" $2 "\t" $3 } ' |
> uniq -u -1  [Use the digit one, not the lowercase letter L.]
> }
#
```

This shell function is executed as if it were a shell script and is stored in memory rather than in a file. It disappears after you log off, and cannot be exported to nested shells. It is invoked by typing **audit** *audit_name* to the shell, where *audit_name* is the name given in line 3 of the status display (see page 2). For example, **audit** can be used to audit the files for a product named "menus" by typing:

```
# audit menus
#
```

If the **upgrade** installation executes properly, this command returns a shell prompt. Any file that doesn't transfer properly has its name, crc, length, and modification date displayed. If just *one* line for each name appears, it will usually be from the *.cr0 file and indicate that the corresponding file was not installed in the system properly or was erased. If *two* lines for each name appear, then one or more of the last three fields differs. Generally, it will be the modification time-stamp field that differs. However, if an error occurred during installation, the crc and/or length fields may differ; in this case, do a complete reinstallation.

DIAGNOSTICS

Most parameters are validated before being used, either by the program or by the user via the status display. It is possible, however, for an internally executed shell command to die, in which case an error message may or may not be produced. Known error conditions fall into two categories: warnings and fatal errors. Warnings produce a message but allow for continued execution. Fatal errors prevent complete installation from the distribution media. Warning and fatal error messages are listed below.

If the **/usr/bin/crc** program is missing, abort **upgrade** and rerun after the **crc** program is located and installed.

Warnings

The checksums don't match for this volume!

error encountered while reading checksum from distribution media!

popen failed; can't determine available space!

can't accurately determine block-size of target device!

Fatal Errors

You must be logged in as root or have root's setuid permission set!

There is NOT SUFFICIENT SPACE on your system to install this product!

I can't read this media! (incompatible internal structure)
The media was created with an earlier version of the create program.

Can't open device {*dev_name*} for reading!
Verify that the virtual device names are linked properly:

/dev/FLOPPY	for raw floppy device (640K bytes)
/dev/FLOPPY.MB	for raw 1.2MB floppy device
/dev/IOMEGA	for raw 5MB Iomega cartridge disk
/dev/TAPE.CART	for cartridge tape drive
/dev/TAPE.9TRK	for 9-track tape drive

You'll have to (re)install this product before auditing!
Some audit files are missing. Run **upgrade** again without the **-a** switch.

can't read header from distribution media!

error reading from distribution media!

error writing to cpio data stream!

can't **popen** output stream!

FILES

tput(1)
/usr/bin/crc

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options]

DESCRIPTION

Uuclean scans the spool directory for files with the specified prefix and deletes all those which are older than the specified number of hours.

The following options are available.

–d*directory* Cleans *directory* instead of the spool directory.

–p*pre* Scans for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following causes all files older than the specified time to be deleted.

–n*time* Deletes files whose age is more than *time* hours, if the prefix test is satisfied. (default time is 72 hours)

–w*file time* Finds files which are older than *time* hours; however, the files are not deleted. If the argument *file* is present, the warning is placed in *file*; otherwise, the warnings go to the standard output.

–s*sys* Examines only files destined for system *sys*. Up to 10 **–s** arguments may be specified.

–m*file* Sends mail to the owner of the file when it is deleted. If a *file* is specified, then an entry is placed in *file*.

This program is typically started by *cron*(1M).

FILES

/usr/lib/uucp (directory with commands used by *uuclean* internally)

/usr/spool/uucp (spool directory)

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uusub — monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

Uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

—a*sys* Add *sys* to the subnetwork.
—d*sys* Delete *sys* from the subnetwork.
—l Report the statistics on connections.
—r Report the statistics on traffic amount.
—f Flush the connection statistics.
—u*hr* Gather the traffic statistics over the past *hr* hours.
—c*sys* Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g., ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g., line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the **—u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

The command:

uusub —c all —u 24

is typically started by *cron*(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L_sub	connection statistics
/usr/lib/uucp/R_sub	traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

vm22fmt — format disks on the VM22 disk controller

SYNOPSIS

/etc/vm22fmt [—**h** starthead] [—**t** trkcyl] [—**e** sectrk] [—**r** steprate] [—**a** attr] [—**g** gpl3] [—**l** spiral] [—**c** cyls] [—**p** precomp] [—**s** secsiz] [—**x**] [—**F**] special

DESCRIPTION

Vm22fmt is used to format disks on the VM22 disk controller. This disk controller requires the following information to format the disk:

- starting head number of the drive.
- tracks per cylinder.
- drive step rate.
- drive I/O attribute byte.
- drive GPL3 code.
- spiral offset to use.
- cylinders per drive.
- precompensation cylinder.
- sector size in bytes per sector.

Each of these parameters can be set by the appropriate command option. The options are:

—**h** Starting head number. All drives have starting head number zero except the fixed disk on LMD (Lark) and CMD drives. Default: 0. The fixed LMD drive has starting head number 2. The fixed CMD drive has starting head number 16.

—**t** Tracks per cylinder (heads). Default: 2. LMD disks all have 2 heads. All 16Mb CMD drives have 1 head. The 80Mb CMD drive has 5 heads. Single-sided floppies have 1 head while double-sided ones have 2.

—**e** Sectors per track. Default: 64. All hard disks have 64 sectors per track. The 8-inch floppies have 26 sectors per track while 5¼ inch floppies have 16 sectors per track.

—**r** Step rate. A step rate of zero informs the VM22 to use its built-in default step rate for that drive. Default: 0. Only specify a different step rate after carefully studying the VM22 User's Manual.

—**a** Attribute byte. The attribute byte is used during normal read and write commands to the VM22. Default: 9. The bits of the attribute byte are shown in the table at the end of this section (refer also to the VM22 User's Manual).

—**g** VM22 GPL3 parameter (floppy only). This parameter determines the number of bytes inserted after the data field on each sector to compensate for motor speed variation. If this parameter is zero, the controller will use the correct default for the present configuration. Default: 0. Only specify a different gpl3 after carefully studying the VM22 User's Manual.

—**l** Spiral offset. Defines the physical sector in which the first logical sector of a track appears. Default: 0.

—**c** Cylinders. Default: 624. The LMD drives have 624 cylinders. The CMD drives have 823 cylinders. The 8-inch floppies have 77 cylinders. The 5¼-inch floppies have 80 cylinders.

—**p** Precompensation cylinder (Floppy only). Default: 0. The precompensation cylinder is usually set to be the number of cylinders divided by two.

—**s** Sector size. Default: 256. On floppies, the sector size for single-density is 128 bytes. Double-density sector size is 256 bytes.

In addition, the —**x** option will print out example command line options for various VM22 compatible disk drives and a list of the default settings. Lastly, the —**F** option tells *vm22fmt*

VM22FMT (1M)

to set the disk configuration but not format the disk. This option is used when the disk configuration block on the disk is incorrect and needs to be overwritten. When formatting floppies, *vm22fmt* will always format the first track of the floppy as single-density, 128 bytes per sector. This track is used for holding the disk configuration information. The default settings are for a CDC 9457 25Mb LMD (Lark) cartridge.

BITS OF ATTRIBUTE BYTE

BIT #	NAME	USE
7	FTD	Floppy Track Density (Floppy Only). if 0 - drive and media density the same. if 1 - media is 1/2 drive density (i.e., 96 TPI drive and 48 TPI media).
6	FSN	Floppy Sector Numbering (Floppy Only). if 0 - sectors number 1-N on both sides (IBM). if 1 - Motorola format, 1 to N on side 0 and N+1 to 2N on side 1.
5	FMF	Floppy recording method (Floppy Only). if 0 - FM (single-density). if 1 - MFM (double-density).
4,3	SSC	Sector Size Code. if 00 - 128 bytes per sector. if 01 - 256 bytes per sector. if 10 - 512 bytes per sector. if 11 - 1024 bytes per sector.
2	FDS	Floppy Diskette Sides (Floppy Only). if 0 - single-sided. if 1 - double-sided.
1	FDR	Floppy Data Rate (Floppy Only). if 0 - 5¼-inch data rate (250 kilobits per second). if 1 - 8-inch data rate (500 kilobits per second).
0	IBS	Imbedded Servo Drive (SMD only). if 0 - Drive does not require a seek when a head switch is performed. if 1 - Drive requires a seek when a head switch is performed. (Used for LARK and LARK-compatible drive.)

FILES

/dev/dsk/*
/dev/rdsk/*
/etc/diskdefs
/etc/diskalts/vm22*

SEE ALSO

dinit(1M), cm16(7), cm80(7), f18(7), lrk(25), vm22(7).
Storage Module Drive Disk Controller User's Manual, M68KVM22/D1

VM22FMT (1M)

DIAGNOSTICS

DIAGNOSTIC	PROBLEM
Can't stat <special>	Special file given on command line nonexistent. Create file.
<special> not a raw disk	Special file given on command line must be a character special file. Be sure the correct file is being used.
Can't open <special>	Could not open special file given on command line for read/write. Check modes of special file.
vm22fmt: configuration error	Could not successfully set the drive configuration. Check command line options for incompatible configuration information (e.g., if the sector size information in the attribute byte disagrees with the sector size option, then the configuration will fail).
vm22fmt: format unit failure	The "format unit" command to format the entire disk failed. The system console should have an error message on it that will indicate the nature of the problem. That message is generated by the disk driver's error message routine. Refer to the *SYSTEM V/68 Error Message Manual* for more information.
vm22fmt: format track failure	The "format track" command to format the first track failed. The system console should have an error message on it that will indicate the nature of the problem. That message is generated by the disk driver's error message routine. Refer to the *SYSTEM V/68 Error Message Manual* for more information.

BUGS

The default setting on the configuration items is for a 25Mb LMD cartridge. When formatting any other media type, be careful to supply the correct options. Use the **—x** option for some examples; also, refer to the VM22 User's Manual for more details.

NAME

volcopy, labelit — copy file systems with label checking

SYNOPSIS

/etc/volcopy [options] fsname special1 volname1 special2 volname2

/etc/labelit special [fsname volume [**—n**]]

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

- **—a** invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made,
- **—s** (default) invoke the **DEL if wrong** verification sequence.

Other *options* are used only with tapes:

- **—bpi**density bits-per-inch (i.e., **800/1600/6250**),
- **—feet**size size of reel in feet (i.e., **1200/2400**),
- **—reel**num beginning reel number for a restarted copy,
- **—buf** use double buffered I/O.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* prompts for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g., *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (e.g., **root**, **u1**) of the file system being copied.

The *special* argument should be the physical disk section or tape (e.g., **/dev/rdsk/***cntrlr***_1s5** or **/dev/rmt/***cntrlr***_0m**).

The *volname* is the physical volume name (e.g., **pk3**, **t0122**, etc.) and should match the external label sticker; such label names are limited to six or fewer characters. *Volname* may be — to use the existing volume name.

Special1 and *volname1* are the device and volume from which the copy of the file system is being extracted. *Special2* and *volname2* are the target device and volume.

Fsname and *volume* are recorded in the the superblock (**char fsname[6], volname[6];**).

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, *labelit* prints current label values. The **—n** option provides for initial labeling of new tapes only (this destroys previous contents).

FILES

/etc/log/filesave.log (a record of file systems/volumes copied)

SEE ALSO

fs(4).

BUGS

If the **—buf** option is selected, *volcopy* will core dump if it gets to the end-of-tape.

NAME

wall — write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends the message to all currently logged in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be superuser to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

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

NAME

wffmt — format floppies for the VME/10 processor

SYNOPSIS

wffmt device

DESCRIPTION

The *wffmt* utility is used to format 5¼-inch floppy diskettes for the VME/10. The user specifies a raw device that is a floppy drive. In the basic configuration, this will be **/dev/rdsk/***cntrlr***_2s7**. If a second floppy drive is connected, it has the device name **/dev/rdsk/***cntrlr***_3s7**. When the command

wffmt /dev/rdsk/*cntrlr***_2s7**

is given, the following message appears on the screen:

Formatting floppy in /dev/rdsk/*cntrlr***_2s7**

When the SYSTEM V/68 prompt appears on the screen, formatting is complete.

Wffmt checks to make sure that a correct device name is specified. The following diagnostic messages may appear if the command is incorrect:

Cannot stat/access /dev/*device name*	The device specified is not connected or permissions are incorrect.
Not a floppy drive.	The *number* given was not **rdsk/***cntrlr***_2s7** or **rdsk/***cntrlr***_3s7.**
Must use a raw/character device.	The correct specification is **rdsk/**, not **dsk/.**

FILES

/etc/wffmt

NAME

whodo — who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

SEE ALSO

ps(1), who(1).

NAME

intro — introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and SYSTEM V/68 device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding SYSTEM V/68 device driver are discussed where applicable.

SYSTEM V/68, Release 2, Version 1 incorporates a new convention for naming disk and tape devices. In earlier releases, the standard format for naming disk devices was **/dev/dk***xy* where *x* referred to the disk device number and *y* referred to the disk section or partition. Raw access to a disk device was indicated with an **r**, e.g., **/dev/rdk01**. Standard format for naming tape devices was **/dev/mt***x* where *x* referred to the magnetic tape device number. Raw access to a tape device was indicated with an **r**, e.g., **/dev/rmt0**.

The new naming convention creates separate subdirectories under **/dev** for each type of disk or tape device. The new format for disk devices is:

/dev/ {**r**} **dsk/** [**r**] [*cntrlr_*] [*controller_number***d**] *drive_number***s***section_number*

Fields in square brackets are entirely optional: they do not affect the operation of any software or hardware; they are for informational purposes only, for the convenience of administrators, operators, and users. Fields in curly brackets represent options that affect software; they must be present if that option is being selected.

r (Not Required) (The first r) indicates a raw interface to the disk. The default is normal system buffering.

dsk/ (Required) Indicates that the device is a disk.

r (Not Required) (The second r) indicates that this disk is on a remote system.

*cntrlr_* (Not Required) Indicates the appropriate disk device in systems with multiple disk drivers. In Release 2, Version 1 of SYSTEM V/68, the controller names are present and must be used on command lines that specify a disk device. The disk devices available are:

vm21_ Intelligent Universal Disk Controller, M68KVM21

vm22_ Intelligent SMD Disk Controller with Floppy Disk, M68KVM22

wd_ Winchester Disk Controller, M68RWIN1

ud_ General Universal Disk Controller

c Generic Controller

*controller_number***d** (Not Required) System administrators decide whether or not to specify the controller number in the disk device name. If the controller number is specified, the **d** introduces the drive number.

drive_number (Required) The drive number. The field is free format; there is no default drive number.

s*section_number* (Required) The section number. The field is free format; there is no default section number.

As an example, the name for disk drive 0, section 0 might be **/dev/dsk/vm22_0s0.**

The new format for tape device names is:

/dev/ {r} mt/ [*c**controller_number**d* **]** *drive_number**density* **{ n }**

where:

r	(Not Required) Indicates a raw device. The default is a blocked device.
mt/	(Required) Indicates a magnetic tape device.
*c**controller_number**d*	(Not Required) The **c** introduces the controller number. System administrators decide whether or not to specify the controller number in the tape device name. If the controller number is specified, the **d** introduces the device number.
device_number	(Required) The drive number. The drive number is followed immediately by the density value of the tape.
density	(Required) Tape density must be specified for each drive. The density is indicated with an **h**, **m**, or **l**, where: **h** (high) is a tape density of 6250 bpi **m** (medium) is a tape density of 1600 bpi **l** (low) is a tape density of 800 bpi
n	(Not Required) Indicates no rewind on close. The default condition is to rewind.

As an example, the name for a 6250 bpi magnetic tape drive might be **/dev/mt/0h.**

To ensure a smooth transition, old device names are accepted by the SYSTEM V/68 Release 2 software. However, all documentation and sample shell scripts distributed with this release use the new naming convention. System administrators are encouraged to rename existing devices and incorporate the new names into shell scripts as soon as possible.

The following table compares existing device filenames with the new filenames that will be found in the documentation.

Disk Devices		Tape Devices	
Old Disk Name	New Disk Name	Old Tape Name	New Tape Name
/dev/dk00	**/dev/dsk/*cntrlr*_0s0**	**/dev/mt01**	**/dev/mt/0l**
/dev/dk10	**/dev/dsk/*cntrlr*_1s0**	**/dev/mt5**	**/dev/mt/5mn**
/dev/rdk00	**/dev/rdsk/*cntrlr*_0s0**		

BUGS

While the names of the entries generally refer to vendor hardware names, in certain cases these names are seemingly arbitrary for various historical reasons.

NAME

acia — Asynchronous Communications Interface Adapter

DESCRIPTION

Each line attached to an *acia* behaves as described in *termio*(7). The EXORmacs debug module and up to five quad communications modules (M68KV7) are supported. The line speed of the EXORmacs debug module can be changed under software control (output speed = input speed), while the line speed of the quad communications modules can be changed by hardware strapping. Eight combinations of data bit, stop bit, and parity bit options are supported (refer to *stty*(1)).

FILES

/dev/console /dev/tty*

SEE ALSO

stty(1), termio(7).

NAME

cm16 — 16Mb Cartridge Module Drive for Universal Disk Driver

DESCRIPTION

The files **dsk/***cntrlr*_**0s0** ... **dsk/***cntrlr*_**0s7** refer to sections of the CM16 disk drive 0. The files **dsk/***cntrlr*_**1s0** ... **dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	26336
1	3292	23044
2	6584	19752
3	9876	16460
4	13168	13168
5	16460	9876
6	19752	6584
7	23044	3292

The start address is a block address, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk/** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

ud(7), cm80(7), lrk25(7), f18(7).

NAME

cm80 — 80Mb Cartridge Module Drive for Universal Disk Driver

DESCRIPTION

The files **dsk/***cntrlr*_**0s0 ... dsk/***cntrlr*_**0s7** refer to sections of the CM80 disk drive 0. The files **dsk/***cntrlr*_**1s0 ... dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	26336
1	26336	26336
2	52672	26336
3	79008	13168
4	92176	13168
5	105344	13168
6	118512	13168
7	0	131680

The start address is a block address, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

ud(7), cm16(7), lrk25(7), f18(7).

NAME

cmd16 — 16Mb Cartridge Module Drive for VM21 Driver and VM22 Driver

DESCRIPTION

The files **dsk/***cntrlr***_0s0** ... **dsk/***cntrlr***_0s7** refer to sections of the CMD16 disk drive 0. The files **dsk/***cntrlr***_1s0** ... **dsk/***cntrlr***_1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces. This new slicing also allows space on the disk for replacement tracks to be used by the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	7	25984
1	107	22784
2	207	19584
3	307	16384
4	407	13184
5	507	9984
6	607	6784
7	0	26336

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd80(7), lark25(7), lark8(7), sa800f121(7), sa800f122(7), vm21(7), vm22(7).

NAME

cmd80 — 80Mb Cartridge Module Drive for VM21 Driver and VM22 Driver

DESCRIPTION

The files **dsk/***cntrlr***_0s0 . . . dsk/***cntrlr***_0s7** refer to sections of the CMD80 disk drive 0. The files **dsk/***cntrlr***_1s0 . . . dsk/***cntrlr***_1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces. This new slicing also allows space on the disk for replacement tracks to be used by the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	6	130080
1	126	110880
2	333	77760
3	495	51840
4	576	38880
5	657	25920
6	738	12960
7	0	131680

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), lark25(7), lark8(7), sa800f121(7), sa800f122(7), vm21(7), vm22(7).

NAME

err — error-logging interface

DESCRIPTION

Minor device 0 of the *err* driver is the interface between a process and the system's error-record collection routines. The driver may be opened only for reading by a single process with superuser permissions. Each read causes an entire error record to be retrieved; the record is truncated if the read request is for less than the record's length.

FILES

/dev/error special file

SEE ALSO

errdemon(1M).

NAME

fl8 — 8-inch Floppy Disk Drive for Universal Disk Driver

DESCRIPTION

Although it is possible to have 8 devices for each floppy disk drive, only two devices per floppy disk drive are currently defined: **dsk/***cntrlr_x***s0** and **dsk/***cntrlr_x***s1**

Dsk/*cntrlr_x***s0** is defined for single-sided drives or single-sided disks in double-sided drives. **Dsk/***cntrlr_x***s1** is defined for double-sided disks in double-sided drives.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	500
1	0	1000
2	-	-
3	-	-
4	-	-
5	-	-
6	-	-
7	-	-

The start address is a block address with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

ud(7), cm16(7), cm80(7), lrk25(7).

NAME

lark8 — 16Mb LARK Module Drive for VM21 Driver and VM22 Driver

DESCRIPTION

The files **dsk/***cntrlr*_**0s0** ... **dsk/***cntrlr*_**0s7** refer to sections of the LARK25 disk drive 0. The files **dsk/***cntrlr*_**1s0** ... **dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces. This new slicing also allows space on the disk for replacement tracks to be used by the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	4	12736
1	29	11136
2	54	9536
3	79	7936
4	104	6336
5	129	4736
6	154	3136
7	0	13184

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark8(7), sa800f121(7), sa800f122(7), vm21(7), vm22(7).

NAME

lark25 — 50Mb LARK Module Drive for VM21 Driver and VM22 Driver

DESCRIPTION

The files **dsk/***cntrlr***_0s0 ... dsk/***cntrlr***_0s7** refer to sections of the LARK25 disk drive 0. The files **dsk/***cntrlr***_1s0 ... dsk/***cntrlr***_1s7** refer to section of drive **1.** This slicing allows the pack to be broken up into more manageable pieces. This new slicing also allows space on the disk for replacement tracks to be used by the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	15	38144
1	90	33344
2	165	28544
3	240	23744
4	315	18944
5	390	14144
6	465	9344
7	0	39936

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark8(7), sa800f121(7), sa800f122(7), vm21(7), vm22(7).

NAME

lp — MVME410 line printer interface

DESCRIPTION

Lp provides the interface to any of the standard Printronix-type line printers or to any of the standard Centronics line printers. When opened or closed, a suitable number of page ejects are generated. Bytes written are printed.

An internal parameter within the driver determines whether or not the device is treated as having a 96- or 64-character set. In half-ASCII mode, lowercase letters are turned into uppercase letters and certain characters are escaped according to the following table:

{	(-
}	-)
`	'
|	+
~	^

The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed. The default line length is 132 characters, indent is 4 characters and lines per page is 66. Lines longer than the line length minus the indent (i.e., 128 characters, using the above defaults) are truncated.

Two *ioctl*(2) system calls are available:

#include <sys/lprio.h>
ioctl (*fildes, command, arg*)
struct lprio **arg;*

The commands are:

LPRGET
Get the current indent, columns per line, and lines per page and store in the *lprio* structure referenced by *arg*.

LPRSET Set the current indent, columns per line, and lines per page from the structure referenced by *arg*.

Thus, indent, page width, and page length can be set with an external program.

FILES

/dev/lp*

SEE ALSO

lp(1), ioctl(2).

NAME

lp050 – MVME050 line printer interface

DESCRIPTION

SYSTEM V/68 supports the parallel port on the MVME050 System Controller Module as the printer port. *Lp050* provides the interface to any of the standard Printronix-type parallel line printers or the standard Centronics-type parallel line printers. When the device is opened or closed, a suitable number of page ejects are generated. Bytes written are printed using a buffered interface.

The driver supports the printable ASCII character set (96 characters) and correctly interprets carriage returns, backspaces, tabs, and form-feeds. The defaults for line length, indent, and lines per page are 132, 4, and 66, respectively. Lines longer than the line length minus the indent (i.e., 128 characters, using the above defaults) are truncated. These defaults can be accessed and modified with an external program using the following calls.

The two *ioctl*(2) system calls available are of the following form:

#include <sys/lprio.h>
ioctl(*fildes, command, arg***)**
struct lprio ****arg*;**

The commands are:

LPRGET Get the current indent, columns per line, and lines per page and store in the **lprio** structure referenced by *arg*.

LPRSET Set the current indent, columns per line, and lines per page from the structure referenced by *arg*.

This driver does not support unbuffered parallel-to-serial-port conversion utilizing the parallel port on the MVME050 module.

FILES

/dev/lp*

SEE ALSO

lp(1), ioctl(2), mvme050(7), m050(7).
MVME050 System Controller Module and MVME701 I/O Transition Module User's Manual.

NAME

lrk25 — 25Mb LARK Module Drive for Universal Disk Driver

DESCRIPTION

The files **dsk/***cntrlr*_**0s0** ... **dsk/***cntrlr*_**0s7** refer to sections of the LRK25 disk drive 0. The files **dsk/***cntrlr*_**1s0** ... **dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the pack to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	0	39936
1	4992	34944
2	9984	29952
3	14976	24960
4	19968	19968
5	24960	14976
6	29952	9984
7	34944	4992

The start address is a block address, with each block containing 512 bytes. Since there is overlap, it is unwise for all of these files to be present in one installation.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

ud(7), cm16(7), cm80(7), f18(7).

NAME

m050 – MVME050 Serial Port

DESCRIPTION

Each MVME050 board supports two devices. Each line attached to an *m050* behaves as described in *termio*(7). The line speed of each device can be changed under software control (output speed = input speed). The number of data bits (5, 6, 7, or 8), parity (even, odd, or no), and the number of stop bits (1 or 2) are also software selectable (refer to *stty*(1)).

FILES

/dev/mpcc0, /dev/mpcc1

SEE ALSO

stty(1), termio(7).

NAME

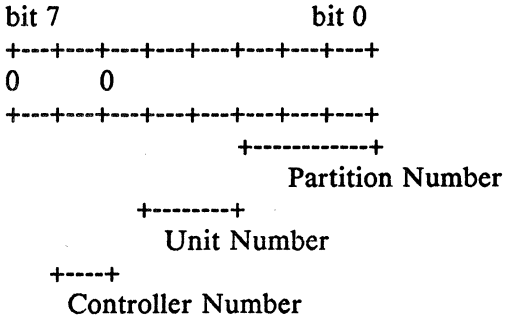
m320 – general disk driver for the MVME320

DESCRIPTION

The *m320* driver provides support for Winchester Disks and 5 ¼" IBM format floppy disks.

Winchester support includes a provision for mapping bad tracks to alternates on the disk. The *dinit*(1M) program must be used to record bad track information for the use of the driver.

The driver interprets the minor device number as follows:



(Zeros required)

The controller number defines one of two MVME320 controllers, 0 or 1.

The unit number defines one of four units on the controller. These assignments are fixed as follows:

unit 0 Winchester Disk

unit 1 Winchester Disk

unit 2 5 ¼" IBM format floppy disk

unit 3 5 ¼" IBM format floppy disk

The partition number (also referred to as a "slice") defines one of eight partitions (0 to 7). The *m320* driver uses the same slicing defined for the *wd*(7) driver.

Two *ioctl*(2) calls are available.

```
int fildes;
int command;
union
{
int blkno;
    struct
    {
        int heads;
        int cylinders;
    }
        volfmt;
}
    arg;
.
.
.
ioctl (fildes, command, &arg);
```

Fildes must be an opened file descriptor (as from *open*(2)) for a raw device.

Command may be **1** or **2**. When *command* is **1**, a single track is formatted. *Arg.blkno* specifies a block in the track to be formatted relative to the origin of the slice opened as *fildes*. When *command* is **2**, the entire volume is formatted. If the target disk is a Winchester, *arg.volfmt.heads* and *arg.volfmt.cylinders* must be the number of heads and cylinders, respectively, on the target disk. This parameter is not required for a floppy disk, but *&arg* must be a valid user memory address.

It is strongly recommended that *dinit*(1M) or *m320fmt*(1M) be used instead of direct calls to *ioctl*(2).

NOTES

A volume format request keeps the controller busy until the format completes.

On IBM format 5 ¼" floppy disks, track 0 contains 4 UNIX (512-byte) blocks (FM recording). Tracks 1 through 159 contain 8 blocks (MFM recording).

FILES

/dev/dsk/m320_*
/dev/rdsk/m320_*
<sys/mvme320.h>
<sys/io/m320io.h>
<sys/io/win.h>
<sys/io/sa400.h>
<sys/space/m320space.h>

SEE ALSO

config(1M), master(4), dinit(1M), m320fmt(1M).

NAME

m400 — MVME400 Dual RS-232C Serial Port Module

DESCRIPTION

Each MVME400 board supports two devices, with a maximum configuration of eight devices (four boards). Each line attached to an *m400* behaves as described in *termio*(7). The line speed of each device can be changed under software control (output speed = input speed). The number of data bits (5, 6, 7, or 8), parity (even, odd, or no) and the number of stop bits (1 or 2) are also software selectable (refer to *stty*(1)).

FILES

/dev/console /dev/tty40*

SEE ALSO

stty(1), termio(7).

NAME

m564 – serial ports on the MVMESYS131

DESCRIPTION

Each MVMESYS131 module supports two serial ports. The lines behave as described in *termio*(7). The line speed of each device can be changed under software control (output speed = input speed). The number of data bits (5, 6, 7, or 8), parity (even, odd, or no), and the number of stop bits (1 or 2) are also software selectable (refer to *stty*(1)).

FILES

/dev/tty00, /dev/tty01

SEE ALSO

stty(1), termio(7).

NAME

mem, kmem — core memory

DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine and even to patch the system.

Byte addresses in *mem* are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

FILES

/dev/mem, /dev/kmem

BUGS

Mem does not access addresses outside of physical ram memory; hence, no device registers are available.

NAME

mvme050 – driver for the MVME050 controller

DESCRIPTION

Mvme050 provides initialization of and an interface to the MVME050 System Controller Module. Separate drivers and appropriate special devices exist for the system clock, two serial ports and parallel port which also exist on the MVME050 board.

Reading one byte from this device returns the current state of the switch bank accessible from the front of the controller. Each bit represents the state of one of the eight switches (ON=1, OFF=0).

Writing one byte to this device will cause the hex value of the byte to be displayed on the front panel display. Refer to the chapter "Operating Instructions" in the MVME050 System Controller Module and MVME701 I/O Transition Module User's Manual for additional information.

Two *ioctl*(2) system calls are available, which are of the form:

#define DSPLY_ON 0
#define DSPLY_OFF 1
ioctl(*fildes, command, arg***)**

The argument *arg* is ignored.

The commands are:

DSPLY_ON Turns on the display.

DSPLY_OFF Turns off the hex display. The default condition of the display is ON.

FILES

/dev/mvme050

SEE ALSO

m050(7), lp050(7), ioctl(2).

NAME

mvme350 – MVME350 Streamer Tape Controller VMEmodule Interface

DESCRIPTION

The MVME350 driver controls one streaming tape drive per controller. It provides advanced read/write access and tape control. Though the user interface is similar to that of 9-track tapes, the QIC-02 Streaming Tape Interface prevents some forms of tape accesses. In particular, streaming tapes may be completely rewritten (truncated) or files may be appended at the *end-of-data*. Unlike 9-track tapes, streaming tapes may not be overwritten in the middle of a tape file (terminated by a filemark). For this reason, the MVME350 driver only supports two types of writing: truncating (the **O_TRUNC** *open*(2) option), and appending (the **O_APPEND** *open*(2) option). If **O_TRUNC** and **O_APPEND** are both missing from an *open*(2) request, the MVME350 driver will supply the appropriate one depending upon the tape's minor number.

To support the "append" and "truncate" functions, two forms of MVME350 special files are defined: *append* devices and *truncate* devices. When an *open*(2) occurs with neither **O_TRUNC** nor **O_APPEND** present, then the file will be opened for truncation on the *truncate* device and for append on the *append* device. If the **O_TRUNC** or **O_APPEND** flags are present, it does not matter whether the *truncate* device or *append* device is used.

CAUTION: the **O_TRUNC** and **O_APPEND** options to *open*(2) **override** the *truncate* and *append* special file designations. Therefore, if a program opens a *truncate* device with the **O_APPEND** option, then data will be appended. If a program opens a *append* device with the **O_TRUNC** option, then the tape will be rewritten. When using an unknown or untried program with the MVME350, first experiment by writing small amounts of information to determine whether the program uses the **O_APPEND** or **O_TRUNC** options, or neither. Some utilities may use both options in different situations; that is, some *open*(2) calls will use **O_APPEND** while others in the same program will use **O_TRUNC**, depending upon context. Most commonly used utilities are discussed below.

Double Buffering.

The MVME350 driver has been implemented with *double buffering* I/O processing. When making an *open*(2) system call on a character (or raw) tape device, the MVME350 will normally allocate two large system buffers to use during I/O. (See the section below regarding *open*(2) processing for exceptions.) These buffers will then be used for all direct memory accesses (DMA) to and from the MVME350 Streaming Tape Controller. When making a *read*(2) system call, the data will first be read into one of the buffers, and then transferred to the reading program. When make a *write*(2) system call, the data will first be transferred from the writing program into one of the buffers. Only when the buffer is full will it be written to a streaming tape.

The advantage of this *double buffering* scheme is that most MVME350 DMA transfers will be done very efficiently, thus keeping the tape "streaming" and wasting little or no space on the tape due to streaming tape underruns. The default buffer size of the double buffers is 128 Kbytes for each buffer.

When reading using the double buffers, the double buffering software will attempt to read ahead of the current read point. Therefore, when the first buffer is finally exhausted, it's likely that the second buffer has already been filled. In this case, the new *read*(2) request may be completed immediately and a new read ahead may begin. The double buffering scheme will then stay ahead of the program that is reading the tape, and provide enough I/O overlap to permit efficient tape reading.

When writing using the double buffers, the double buffering software will accept new *write*(2) requests until the current buffer is full. The software will then write out the full buffer and switch to the other buffer to accept more output. If the other buffer has not yet

been completely written to tape, then the request will hold up the writing program until the I/O request is completed.

The double buffering software may be permanently disabled, or the double buffering buffer sizes may be changed, using the *m350ctl*(1M) control program.

Open(2) Processing.

When the *open*(2) system call is made on an MVME350 streaming tape, the following processing will occur:

1. If the minor device number is illegal, then the *open*(2) will fail, returning the error status **ENXIO.**
2. If the tape unit is already open, then the *open*(2) will fail, returning the error status **ENXIO.**
3. If the tape unit is not "on line", and if an *open*(2) call is made without the **O_NDELAY** option, then the system call will fail, returning the error status **ENXIO.** If an I/O request is made prior to the device coming "on line", then a device I/O error will occur.
4. If the tape unit is being opened for writing and the tape is write-protected, then the *open*(2) will fail, returning the error status **ENXIO.**
5. If the *open*(2) call is made with the **O_NDELAY** option, then no DMA buffers will be allocated and all character I/O will occur unbuffered by the MVME350 driver.
6. If the *open*(2) call is not made with the **O_NDELAY** option, and if the driver is unable to allocate DMA buffers on a character device open, then the *open*(2) will fail, returning the error status **ENXIO.**
7. When a tape is being opened for writing, it must be opened for *rewrite* or for *append*. If the open request is made with the **O_TRUNC** flag (to truncate), the tape will be rewound and the tape will be rewritten. If the open request is made with the **O_APPEND** flag (to append), the tape will be positioned following the last data written onto the tape (*end-of-data*). If the open request is not made with either **O_TRUNC** or **O_APPEND**, the device minor number will be used to determine which one to use. If the open request is made with both **O_TRUNC** and **O_APPEND, O_TRUNC** will override and the tape will be rewritten.

Close(2) Processing.

The following processing will occur upon a *close*(2) of a tape device:

1. If the device was opened for writing, a filemark will be written onto the tape.
2. If the special file indicated that the tape needs to be rewound, then the tape will be rewound ("rewind on close").
3. If the special file indicated that the tape is not to be rewound, and the device was opened for reading, then the tape will be positioned following the next filemark.

When a character (raw) device is closed, the double buffers (if any) will normally be deallocated. The only exception to this rule occurs when an *end-of-media* is encountered while writing a streamer tape. This condition occurs when the program attempts to write off the end of a tape. The operator is notified of the condition by a message printed on the system console. The MVME350 driver will then retain the double buffers and all data currently residing in them (which is not yet written to tape). If the same process then makes an *open*(2) request (for writing) on the same tape, then all remaining data in the double buffers will be written to tape **prior** to any new I/O requests. In this way, programs like *cpio*(1) may be used to write files that span more than one tape. If any other process opens the device next, or if the device is not opened for writing, then the double buffers will be flushed (with

an error message printed on the system console).

In all cases, the MVME350 driver will not complete *close*(2) processing until all streaming tape operations are done. Thus, the streaming tape should not be ejected before the program that is using the streaming tape is done.

Ioctl(2) Processing

The MVME350 driver supports several *ioctl*(2) functions on the character or raw device. These functions permit control functions beyond the normal *open*(2), *close*(2), *read*(2), and *write*(2) system calls. The following functions are supported:

M350REWIND
: rewinds the tape.

M350ERASE
: erases and then rewinds the tape.

M350RETENSION
: retensions the tape (needed whenever a tape has not been used for some time).

M350WRTFM
: writes a filemark onto the tape. A filemark can only be written immediately after data has been written using the *write*(2) system call.

M350RDFM
: reads the tape (and discards the data) up to and including the next filemark.

M350SETDMA
: sets the character device DMA buffering size. The *ioctl*(2) argument parameter is the new buffer size. A buffer size of zero disables double buffering. Only the superuser may set the DMA buffer size.

M350GETDMA
: returns the character device DMA (double) buffering size. The *ioctl*(2) argument parameter is the address (in the user program) of the memory location to put the DMA buffering size. If double buffering is currently enabled, this function returns the size of the smallest of the two buffers.

Human Interfaces.

The filenames used for the MVME350 incorporate the drive number, whether the device is a "rewind-on-close" device, and the DEFAULT write action. The general form for MVME350 file names is:

/dev/{r}mt/m350_#[ta]{n}

where:

{r}	is optional **r**
#	is the drive number
[ta]	is one of **t** or **a**
{n}	is optional **n**

If the optional **r** is present, then the device is the raw (or character) interface. If the **t** is present, and if the tape is opened for writing with neither **O_TRUNC** nor **O_APPEND** specified, then the tape will be opened with **O_TRUNC**. Similarly, if the **a** is present, and if the tape is opened for writing with neither **O_TRUNC** nor **O_APPEND** specified, then the tape will be opened with **O_APPEND** (called the *append* device). If the **n** is present, then the tape will not be rewound when the tape is closed. The filenames to use for the MVME350 block and character (raw) devices are:

	BLOCK DEVICES	
	REWIND ON CLOSE	NO REWIND ON CLOSE
Truncate Device	/dev/mt/m350_Ct	/dev/mt/m350_Ctn
Append Device	/dev/mt/m350_Ca	/dev/mt/m350_Can

	CHARACTER (OR RAW) DEVICES	
	REWIND ON CLOSE	NO REWIND ON CLOSE
Truncate Device	/dev/rmt/m350_Ct	/dev/rmt/m350_Ctn
Append Device	/dev/rmt/m350_Ca	/dev/rmt/m350_Can

where: C is the controller number to which the tape drive is attached.

For example, to use the *dd*(1) command to copy a filesystem from disk **0s0** to the tape on the MVME350 controller zero, rewrite the tape, and to rewind the tape when done, execute:

$ dd if=/dev/rdsk/0s0 of=/dev/rmt/m350_0t

This command will perform a "raw" copy of the filesystem on disk **0s0** to the tape. The tape driver will prohibit accidental sharing of a tape drive by only permitting one open per drive at a time.

Tape Usage.

Due to the *append* or *truncate* restrictions on writing a tape, the normal tape compatible utilities need some special handling. If the tape is being accessed through I/O redirection in the shell (*sh*(1)), then the *append* device should be used. If the tape is being opened by the utility, then the *truncate* device should normally be used. When writing new utilities to use the MVME350, explicit uses of **O_TRUNC** and **O_APPEND** in the *open*(2) should be made. The normal tape utilities need to be used in special ways:

dd(1): The *dd*(1) program will always open the output file specified by the **of=** option, with the **O_TRUNC** flag. If no such option is specified, *dd*(1) will use its standard output. Therefore, to append to a tape using *dd*(1), redirect standard output using the shell append redirection (**>**) with the *append* device; do not use the **of=** option. For example, to dump **/dev/rdsk/0s0** onto the tape followed by **/dev/rdsk/1s0**, execute:

$ dd if=/dev/rdsk/0s0 of=/dev/rmt/m350_0t
$ dd if=/dev/rdsk/1s0 >/dev/rmt/m350_0a

cpio(1):

Since *cpio*(1) uses its standard output to write tapes, simply use the shell's redirection with the *append* device. For example, to rewrite a tape using *cpio*(1):

$ cpio -oBv <*filelist* **>/dev/rmt/m350_0a**

To append to a tape using *cpio*(1):

$ cpio -oBv <*filelist* **>/dev/rmt/m350_0a**

tar(1): To specify the output tape, *tar*(1) requires the **-f** option on the command line. If the filename supplied via the **-f** option is **-**, then *tar*(1) will use its standard output. In this mode, *tar*(1) may be used in the same manner as *cpio*(1). For example, to rewrite

a tape using *tar*(1):

$ tar -cf /dev/rmt/m350_0t *files* ...

or

$ tar -cf - *files* ... >/dev/rmt/m350_0t

To append to a tape using *tar*(1):

$ tar -cf /dev/rmt/m350_0a *files* ...

or

$ tar -cf - *files* ... >/dev/rmt/m350_0a

finc(1M):

To write a tape using *finc*(1M), it is first necessary to label the tape by using *labelit*(1M). For this purpose, it is better to use the *truncate* device. To label a tape and then write the tape using *finc*(1M), execute:

$ labelit /dev/rmt/m350_0t *fsname volname* **-n**
$ finc -m -10 /dev/rdsk/0s0 /dev/rmt/m350_0t

The above example copies all files on the device **/dev/rdsk/0s0** that have been modified in the last 10 days onto the streamer tape.

frec(1M):

Since *frec*(1M) does not write tapes, it does not matter whether the *truncate* device or *append* device is used.

volcopy(1M):

Like *finc*(1M), *volcopy*(1M) requires the tape to have a *labelit(1M)* label at the beginning. For this purpose the *truncate* device is the best to use. To label a tape and then use *volcopy*(1M) to write it, execute:

$ labelit /dev/rmt/m350_0t *fsname volname* **-n**
$ volcopy *fsname* **/dev/rdsk/0s0** *volname* **/dev/rmt/m350_0t** *volname*

The above example copies the filesystem on device **/dev/rdsk/0s0** onto streamer tape.

When reading tapes, either the *append* or *truncate* devices may be used. To begin reading from the second or subsequent file on a tape, the tape must be spaced to that file using the "no-rewind-on-close" device. For example, to space the tape to the second file on a tape, execute:

$ dd if=/dev/rmt/m350_0tn of=/dev/null

When the *dd*(1) completes, the tape will be left at the beginning of the second file on the tape. If another read is attempted, it will read from the second file. If there is no second file, the MVME350 will generate an error.

M350ctl(1M) Usage.

The *m350ctl*(1M) utility is used to gain quick access to the MVME350 devices. See the *m350ctl*(1M) manual page for more details. The program supports the following functions:

rewind, retension, erase, tape positioning, and DMA buffer size get and set.

Error Messages

The MVME350 generates many different error messages. These error messages, printed in English, attempt to provide enough information to permit the operator to diagnose tape problems. Most error messages start with a line that prints out the controller and drive number that has the error. The first line of the error message looks like:

MVME350: Error on controller 0, drive 0

The second and subsequent lines of the error message describe the symptoms encountered:

Filemark detected.
The last operation encountered a filemark. When encountered, a filemark will normally terminate reading and return without an error status.

Unrecoverable data error.
Some form of unrecoverable error has occurred. The operation should be retried. If the operation continues to get this error, then the tape is probably damaged.

End of Media.
The tape has encountered the *end-of-media* indicator. Further reading or writing of this tape is not allowed without rewinding.

Write Protected.
The tape's write-protect switch is set to **SAFE**. Normally, the *open*(2) will fail when attempting to open a write-protected tape for write.

Drive not online.
The MVME350 does not detect an "on-line" status from the streamer tape drive. Check that all cables are properly attached. If so, then retry the operation. If the problem persists, then the streamer tape drive is probably damaged.

Cartridge not in place.
No streamer tape cartridge has been loaded into the selected tape drive. Check to be sure the proper special file has been used to access the tape. If this problem persists, then the streamer tape drive is probably damaged.

Beginning of Media.
The *beginning-of-media* has been encountered. The tape is now rewound correctly.

No data detected.
The MVME350 has not detected any data on the tape during a read operation. A read has probably been attempted past the *end-of-media*.

No file mark encountered
An attempt to find a filemark has failed. The desired filemark appears not to be on the tape.

Not a beginning of tape.
The tape is not at the *beginning-of-tape* as expected.

Tape reset did not occur.
After every reported error, the MVME350 attempts to reset the tape drive. If the reset fails, this error message will be printed.

Timeout.
Some internal timeout has occurred, aborting the operation. Try the operation again. If the problem persists, then try a new tape or tape drive.

Bad Unit.
A tape drive unit failed to respond. Try the operation again. If the error persists, then the cables or tape drive are probably damaged.

Bad Drive.
A tape drive unit failed to respond. Try the operation again. If the error persists, then the cables or tape drive is probably damaged.

If an error message has only the first line, then retry the operation. If the error persists, then the MVME350 controller, the MVME350 driver software, or the streaming tape drive may be damaged or confused. In persistent errors, resetting the machine or cycling power will sometimes clear the problem.

The MVME350 driver also produces some warning messages. These warning messages are not fatal errors but are important to the operator. The warning messages are:

DMA buffers still active.
The last write onto the tape (using double buffering) encountered the *end-of-media*. The buffers are being retained for subsequent write operations (See the section on Double Buffering above).

DMA buffers discarded.
The new *open*(2) request is by a different program or is not requesting to write. The retained buffers are discarded (See the section on Double Buffering above).

Initialization error.
The MVME350 initialization sequence did not succeed. The hexadecimal value following the error message should be reported to the system administrator.

FILES

/dev/rmt/m350_* /usr/include/sys/mvme350.h

SEE ALSO

m350ctl(1M), cpio(1), dd(1), tar(1), finc(1M), frec(1M), volcopy(1M), open(2), close(2), ioctl(2).

NAME

null — the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf — operating system profiler

DESCRIPTION

The file *prf* provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file *prf* is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

config.68(1M), profiler(1M).

NAME

sa400fl22 — 5¼-inch Floppy Disk Drive for VM22 Driver

DESCRIPTION

Although it is possible to have eight devices for each floppy disk drive, only five devices per floppy disk drive are currently defined: **dsk/vm22_xs0**, **dsk/vm22_xs1**, **dsk/vm22_xs2**, **dsk/vm22_xs3**, and **dsk/vm22_xs7**.

Dsk/vm22_xs1 and **dsk/vm22_xs3** are defined for single-sided drives or single-sided disks in double-sided drives. **Dsk/vm22_xs0** and **dsk/vm22_xs2** are defined for double-sided disks in double-sided drives. **Dsk/vm22_xs7** is defined to cover the entire contents of the largest supported disk. These slices are created to be compatible with the slicing definition needed by the hard disk when using the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*	*description*
0	1	632	double-sided, single density
1	1	316	single-sided, single density
2	1	1264	double-sided, double density
3	1	632	single-sided, double density
4	-	-	
5	-	-	
6	-	-	
7	0	1276	largest floppy size

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark25(7), lark8(7), sa800fl22, vm22(7).

NAME

sa400flwd – 5 ¼inch Floppy Disk Drive for the Winchester Disk Driver and the general disk driver for the MVME319 and MVME320 Disk Controllers

DESCRIPTION

Although it is possible to have eight devices for each floppy disk drive, only two devices per floppy disk drive are currently defined: **dsk/**[*cntrlr*_]**xs0** and **dsk/**[*cntrlr*_]**xs7**.

Dsk/[*cntrlr*_]**xs0** and **dsk/**[*cntrlr*_]**xs7** are defined for double-sided disks in double-sided drives. These slices are created to be compatible with a double-sided, double density floppy created on a VME/10. A floppy created in **wd_0s0** on a RWIN1 (VME/10) can be read directly in [*cntrlr*_]**xs0** on a M320 or ID.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*	*description*
0	12	1264	double-sided double density
1	-	-	
2	-	-	
3	-	-	
4	-	-	
5	-	-	
6	-	-	
7	0	1276	Largest floppy size

The start address is a block address. The length is expressed in blocks, with each block containing 512 bytes. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read and write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

id(7), m320(7), wd15(7), wd40(7).

NAME

sa800fl21 — 8-inch Floppy Disk Drive for VM21 Driver

DESCRIPTION

Although it is possible to have eight devices for each floppy disk drive, only four devices per floppy disk drive are currently defined: **dsk/***vm21_x***s0**, **dsk/***vm21_x***s1**, **dsk/***vm21_x***s6**, and **dsk/***vm21_x***s7**.

Dsk/*vm21_x***s0** and **dsk/***vm21_x***s6** are defined for single-sided drives or single-sided disks in double-sided drives. **Dsk/***vm21_x***s1** and **dsk/***vm21_x***s7** are defined for double-sided disks in double-sided drives. These slices are created to be compatible with the slicing definition needed by the hard disk when using the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*	*description*
0	1	988	double-sided, single density
1	2	487	single-sided, single density
2	-	-	
3	-	-	
4	-	-	
5	-	-	
6	0	500	
7	0	1000	

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark25(7), lark8(7), vm21(7).

NAME

sa800fl22 — 8-inch Floppy Disk Drive for VM22 Driver

DESCRIPTION

Although it is possible to have eight devices for each floppy disk drive, only five devices per floppy disk drive are currently defined: **dsk/vm22_xs0**, **dsk/vm22_xs1**, **dsk/vm22_xs2**, **dsk/vm22_xs3**, and **dsk/vm22_xs7**.

Dsk/vm22_xs1 and **dsk/vm22_xs3** are defined for single-sided drives or single-sided disks in double-sided drives. **Dsk/vm22_xs0** and **dsk/vm22_xs2** are defined for double-sided disks in double-sided drives. **Dsk/vm22_xs7** is defined to cover the entire contents of the largest supported disk. These slices are created to be compatible with the slicing definition needed by the hard disk when using the software bad track replacement scheme.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*	*description*
0	1	988	double-sided, single density
1	2	487	single-sided, single density
2	1	1976	double-sided, double density
3	2	975	single-sided, double density
4	-	-	
5	-	-	
6	-	-	
7	0	2002	largest floppy size

The start address is a cylinder address. The length is expressed in blocks, with each block containing 512 bytes. Information about recommended file system partitioning is provided in the "Administrative Guidelines" section of the *SYSTEM V/68 Administrator's Guide.*

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O, the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). *Lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark25(7), lark8(7), sa400fl22(7), vm22(7).

NAME

sxt — pseudo-device driver

DESCRIPTION

Sxt is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the *sxt* driver. *Sxt* acts as a discipline manipulating a *real tty* structure declared as a real device driver. The *sxt* driver is currently only used by the *shl*(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl*(2) call to initiate the multiplexing.

Only one channel, the "controlling" channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: further opens are once again permitted.

The second group are directives to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL The argument is out of range.
- ENOTTY The command was not issued from a real tty.
- ENXIO *linesw* is not configured with *sxt*.
- EBUSY An SXTIOCLINK command has already been issued for this real *tty*.
- ENOMEM There is no system memory available for allocating the virtual tty structures.
- EBADF Channel 0 was not opened before this call.

SXTIOCSWTCH Set the controlling channel. Possible errors include:

- EINVAL An invalid channel number was given.
- EPERM The command was not executed from channel 0.

SXTIOCWF Cause a channel to wait until it is the controlling channel. This command will return the error, **EINVAL**, if an invalid channel number is given.

SXTIOCUBLK Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error **EINVAL** will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error **EFAULT** will be returned if the structure cannot be

written.

SXTIOCTRACE — Enable tracing. Tracing information is written to **/dev/osm** on the 3B 20 computer or to the console on the VAX. This command has no effect if tracing is not configured.

SXTIOCNOTRACE — Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7] Virtual tty devices
/usr/include/sys/sxt.h Driver specific definitions.

SEE ALSO

sh1(1), stty(1), ioctl(2), open(2), termio(7).

NAME

termio — general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. Common features of this interface are presented in this section.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty* and become a user's standard input, output, and error files. The first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, and are only lost when the system's character input buffers become completely full, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are discarded without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read is suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, one line at most is returned. It is not necessary, however, to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it does not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a newline character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case, the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it is not only terminated but a core image file (called **core**) is created in the current working directory.

SWTCH ASCII NUL is used by the job control facility, *shl*(1), to change the current layer to the control layer.

ERASE (#) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the

program, without waiting for a newline, and the EOF is discarded. Thus, if there are no characters waiting, i.e., the EOF occurred at the beginning of a line, zero characters are passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, similar to NL. Normally, it is not used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output that has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hangup* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished printing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they are printed, it is suspended when its output queue exceeds some limit. When the queue has drained to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define NCC     8
struct  termio {
        unsigned  short   c_iflag;    /* input modes */
        unsigned  short   c_oflag;    /* output modes */
        unsigned  short   c_cflag;    /* control modes */
        unsigned  short   c_lflag;    /* local modes */
        char              c_line;     /* line discipline */
        unsigned  char    c_cc[NCC];  /* control chars */
};
```

CONTROL CHARACTERS

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	VSWTCH	NUL

Refer to the section "LOCAL MODES" for information about enabling and disabling the functions of these characters. As stated in that section and shown in **termio.h**, if canonical processing is not set, positions 4 and 5 contain values for VMIN and VTIME, respectively.

INPUT MODES

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map uppercase to lowercase on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and, therefore, not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flushes both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7 bits; otherwise, all 8 bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character suspends output, and a received START character restarts output. All start/stop characters are ignored and not read. If IXANY is set, any input character restarts output that has been suspended.

If IXOFF is set, the system transmits START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all bits clear.

OUTPUT MODES

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lowercase to uppercase on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.

NLDLY	0000400	Select newline delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If OFILL is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise, it is NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

CONTROL MODES

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled, and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set; otherwise, even parity is used.

If CREAD is set, the receiver is enabled; otherwise, no characters are received.

If HUPCL is set, the line is disconnected when the last process with the line open closes it or terminates, i.e., the data-terminal-ready signal is not asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. If it is not set, modem control is assumed.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

LOCAL MODES

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus, these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least VMIN characters have been received or the timeout value VTIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The VMIN and VTIME values are stored in the positions for the EOF and EOL characters, respectively. The VTIME value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an uppercase letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters is not done.

The initial line-discipline control value is all bits clear.

I/O SYSTEM CALLS

The primary *ioctl*(2) system calls have the form:

ioctl (*fildes, command, arg*)
struct termio **arg;*

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
TCSETAW	Wait for the output to drain before setting new parameters. This form should be used when changing parameters that affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

ioctl (*fildes, command, arg*)
int *arg;*

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1), fork(2), ioctl(2), setpgrp(2), signal(2).

NAME

tty — controlling terminal interface

DESCRIPTION

The file **/dev/tty** is a synonym for the control terminal associated with the process group of each process. It is useful for programs or shell sequences that require written messages on the terminal, no matter how output has been redirected. It can also be used for programs that need the name of a file for output, when typed output is desired.

FILES

/dev/tty
/dev/tty*

NAME

ud — general driver for all disk units supported by the M68KVM21 disk controller

DESCRIPTION

Ud provides a general interface to M68KHDS32-1 (32Mb Cartridge Module Drive; refer to *cm16*(7)), M68KHDS50-1 (50Mb Lark Module Drive; refer to *lrk25*(7)), and the M68KHDS96-1 (Cartridge Module Drive; refer to *cm80*(7)). The driver can be modified to support other disk units compatible with the M68KVM21 controller.

Drive dependent partitioning must be selected when the system is configured (*config.68*(1M)). Also, manual entries describing the above disk drives should be referred to for information regarding those particular drives.

FILES

/dev/dsk, /dev/rdsk

SEE ALSO

config.68(1M), master(4), cm16(7), cm80(7), f18(7), lrk25(7).

NAME

v10graph - VME/10 graphics subsystem interface

DESCRIPTION

The VME/10 System Control Module (SCM) contains an MC68010, time-of-day clock with battery backup, a keyboard interface, an I/O Channel interface, a VMEbus interface, a color or monochrome video interface, and an operator panel interface. The 384Kb RAM in the SCM has the dual purpose of general software storage and graphics data storage. The RAM may be accessed by the MPU or by a VMEbus device; both the MPU and the VMEbus device use the SCM local bus to access RAM.

If the *graphics*(1G) program is not being used, all the RAM on the SCM is available as system RAM. In this mode, RAM appears at locations 0x00000 through 0x5ffff. If the *graphics* program is being used and the SCM is in low resolution graphics mode, system RAM appears at locations 0x00000 through 0x47fff, and the graphics RAM appears at locations 0x48000 through 0x5ffff (96Kb). If the *graphics* program is being used and the SCM is in the high resolution graphics mode, system RAM appears at locations 0x00000 through 0x2ffff, and the graphics RAM appears at locations 0x30000 through 0x5ffff (192Kb).

The graphics RAM is divided into three banks. Each bank is color for color monitors and an intensity for monochrome monitors.

The graphics RAM block in the memory map is organized so that the first third of the graphics RAM locations is bank 3, the second third is bank 2, and the last third is bank 1. This allows the MPU to change 8 or 16 pixels at a time in one bank (color/intensity) on the screen.

For example, if the monitor is a color monitor, then bank 3 is green, bank 2 is blue, and bank 1 is red. When the graphics RAM is zeroed, the screen is blank. If the MPU, using word writes, sequentially fills the graphics RAM from the lowest address to highest address with 0xffff's, the screen fills with green from top to bottom, 16 pixels at a time, until bank 3 is all f's. Next, the screen fills with cyan (green and blue mixed) from top to bottom, 16 pixels at a time, until bank 2 is all f's. Then, the screen fills with white (green, blue and red mixed) from top to bottom, 16 pixels at a time, until bank 1 is all f's. This is useful when drawing bar graphs, color filling an object, or changing a background color.

The graphics RAM is accessible in another mode, called the pixel access mode. In pixel access mode, read/write hardware enables the processor to change one pixel at a time in all three banks (in one memory cycle). The processor uses only word accesses, and writes a special pixel access word to addresses defined in non-existent memory. This mode is oriented toward drawing lines or changing a portion of the display. A 3-bit mask field allows the user to avoid disturbing the contents of a given plane (or planes) while changing the contents of another plane (or planes). It eliminates rewriting data into bank address when the data remains unchanged. The lower three bits of a pixel access word affect the three banks of the pixel, where bit 0 is bank 1, bit 1 is bank 2, and bit 2 is bank 3. Bits 3 through 7 and 11 through 15 have no function, and bits 8 through 10 are mask bits. Bit 8, when low, disallows any effect by bit 0 on bank 1 of the pixel accessed. Bit 9, when low, disallows any effect by bit 1 on bank 2 of the pixel accessed. Bit 10, when low, disallows any effect by bit 2 on bank 3 of the pixel accessed. Bits 8 through 10, when high, have no effect on their corresponding bank bit. For example, if the graphics RAM is zero, the screen is blank. If the MPU writes 0x0707, starting at the lowest address of the pixel access block of the memory map to the highest address, the screen fills with white, one pixel at a time from top to bottom. The pixel access block appears at locations 0xe00000 through 0xe7ffff in low resolution graphics mode, and at locations 0xe00000 through 0xefffff in high resolution graphics mode.

The VME/10 also includes 8Kb of static RAM for storage of user definable character sets and display attributes. The character generator RAM is initialized with the standard ASCII character definitions, but can be modified by the user to define alternative symbols required by an application. The font is 8 x 16 in a 10 x 24 character field for the high resolution,

monochrome display, and 8 x 10 in a 10 x 12 character field for low resolution, color dislay. Characters or individual pixels can be displayed on any one of seven levels of grey scale on the monochrome display. For more information on the VME/10 hardware, refer to *VME/10 Microcomputer System System Control Module User's Manual.*

The fastest access to the VME/10 graphic subsystem is through shared memory, but the special shared memory segments must be at fixed addresses. This requirement necessitates a special system configuration and initialization. When the system is booted with a graphics kernel, 196Kb of graphics RAM is automatically reserved. All or part of this memory may be returned after the graphics driver determines which shared memory segments are to be pre-allocated, as configured in the graphics **dfile.** For more information on building and installing a graphics **dfile** and graphics kernel, refer to the *SYSTEM V/68 Graphics Guide*, "Installing Graphics." If the **dfile** specifies that only the character and attribute shared segment is to be pre-allocated, 196Kb of RAM is returned to the coremap. If the **dfile** specifies that the banks' area and pixel access area are to be pre-allocated, the resolution of the VME/10 is checked. If the VME/10 is in low resolution mode, 96Kb of RAM is returned to the coremap. If high resolution is set, 196Kb of RAM remains reserved.

WARNING

The resolution mode must be set by a TENbug Video Map (VM) command BEFORE the system is booted. For more information on the TENbug, refer to TENbug Debugging Package User's Manual.

User access to the VME/10 graphics subsystem is provided through shared memory system calls and the *ioctl*(2) system calls. These three segments are owned by **root**; read and write permissions are available for group users and all others. A user accesses a particular segment by issuing a *shmget*(2) with a pre-defined key for the segment, and then issues a *shmat*(2) system call with the shared memory identifier returned by the *shmget* system call. When a user is finished with the the shared memory segment, a *shmdt*(2) system call must be issued.

The *shmget* system call has the form:

```
int   shmget(key, size, shmflg)
      key_t  key
      int    size, shmflg
```

The predefined keys for shared memory graphics are:

```
#define CGENKEY    -1   /*character and attribute shared memory key*/
#define PIXKEY     -2   /*pixel access area shared memory key*/
#define BANKKEY    -3   /*color banks shared memory key*/
```

For normal shared memory segments, the size and *shmflg* are normally supplied by the user process. However, for the special segments, these values are not used since the segments have been pre-allocated. A recommended value for both of these arguments is 0.

The *shmat*(2) system call attaches the shared memory segment associated with the shared memory identifier specified by *shmid*(2) to the data segment of the calling process. The segment is attached at the address specified by *shmaddr*(2).

The *shmat* system call has the form:

```
char*   shmat(shmid,shmaddr,shmflg)
        int    shmid
        char   *shmaddr
        int    shmflg
```

The *shmaddr* value supplied in the user's *shmat*(2) call will affect the speed of graphics output, due to the operation of the M68451 Memory Management Unit (MMU). The fewer MMU descriptors that are required to describe the segment, the less time spent in address translation by the MMU. Experimentation on the VME/10 shows that performance is enhanced if one of the constants listed below is selected for the input parameter, *shmaddr*, in the *shmat*(2) system call.

Suggested *Shmaddr* Value for Color/Intensity Banks Shared Segment
0x810000 (for high resolution only) 0x818000 (for low resolution only)

Suggested *Shmaddr* Value for Pixel Access Shared Memory Segment
0x880000 (for high resolution only) 0x840000 (for low resolution only)

Suggested *Shmaddr* Value for Character/Attribute Generator Shared Segment
0x801000 (for both high and low resolution)

The *shmdt*(2) system call detaches the shared memory segment from the calling process's data segment. The shared memory segment is located at an address specified by ***shmaddr.***

The *shmdt*(2) system call has the form:

```
int    shmdt(shmaddr)
       char    *shmaddr
```

The shared memory segment used to access the color banks may be removed using the *shmctl*(2) call which frees the associated graphics memory, making it available for general system use. Once removed, graphics capability may be regained only by re-booting the system. In addition, access to the pixel access shared memory segment after the memory is released will have unpredictable and potentially disastrous effects. To protect against this possibility, the pixel access shared memory segment can be made unreadable and unwritable before the color bank segment is removed.

The shared memory segments associated with the pixel access area and the character generator RAM should never be removed using the *shmctl*(2) call. Doing so will place the special hardware addresses into the available memory pool. The system may then try to load programs or data into these areas, causing unpredictable behavior.

The *shmctl* system call has the following form:

```
int    shmctl(shmid, cmd, buf)
       int    shmid, cmd
       struct shmid_ds*buf
```

To remove the shared memory identifier for a color/intensity bank segment, the cmd is IPC_RMID. Only the super user is permitted to free this segment. *Shmctl* can also be used to change the access permissions of the special segments.

Access to some of the VME/10 graphic hardware registers has been provided through the *ioctl*(2) system call on the special device, **/dev/v10graph**. An *open*(2) system call on this device returns a file descriptor. This file descriptor is necessary for the *ioctl* system calls for graphics.

Control register 0 and control register 1 affect the display of graphics. Control register 0 is cleared to 0 when any of the four VME/10 reset conditions occur. Control register 1 is cleared to 0 only when the power-on-reset condition occurs. Control registers 0 and 1 are writable and readable by the MPU when it is in the user state. However, the data read is not reliable unless the control register has been written to by the processor at least once since the last reset condition occurred.

The MC6845 (CRTC) registers and the graphics offset registers have fixed values for high and low resolution graphics display. The *ioctl* commands **SETHIRES** and **SETLORES** set these registers at their fixed values.

Two separate registers control horizontal and vertical cursor display. The vertical graphics cursor register is set to display a vertical line on the screen. The horizontal graphics cursor register is set to display a horizontal line on the screen.

The graphics *ioctl* calls use the following structure types, defined in **sys/v10gr.h**:

```
typedef struct      {
                    unsigned char       cdis:3;
                    unsigned char       curbk:1;
                    unsigned char       dutycycle:1;
                    unsigned            ivs:1;
                    unsigned char       :2
} v10_cr0;
```

The values of the bit fields are:

cdis:3 — Character disable: This 3-bit field disables the display of the red/green/blue portions of the character. Permissible values range from 0 to 7 where:

- 001 disables red portion of the character
- 010 disables blue portion of the character
- 100 disables green portion of the character

curbk:1 — Cursor blink: The character cursor will blink when set(1) and will be steady when reset(0).

dutycycle:1 — Display dutycycle: This bit selects the display dutycycle of 50% when set(1) and 100% when reset(0). When high, this bit will prevent every other dot on each line from being displayed. This prevents horizontal lines, such as those in the letter B, from standing out more than non-horizontal lines, such as those in the letter X.

ivs:1 — Invert video screen: When set(1), all characters on the screen are inverted. When low, all characters are normal.

:2 — An unnamed bit field. These bit fields are not used for graphics.

```
typedef struct      {
                    unsigned char       :1;
                    unsigned char       sel:2;
                    unsigned char       hires;
                    unsigned            gre:3;
                    unsigned char       :1
    } v10_cr1;
```

The values of the bit fields are:

:1	An unnamed bit field. This bit field is not used for graphics.
sel:2	Character cursor: This field selects the style of character cursor. It may be any value from 0 to 3.
hires:1	High resolution: This bit field is read-only. Memory has been mapped to high resoluton when set(1) and mapped to low resolution when reset(0).
gre:3	Graphics enable: This bit field enables the display of graphics in each of the three color banks. Permissible values range from 0 to 7 where: 001 enables graphics in color bank 1 (red) 010 enables graphics in color bank 2 (blue) 100 enables graphics in color bank 3 (green)
:1	An unnamed bit field. This bit field is not used for graphics.

```
typedef struct      {
                    short h;
                    short v;
    } v10_cursor;
```

h Horizontal line positioning: This 16-bit field will move the horizontal component of the cursor top to bottom for 300 positions.

0xff ff (-1) is the top-most position.
0xfed4 (-300) is the bottom-most position.
0 turns off the horizontal component of the cursor.

v Vertical line positioning: This 16-bit field will move the vertical component of the cursor left to right for 800 positions.

0xff ff (-1) is the left-most position.
0xfed4 (-800) is the right-most position.
0 turns off the vertical component of the cursor.

The VME/10 graphics kernel supports 6 ioctl commands for controlling the display hardware These commands and the type of argument they require are listed below:

```
#include <sys/v10gr.h>
ioctl(fildes, command, arg)
v10_cr0        *arg;
```

The commands using this form are:

SETCR0 Set the contents of control register 0.
GETCR0 Get the contents of control register 0.

```
#include <sys/v10gr.h>
ioctl(fildes, command, arg)
v10_cr1        *arg;
```

The commands using this form are:

SETCR1 Set the contents of control register 1.
GETCR1 Get the contents of control register 1.

```
#include <sys/v10gr.h>
ioctl(fildes, command, arg)
v10_cursor       *arg;
```

The commands using this form are:

SETCUR Set the contents of the cursor registers.
GETCUR Get the contents of the cursor registers.

```
#include <sys/v10gr.h>
ioctl(fildes, command, NULL)
```

The commands using this form are:

SETLORES Change the CRTC registers and graphics offset registers to standard low resolution values.
SETHIRES Change the CRTC registers and graphics offset registers to defaults for high resolution values.

It should be noted that this command changes the resolution of the display only and does not affect the mapping of memory. The default resolution of the display at system initialization is low resolution.

FILES
/dev/v10graph dfile

SEE ALSO
graph(1G), ioctl(2), shmget(2), shmat(2), shmdt(2), shmctl(2)

NAME

vm21 — default general driver for all disk units supported by the M68KVM21 disk controller

DESCRIPTION

Vm21 provides a general interface to M68KHDS32-1 (32Mb Cartridge Module Drive), M68KHDS50-1 (50Mb Lark Module Drive), M68KHDS16-1 (16Mb Lark Module Drive), and M68KHDS96-1 (Cartridge Module Drive). The driver can be modified to support other disk units compatible with the M68KVM21 controller.

Drive dependent partitioning must be selected when the system is configured (*config*(1M)). Manual entries describing the above disk drives should be referred to for information regarding those particular drives. The partitioning and driver provide a mechanism for software bad track replacement. This set supercedes the *ud*(7) driver and disk drive partitioning.

The *vm21* accepts *ioctl*(2) calls of the form:

ioctl (*fildes, command, arg*)
int *arg*;

The following command is available:

UDFMT: Formats the track starting at the block number specified in *arg*.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark25(7), lark8(7), sa800f121(7)

NAME

vm22 — default general driver for all disk units supported by the M68KVM22 disk controller

DESCRIPTION

Vm22 provides a general interface to M68KHDS32-1 (32Mb Cartridge Module Drive), M68KHDS50-1 (50Mb Lark Module Drive), M68KHDS16-1 (16Mb Lark Module Drive), and M68KHDS96-1 (Cartridge Module Drive). The driver can be modified to support other disk units compatible with the M68KVM22 controller.

Drive dependent partitioning must be selected when the system is configured (*config*(1M)). Manual entries describing the above disk drives should be referred to for information regarding those particular drives. The partitioning and driver provide a mechanism for software bad track replacement.

Primary *ioctl*(2) system calls accepted by *vm22* are of the form:

ioctl(*fildes,command,arg*)
struct vm22config **arg*

As defined in the include file **<vm22.h>**, commands using this form are:

VM22GET: Gets the configuration information associated with the drive and places it into the location specified by *arg*.

VM22SET: Sets the drive's configuration to that specified by the *arg* parameter. This command will execute a vm22 controller "configure" command.

Additional *ioctl*(2) calls have the form:

ioctl (*fildes, command, arg*)
int *arg*;

As defined in **<vm22.h>**, commands using this form are:

VM22FMTT: Formats the track starting at the block number specified in *arg*.

VM22FMTU: Formats the entire logical unit.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cmd16(7), cmd80(7), lark25(7), lark8(7), sa800f122(7), sa400f122(7), ioctl(2).

NAME

wd15 – 15Mb Winchester Disk Drive

DESCRIPTION

The files **dsk**/*cntrlr*_**0s0** ... **dsk**/*cntrlr*_**0s7** refer to sections of the **wd15** disk drive 0. The files **dsk**/*cntrlr*_**1s0** ... **dsk**/*cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the device to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	192	24504
1	7056	21168
2	10584	17640
3	14112	14112
4	17640	10584
5	21168	7056
6	24696	3528
7	0	28224

Slice 0 follows a 192-block reserved area used for tables and tracks for disk diagnostics.

The start address is a block address, with each block containing 512 bytes. It is extremely unwise for all of these files to be present in one installation because there is overlap in addresses, and protection becomes a problem.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number which selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

cm16(7), cm80(7), fl8(7), id(7), lrk25(7), m320(7), ud(7), wd40(7).

NAME

wd40 – 40Mb Winchester Disk Drive

DESCRIPTION

The files **dsk/***cntrlr*_**0s0** ... **dsk/***cntrlr*_**0s7** refer to sections of the **wd40** disk drive 0. The files **dsk/***cntrlr* **_1s0** ... **dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the device to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	192	77328
1	18792	58728
2	24192	53328
3	28224	49296
4	38640	38880
5	46824	30696
6	63072	14448
7	0	77520

Slice 0 follows a 192-block reserved area used for tables and tracks for disk diagnostics.

The start address is a block address, with each block containing 512 bytes. It is extremely unwise for all of these files to be present in one installation because there is overlap in addresses and protection becomes a problem.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

id(7), wd15(7), m320(7).

NAME

wd70 – 70Mb Winchester Disk Drive

DESCRIPTION

The files **dsk/***cntrlr*_**0s0** ... **dsk/***cntrlr*_**0s7** refer to sections of the **wd70** disk drive 0. The files **dsk/***cntrlr* **_1s0** ... **dsk/***cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the device to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	1280	128640
1	28160	101760
2	45440	84480
3	60800	69120
4	76160	53760
5	92160	37760
6	111360	18560
7	0	130944

Slice 0 follows a 1280-block reserved area used for drive-specific tables and tracks for disk diagnostics.

The start address is a block address, with each block containing 512 bytes. It is extremely unwise for all of these files to be present in one installation because there is overlap in addresses and protection becomes a problem.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

sa400flwd(7), wd15(7), wd40(7).

? 00flwd(7), wd15(7), wd40(7).

NAME

wd140 – 140Mb Winchester Disk Drive

DESCRIPTION

The files **dsk**/*cntrlr*_**0s0** ... **dsk**/*cntrlr*_**0s7** refer to sections of the **wd140** disk drive 0. The files **dsk**/*cntrlr*_**1s0** ... **dsk**/*cntrlr*_**1s7** refer to sections of drive 1. This slicing allows the device to be broken up into more manageable pieces.

The origin and size of the sections on each drive are as follows:

section	*start*	*length*
0	1680	217200
1	32400	186480
2	64800	154080
3	93600	125280
4	122400	96480
5	151200	67680
6	180000	38880
7	0	220320

Slice 0 follows a 1680-block reserved area used for drive-specific tables and tracks for disk diagnostics.

The start address is a block address, with each block containing 512 bytes. It is extremely unwise for all of these files to be present in one installation because there is overlap in addresses and protection becomes a problem.

The **dsk/*** files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a "raw" interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files begin with **rdsk** and end with a number that selects the same disk section as the corresponding **dsk** file.

In raw I/O the buffer must begin on a word boundary, and counts must be a multiple of 512 bytes (a disk block). Likewise, *lseek*(2) calls should specify a multiple of 512 bytes.

FILES

/dev/dsk/*, /dev/rdsk/*

SEE ALSO

sa400flwd(7), wd15(7), wd40(7), wd70(7).

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section outlines procedures for those charged with the task of system maintenance. Included are discussions on boot procedures, recovery from crashes, and file backups.

NAME

bo.macs — bootstrap operating procedure for system restart on EXORmacs

SYNOPSIS

bo [<device>] [,<controller>] [,<string>]

Options

device	a single hexadecimal digit (0-F) specifying the device to be used (default = 0).
controller	a single hexadecimal digit (0-F) specifying the controller to which the device is connected (default = 0).
string	an optional ASCII character string (maximum of 18 characters) that is passed to the program being loaded from the specified device and controller. This string may be the pathname of the SYSTEM V/68 program to be booted (default = **/stand/unix**).

DESCRIPTION

When the system is turned on, the front panel status should be **01**. Perform the system self-test by holding the system reset and the system test buttons depressed. Release first the system reset button and then release the system test button. The status changes to **EA** while memory is initialized, then to **01** when the test is complete. The prompt **P*** appears after the Return key is pressed. After receiving the prompt, type: **bo** (drive 0 is the default and accesses the fixed media). If the system resides on the removable media, type: **bo 1** (drive 1 is accessed). The CRT displays:

INIT: SINGLE USER MODE
#

Enter: **init 2**

The CRT displays:

INIT: New run level: 2
Is the date <day> <month> <date> <time> <year> correct? (y or n)

If the date is incorrect, type: **n** and set the date and time. For the correct date and time, the following format is required: *mmddhhmm[yy]*, where mm=month, dd=day, hh=hour, mm=minute, yy=year (refer to *date*(1)). For example:

Sept. 28, 1983 at 7:30am is **0928073083**

If the date is correct, type: **y** ; the CRT displays:

Do you want to check the file system? (y or n)

To prevent possible system damage, a file system check is recommended. Enter: **y** . The following is an example of a file system check display:

/dev/dsk/vm21_1s0
File System: /d0 Volume: fixed
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Free List <n> files <n> blocks <n> free**

*****SYSTEM MULTI-USER <day> <month> <date> <time> <year>*****

If a file system check is not required, enter: **n** ; the "SYSTEM MULTI-USER" display is printed.

A login prompt now appears on activated CRT terminals.

FILES

/stand/unix

SEE ALSO

date(1), fsck(1M), init(1M), ops.macs(8).

WARNINGS

Memory initialization must be completed before this boot procedure is used.

BO.VME(8)

NAME

bo.vme — bootstrap operating procedure for system restart on VME/10

SYNOPSIS

bo [<device>] [,<controller>] [,<string>]

Options

device	a single hexadecimal digit (0-F) specifying the device to be used (default = 0).
controller	a single hexadecimal digit (0-F) specifying the controller to which the device is connected (default = 0).
string	an optional ASCII character string (maximum of 18 characters) that is passed to the program being loaded from the specified device and controller. This string may be the pathname of the SYSTEM V/68 program to be booted (default = **/stand/unix**).

DESCRIPTION

When the system is turned on and the KYBD LOCK switch is in the horizontal position, the system self-test is automatically performed. The first message to appear will be:

Power-up test in progress.
Waiting for disk to spin up.

After the message

Power-up test complete.

has been received, the TENbug prompt will appear. This prompt should be:

TENbug 2.0 >

If the serial port (MVME400) is connected, however, the prompt that will appear on the screen is:

TENbug

Type a carriage return to obtain the **TENbug 2.0 >** prompt. If the **TENbug 2.0 >** prompt does not appear, unplug the serial port, press the RESET button, and boot the system before plugging in the serial port again.

After receiving the prompt, type: **bo** (drive 0 is the default and accesses the fixed media). If the system resides on the removable media, type: **bo 1** (drive 1 is accessed). The CRT displays:

INIT: SINGLE USER MODE
#

Enter: **init 2**

The CRT displays:

INIT: New run level: 2
Is the date <day> <month> <date> <time> <year> correct? (y or n)

If the date is incorrect, type: **n** and set the date and time. For the correct date and time, the following format is required: *mmddhhmm[yy]*, where mm=month, dd=day, hh=hour, mm=minute, yy=year (refer to *date*(1)). For example:

Sept. 28, 1983 at 7:30am is **0928073083**

If the date is correct, type: **y** ; the CRT displays:

Do you want to check the file system? (y or n)

To prevent possible system damage, a file system check is recommended. Enter: **y** . The following is an example of a file system check display:

/dev/dsk/wd_1s0
File System: /d0 Volume: fixed
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Free List <n> files <n> blocks <n> free**

*****SYSTEM MULTI-USER <day> <month> <date> <time> <year>*****

If a file system check is not required, enter: **n** ; the "SYSTEM MULTI-USER" display is printed.

A login prompt now appears on activated CRT terminals.

FILES

/stand/unix

SEE ALSO

date(1), fsck(1M), init(1M).

WARNINGS

Memory initialization must be completed before this boot procedure is used.

NAME

crash — what to do when the system crashes

DESCRIPTION

This entry gives at least a few clues about how to proceed if the system crashes. It can't pretend to be complete.

In restarting after a crash, always bring up the system single-user, as specified in *bo.macs*(8) for the EXORmacs Development System or *bo.vme*(8) for the VME/10, as modified for your particular installation. Then perform an *fsck*(1M) on all file systems that could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user.

To even boot the SYSTEM V/68 at all, three files (and the directories leading to them) must be intact. First, the initialization program **/etc/init** must be present and executable. For *init* to work correctly, **/dev/console** and **/bin/sh** must be present. If either does not exist, the symptom is best described as thrashing. *Init* goes into a *fork/exec* loop trying to create a shell with proper standard input and output.

If you cannot get the system to boot, a runnable system must be obtained from a backup medium. The root file system may then be doctored as a mounted file system as described below. If there are any problems with the root file system, it is probably prudent to go to a backup system to avoid working on a mounted file system.

Repairing disks. The first rule to keep in mind is that a disk in need of repair should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it.

Fsck(1M) is adept at diagnosing and repairing file system problems. It first identifies all of the files that contain bad (out of range) blocks or blocks that appear in more than one file. Any such files are then identified by name and *fsck* requests permission to remove them from the file system. Files with bad blocks should be removed. In the case of duplicate blocks, all of the files except the most recently modified should be removed. The contents of the survivor should be checked after the file system is repaired to ensure that it contains the proper data. (Note that running *fsck* with the **—n** option causes it to report all problems without attempting any repair.)

Fsck also reports on incorrect link counts and requests permission to adjust any that are erroneous. In addition, it reconnects any files or directories that are allocated but have no file system references to a "lost+found" directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) *fsck* constructs a new one, with the operator's concurrence.

Why did it crash? SYSTEM V/68 types a message on the console when a crash occurs. Here is the current list of such messages, with enough information to provide a possible remedy. The message has the form "panic: . . .", usually accompanied by other information. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

blkdev

The *getblk* routine was called with a nonexistent major device as argument. Definitely hardware or software error.

devtab

Null device table entry for the major device used as argument to *getblk*. Definitely hardware or software error.

iinit

An I/O error reading the superblock for the root file system during initialization.

no fs
A device has disappeared from the mounted-device table. Definitely hardware or software error.

no imt
Similar to "no fs", but produced elsewhere.

no clock
During initialization, neither the line nor programmable clock was found to exist.

I/O error in swap
An unrecoverable I/O error during a swap. This shouldn't be a panic, but it is hard to fix.

out of swap space
A program needs to be swapped out, and there is no more swap space. It has to be increased. This shouldn't be a panic, but there is no easy fix.

trap
An unexpected trap has occurred within the system. This is accompanied by three numbers: a "ps", which is the user's stack pointer; "pc", which is the user's program counter; and a "trap type" that encodes which trap occurred. The trap types are:

2	bus error
3	address error
4	illegal instruction
5	zero divide fault
6	CHK instruction fault
7	TRAPV instruction fault
8	privileged instruction fault
9	trace trap
10	line 1010 emulator
11	line 1111 emulator
24	spurious interrupt
32	TRAP 0 - system call
33	TRAP 1 - breakpoint
34	TRAP 2 - simulate DEC IOT instruction
35	TRAP 3 - simulate DEC EMT instruction
36	TRAP 4 - floating point exception

In some of these cases it is possible for hexadecimal 200 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, dump the system.

Interpreting dumps. (NOTE: This section does not apply for the VME/10.) All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file **/usr/tmp/core**; *cp*(1) can be used. At this point, you should execute **ps —el —c /usr/tmp/core** and **who** to print the process table and a list of the users who were on at the time of the crash.

SEE ALSO
crash(1M), fsck(1M), bo.macs(8), ops.macs(8), bo.vme(8).

NAME

mk — how to remake the system and commands

DESCRIPTION

All source for SYSTEM V/68 is in a source tree distributed in the directory **/usr/src**. This includes source for the operating system, libraries, commands, miscellaneous files necessary to the running system, and procedures to create everything from this source.

The top level consists of the directories **cmd**, **lib**, **uts**, **head**, and **stand** as well as commands to remake each of these directories. These commands are named *:mk*, which remakes everything, and *:mk* **dir** where **dir** is the directory to be recreated. Each recreation command makes all or part of the piece; over which it has control. *:mk* runs each of these commands and thus recreates the whole system.

The **lib** directory contains libraries used when loading user programs. The largest and most important of these is the C library. All libraries are in sub-directories and are created by a makefile or runcom. A runcom is a shell command procedure used specifically to remake a piece of the system. *:mklib* rebuilds the libraries that are given as arguments. The argument * causes it to remake all libraries.

The **head** directory contains the header files, usually found in **/usr/include** on the running system. *:mkhead* installs those header files that are given as arguments. The argument * causes it to install all header files.

The **uts** directory contains the source for the operating system. *:mkuts* (no arguments) invokes a series of makefiles that recreate the operating system.

The **stand** directory contains stand-alone commands and boot programs. *:mkstand* rebuilds and installs these programs.

The **cmd** directory contains files and directories. *:mkcmd* transforms source into a command based upon its suffix (**.l**, **.y**, **.c**, **.s**, **.sh**), or its makefile (see *make*(1)) or runcom. A directory is assumed to have a makefile or a runcom that takes care of creating everything associated with that directory and its sub-directories. Makefiles and runcoms are named *command* **.mk** and *command* **.rc** respectively.

:mkcmd recreates commands based upon a makefile or runcom if one of them exists; alternatively commands are recreated in a standard way based on the suffix of the source file. All commands requiring more than one file of source are grouped in sub-directories, and must have a makefile or a runcom. C programs (**.c**) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (**.s**) are assembled with **/usr/include/sys.s** which contains the system call definitions. Yacc programs (**.y**) and lex programs (**.l**) are processed by *yacc*(1) and *lex*(1) respectively before C compilation. Shell programs (**.sh**) are copied to create the command. Each of these operations leaves a command in **./cmd** which is then installed by using **/etc/install**.

The arguments to *:mkcmd* are either command names, or subsystem names. The subsystems distributed with SYSTEM V/68 are: **acct**, **graf**, **sccs**, and **text**. Prefacing the *:mkcmd* instruction with an assignment to the shell variable **$ARGS** causes the indicated components of the subsystem to be rebuilt.

The entire **sccs** subsystem can be rebuilt by:

/usr/src/:mkcmd sccs

while the *delta* component of **sccs** can be rebuilt by:

ARGS="delta" /usr/src/:mkcmd sccs

The **log** command, which is a part of the **stat** package, which is itself a part of the **graf** package, can be rebuilt by:

ARGS="stat log" /usr/src/:mkcmd graf

The argument * causes all commands and subsystems to be rebuilt.

Makefiles, both in **./cmd** and in sub-directories, have a standard format. In particular *:mkcmd* depends on there being entries for *install* and *clobber*. *Install* should cause everything over which the makefile has jurisdiction to be made and installed by **/etc/install**. *Clobber* should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

Most of the runcoms in **./cmd** (as opposed to sub-directories) relate in particular to a need for separated instruction and data (I and D) space.

Ctime checks the environment (see *environ*(5)) for the time zone. This results in time zone conversions possible on a per-process basis. **/etc/profile** sets the initial environment for each user, and **/etc/rc** sets it for certain system daemons. These two programs are the only ones which must be modified outside of the eastern time zone.

An effort has been made to separate the creation of a command from source, and its installation on the running system. The command **/etc/install** is used by *:mkcmd* and most makefiles to install commands in the proper place on the running system. The use of install allows maximum flexibility in the administration of the system. Install makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in install. The object is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles. The minimization of makefiles in most cases, and the site independence of the others should greatly reduce the necessary maintenance, and allow makefiles to be considered part of the standard source.

SEE ALSO

install(1M), make(1).

NAME

ops.macs — EXORmacs operations

DESCRIPTION

The procedures described include the major operational sequences involved in running SYSTEM V/68 on the EXORmacs.

INSTALLATION BOOT PROCEDURES

Refer to "Setting Up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide.*

DAILY PROCEDURES

DISK BOOT

For system restart, refer to: *bo.macs*(8).

BRINGING THE SYSTEM DOWN

The shutdown procedure is designed to turn off all processes and bring the system back to single user state with all buffers flushed. To do this the operator should execute *shutdown*(1M). If *shutdown* is not successful, use the following sequence of commands:

killall
sync
telinit S
fsck (optional)
sync
sync

The system may then be halted by pressing the RESET button on the chassis.

SYSTEM DUMPS

After a crash, the following procedure should be used to get a system dump:

1. Press the SOFTWARE ABORT button on the EXORmacs (pressing the SYSTEM RESET button also works, but it destroys all of the system interrupt vectors). The prompt **P*** appears.

2. Enter:

g 400

This starts the dump. After a short period of time, the system responds with:

Dump complete. dd skip=*xxx***, dd count=***yyy*

where *xxx* and *yyy* are decimal numbers (of blocks) to be used later.

If the system responds instead with:

I/O error during dump

then some type of I/O error has occurred. Try pressing the SYSTEM RESET button and re-enter **g 400 .** If the error message appears a second time, consult local lab support personnel.

3. Press the SYSTEM RESET button and boot the system (see *bo.macs*(8)). DO NOT enter **init 2** when the system comes up; remain in Single User mode.

Should the system not come up, refer to *crash.macs*(8) for additional information.

4. If the root file system does not have sufficient room for the core dump (at least *yyy* blocks free), then a file system with enough room has to be mounted. Refer to *mount*(1M).

5. If the number *yyy* from Step 2 is larger than 2048, then the maximum writable file size has to be increased in order to save the system dump. To increase file size, enter: **ulimit** *n* where *n* = whatever size is sufficient. For example:

ulimit 32768

The size should be at least *yyy* .

6. Since the dump was written to the system swap area, it must be saved in a file for later analysis. To save the dump in a file, enter:

dd if=/dev/swap of=*filename* **skip=***xxx* **count=***yyy*

where *name* is the name of the file that receives the dump; *xxx* and *yyy* are the numbers from Step 2. If *xxx* is 0, then the *skip* parameter does not have to be included on the **dd** command line.

7. If a file system was mounted in Step 4, unmount it now. Refer to *umount* in *mount*(1M).

8. Check the file system by running *fsck*(1M).

9. Boot the system normally (see *bo.macs*(8)), assuming *fsck* completed normally.

10. Once the system is back up, the following command starts *crash*(1M) so that the dump can be analyzed:

/etc/crash /fixed/*filename*

SYSTEM FAULTS

Refer to *MACSbug Monitor Reference Manual* (M68KMACSBG).

FILES

/etc/shutdown
/stand/*

SEE ALSO

date(1), dd(1), fsck(1M), init(1M), shutdown(1M), sync(1), bo.macs(8), *EXORmacs Chassis User's Guide*(M68KCHAS), *MACSbug Monitor Reference Manual*(M68KMACSBG), "Setting Up SYSTEM V/68" in the *SYSTEM V/68 Administrator's Guide*(M68KUNAG).

USER'S COMMENTS

SYSTEM V/68 ADMINISTRATOR'S MANUAL

Product Code 72900
Part Number 41963-00

Motorola welcomes your comments and suggestions. Please use this form.

- Does this manual provide the information you need? ☐ Yes ☐ No
  - — What is missing?

- Is the manual accurate? ☐ Yes ☐ No
  - — What is incorrect? (Be specific.)

- Is the manual written clearly? ☐ Yes ☐ No
  - — What is unclear? (Be specific.)

- What other comments can you make about this manual?

- What do you like about this manual?

- Was this manual difficult to obtain? ☐ Yes ☐ No

Please include your name and address if you would like a reply.

Name_____
Company_____
Address_____
_____

• What is your occupation?

☐ Programmer	☐ Operator	☐ Manager
☐ Systems Analyst	☐ Instructor	☐ Customer Engineer
☐ Engineer	☐ Student	☐ Other_____

• How do you use this manual?

☐ Reference Manual	☐ Introduction to the Subject
☐ In a Class	☐ Introduction to the System
☐ Self Study	☐ Other_____

fold **fold**

MOTOROLA INC.
3013 S. 52nd Street
Tempe, AZ 85282

Attention: Software Publications, X4

---

fold **fold**

Staple Here

MOTOROLA
Computer Systems

3013 S. 52nd St.
Tempe, AZ 85282