SCO® UNIX®
Operating System

System Administrator's Reference



SCO® UNIX®
Operating System

System Administrator's Reference



© 1983-1992 The Santa Cruz Operation, Inc.
© 1980-1992 Microsoft Corporation.
© 1989-1992 UNIX System Laboratories, Inc.
All Rights Reserved.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, The Santa Cruz Operation, Inc., 400 Encinal, Santa Cruz, California, 95061, U.S.A. Copyright infringement is a serious matter under the United States and foreign Copyright Laws.

The copyrighted software that accompanies this manual is licensed to the End User only for use in strict accordance with the End User License Agreement, which should be read carefully before commencing use of the software. Information in this document is subject to change without notice and does not represent a commitment on the part of The Santa Cruz Operation, Inc.

The following legend applies to all contracts and subcontracts governed by the Rights in Technical Data and Computer Software Clause of the United States Department of Defense Federal Acquisition Regulations Supplement:

RESTRICTED RIGHTS LEGEND: USE, DUPLICATION, OR DISCLOSURE BY THE UNITED STATES GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN SUBPARAGRAPH (c) (1) (ii) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE AT DFARS 52.227-7013. "CONTRACTOR/SUPPLIER" IS THE SANTA CRUZ OPERATION, INC. 400 ENCINAL STREET, SANTA CRUZ, CALIFORNIA 95061, U.S.A.

Microsoft, MS-DOS, and XENIX are trademarks of Microsoft Corporation.
UNIX is a trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.
"ACER Fast File System" is a trademark of ACER Technologies Corporation.

Date: 31 January 1992
Document Version: 3.2.4C

System Administration (ADM)

intro(ADM) 1
accept(ADM) 2
acct(ADM) 3
acctcms(ADM) 5
acctcom(ADM) 7
acctcon(ADM) 10
acctmerg(ADM) 12
accton(ADM) 13
acctprc(ADM) 14
acctsh(ADM) 16
addxusers(ADM) 19
aioinfo(ADM) 23
aiolkinit(ADM) 24
ale(ADM) 25
ap(ADM) 27
asktime(ADM) 29
asroot(ADM) 31
atcronsh(ADM) 33
auditcmd(ADM) 35
auditd(ADM) 37
auditsh(ADM) 39
authck(ADM) 40
authsh(ADM) 42
autoboot(ADM) 45
backup(ADM) 47
backupsh(ADM) 48
badtrk(ADM) 49
brc(ADM) 52
btldinstall(ADM) 53
captoinfo(ADM) 55
chg_audit(ADM) 58
checkaddr(ADM) 59
checkque(ADM) 60
checkup(ADM) 62

chroot(ADM) 63
cleanque(ADM) 64
cleantmp(ADM) 65
clri(ADM) 66
cnvtmbox(ADM) 67
configure(ADM) 68
consoleprint(ADM) 76
crash(ADM) 77
custom(ADM) 86
dbmbuild(ADM) 89
dbmedit(ADM) 91
dcopy(ADM) 94
deliver(ADM) 95
dial(ADM) 98
diskusg(ADM) 101
displaypkg(ADM) 103
divvy(ADM) 104
dlvr_audit(ADM) 109
dmesg(ADM) 110
dparam(ADM) 111
ecc(ADM) 113
eisa(ADM) 115
fdisk(ADM) 117
fdswap(ADM) 120
ff(ADM) 121
fixmog(ADM) 123
fixperm(ADM) 125
fsave(ADM) 129
fsck(ADM) 134
fsdb(ADM) 148
fsname(ADM) 152
fsphoto(ADM) 153
fsstat(ADM) 155
fstyp(ADM) 156
fuser(ADM) 157
fwtmp(ADM) 159
goodpw(ADM) 161
graph(ADM) 166
grpck(ADM) 168
haltsys(ADM) 169
idaddld(ADM) 170

idbuild(ADM) 171
idcheck(ADM) 173
idinstall(ADM) 176
idleout(ADM) 179
idmkinit(ADM) 180
idmknod(ADM) 182
idspace(ADM) 184
idtune(ADM) 186
infocmp(ADM) 187
initcond(ADM) 192
initscript(ADM) 193
install(ADM) 195
installf(ADM) 197
installpkg(ADM) 201
integrity(ADM) 202
ipcrm(ADM) 204
ipcs(ADM) 205
kbmode(ADM) 209
killall(ADM) 210
labelit(ADM) 211
link(ADM) 212
link_unix(ADM) 213
list(ADM) 214
lpadmin(ADM) 216
lpfilter(ADM) 227
lpforms(ADM) 239
lpsched(ADM) 249
lpsh(ADM) 251
lpusers(ADM) 252
majorsinuse(ADM) 253
makekey(ADM) 254
menumerge(ADM) 255
mkdev(ADM) 257
mkfs(ADM) 261
mmdf(ADM) 264
mmdfalias(ADM) 265
mnlist(ADM) 266
mount(ADM) 267
mountall(ADM) 270
mvdir(ADM) 272
ncheck(ADM) 273

netconfig(ADM) 274
netutil(ADM) 278
nictable(ADM) 280
nlsadmin(ADM) 281
pipe(ADM) 285
pkgadd(ADM) 286
pkgask(ADM) 288
pkgchk(ADM) 290
pkginfo(ADM) 292
pkgmk(ADM) 294
pkgparam(ADM) 296
pkgproto(ADM) 298
pkgrm(ADM) 300
pkgtrans(ADM) 301
profiler(ADM) 303
proto(ADM) 304
pwck(ADM) 306
pwconv(ADM) 307
rc0(ADM) 309
rc2(ADM) 311
reduce(ADM) 313
relax(ADM) 316
relogin(ADM) 318
removef(ADM) 319
restore(ADM) 320
rmail(ADM) 322
removepkg(ADM) 324
rmuser(ADM) 325
runacct(ADM) 327
sag(ADM) 330
sar(ADM) 332
schedule(ADM) 337
sd(ADM) 340
setclock(ADM) 342
setmnt(ADM) 343
settime(ADM) 344
sfmt(ADM) 345
shutdown(ADM) 346
strace(ADM) 348
strclean(ADM) 350
strerr(ADM) 351

submit(ADM) 353
sulogin(ADM) 365
swap(ADM) 366
sync(ADM) 367
sysadmsh(ADM) 368
sysdef(ADM) 370
tcbck(ADM) 371
timex(ADM) 373
tplot(ADM) 375
ttyupd(ADM) 376
uadmin(ADM) 378
umount(ADM) 379
unretire(ADM) 380
uucheck(ADM) 382
uucico(ADM) 383
uuclean(ADM) 385
uudemon(ADM) 387
uuinstall(ADM) 390
uulist(ADM) 391
uusched(ADM) 392
uutry(ADM) 393
uuxqt(ADM) 394
vectorsinuse(ADM) 395
volcopy(ADM) 396
wall(ADM) 398
wtinit(ADM) 399
xbackup(ADM) 400
xdumpdir(ADM) 403
xinstall(ADM) 404
xrestore(ADM) 405
xtd(ADM) 408
xts(ADM) 409
xtt(ADM) 410

File Formats (F)

intro(F) 411
aio(F) 412
aiomemlock(F) 413
archive(F) 414

authcap(F) 415
btld(F) 417
checklist(F) 426
clock(F) 427
compver(F) 428
copyright(F) 429
cpio(F) 430
default(F) 431
depend(F) 432
devices(F) 434
dialcodes(F) 437
dialers(F) 438
filesys(F) 441
fspec(F) 443
gettydefs(F) 445
gps(F) 447
group(F) 450
hs(F) 451
inittab(F) 452
issue(F) 456
logs(F) 457
maildelivery(F) 459
mapchan(F) 462
maxuuscheds(F) 466
maxuuxqts(F) 467
mcconfig(F) 468
mdevice(F) 482
mmdftailor(F) 487
mnttab(F) 497
mtune(F) 498
mvdevice(F) 499
permissions(F) 501
pkginfo(F) 505
pkgmap(F) 508
poll(F) 512
prototype(F) 513
purge(F) 518
queue(F) 519
queuedefs(F) 523
sdevice(F) 524
space(F) 527

stune(F) 528
sysadmcolor(F) 529
sysadmmenu(F) 532
sysfiles(F) 533
systemid(F) 535
systems(F) 537
tables(F) 538
tar(F) 543
term(F) 544
termcap(F) 548
terminfo(F) 560
timezone(F) 562
top(F) 565
ttytype(F) 566
utmp(F) 567
xbackup(F) 569

Hardware Dependent (HW)

intro(HW) 571
80387(HW) 572
audit(HW) 574
boot(HW) 582
cdrom(HW) 604
clone(HW) 605
cmos(HW) 606
dat(HW) 607
fd(HW) 610
hd(HW) 614
keyboard(HW) 619
log(HW) 631
lp(HW) 634
mouse(HW) 636
parallel(HW) 637
prf(HW) 639
ramdisk(HW) 640
rtc(HW) 644
scancode(HW) 645
screen(HW) 648
scsi(HW) 669

serial(HW) 670
streamio(HW) 674
tape(HW) 684
terminal(HW) 699
timod(HW) 700
tirdwr(HW) 702
xt(HW) 704

Preface

This volume is a companion to the *System Administrator's Guide* and contains all commands that are reserved for exclusive use by system administrators.

The manual includes the following sections:

Section	Description
ADM	Administrative Commands — used for system administration.
HW	Hardware device manual pages — information about hardware devices and device nodes.
F	Files — information about system files essential to the operation of SCO UNIX.

For a complete listing of all commands, refer to the Alphabetized List in the *User's Reference.*

System Administration (ADM)

Intro

introduction to system administration commands

Description

This section contains descriptions of the commands that are used to administer and maintain the operating system. These commands are largely root-only, meaning that they can only be executed by the super user (*root*).

accept, reject

allows/prevents print requests to a lineprinter or class of printers

Syntax

/usr/lib/accept ***destinations***
/usr/lib/reject [**-r** [***reason***]] ***destinations***

Description

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

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

-r [***reason***] Associates a ***reason*** with disabling (using **disable**(C)) the printer. The ***reason*** applies to all printers listed up to the next **-r** option. If the **-r** option is not present or the **-r** option is given without a ***reason***, then a default ***reason*** is used. ***reason*** is reported by **lpstat**(C). Please see **disable**(C) for an example of ***reason*** syntax.

File

*/usr/spool/lp/**

See also

disable(C), **enable**(C), **lp**(C), **lpadmin**(ADM), **lpsched**(ADM), **lpstat**(C)

acct: acctdisk, acctdusg, accton, acctwtmp

overview of accounting and miscellaneous accounting commands

Syntax

/usr/lib/acct/acctdisk

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

/usr/lib/acct/accton [*file*]

/usr/lib/acct/acctwtmp *"reason"*

Description

acctdisk - gathers user disk block data

acctdusg - calculates disk consumption for accounting records

accton - starts/stops process accounting

acctwtmp - writes accounting records to standard output

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. When the system is installed, accounting is initially in the "off" state. **acctsh**(ADM) 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 */etc/utmp*, as described in **utmp**(F). The programs described in **acctcon**(ADM) convert this file into session and charging records, which are then summarized by **acctmerg**(ADM).

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

Process accounting and connect time accounting (or any accounting records in the format described in **acct**(FP)) can be merged and summarized into total accounting records by **acctmerg** (see *tacct* format in **acct**(FP)). **prtacct** (see **acctsh**(ADM)) 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 file names 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*. (See **diskusg**(ADM) 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**(S) and **acct**(FP)).

acctwtmp writes a **utmp**(F) 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**(F)). ***reason*** must be a string of 11 or fewer characters, numbers, $, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```
acctwtmp "uname" >> /etc/wtmp
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 this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

See also

acct(S), **acct**(FP), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **acctsh**(ADM), **diskusg**(ADM), **fwtmp**(ADM), **runacct**(ADM), **utmp**(F)

Standards conformance

acctdisk is conformant with:

AT&T SVID Issue 2.

Value added

accton is an extension to AT&T System V provided by The Santa Cruz Operation, Inc.

acctcms

command summary from per-process accounting records

Syntax

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

Description

acctcms reads one or more ***files***, normally in the form described in **acct**(F). 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**(ADM). 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 (that is, UNIX System V/386) 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(ADM), **acct**(S), **acct**(F), **acctcom**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **acctsh**(ADM), **fwtmp**(ADM), **runacct**(ADM), **utmp**(F)

Notes

Unpredictable output results if **-t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

At the beginning of every year, the file */usr/lib/acct/holidays* must be updated to reflect the correct holidays for the new year, or **acctcms** may become confused when attempting to report the prime/non-prime time usage statistics.

Standards conformance

acctcms is conformant with:

AT&T SVID Issue 2.

acctcom

search and print process accounting file(s)

Syntax

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

Description

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

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

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

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

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

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

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

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

(total CPU time)/(elapsed time).

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

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

-m Show mean core size (the default).

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

-t Show separate system and user CPU times.

-v Exclude column headings from the output.

-l ***line*** Show only processes belonging to terminal */dev/* ***line***

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

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

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

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

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

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

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

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

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

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

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

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

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

Files

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

See also

acct(ADM), **acct**(S), **acct**(FP), **acctcms**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **acctsh**(ADM), **fwtmp**(ADM), **ps**(C), **runacct**(ADM), **su**(C), **utmp**(F)

Notes

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

acctcon: acctcon1, acctcon2

connect-time accounting

Syntax

/usr/lib/acct/acctcon1 [*options*]

/usr/lib/acct/acctcon2

Description

acctcon1 - generates per login accounting records

acctcon2 - generates total accounting records

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 IDs, 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 creates 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**(M) and termination of the login shell each generate logoff records, so that the number of logoffs is often three to four times the number of sessions. See **init**(M) and *utmp*(F).

-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**(FP)).

Examples

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

/usr/lib/acct/acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n +2 >ctmp
/usr/lib/acct/acctcon2 <ctmp | acctmerg >ctacct

File

/etc/wtmp

See also

acct(ADM), **acct**(FP), **acct**(S), **acctcms**(ADM), **acctcom**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **acctsh**(ADM), **fwtmp**(ADM), **init**(M), **runacct**(ADM), **utmp**(F)

Note

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

At the beginning of every year, the file */usr/lib/acct/holidays* should be updated with the appropriate holidays for the new year, or **acctcon1** will become confused when attempting to report on prime/non-prime connect times.

Standards conformance

acctcon1 and **acctcon2** are conformant with:

AT&T SVID Issue 2.

acctmerg

merge or add total accounting files

Syntax

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

Description

acctmerg reads its standard input and up to nine additional files, all in the *tacct* format (see **acct**(FP)) or an ASCII version thereof. 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.

Examples

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

acctmerg -v <*file1* >*file2*
... edit file2 as desired ...
acctmerg -i <*file2 file1*>

See also

acct(ADM), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acctprc**(ADM), **acctsh**(ADM), **fwtmp**(ADM), **runacct**(ADM), **acct**(S), **acct**(FP), **utmp**(F)

Standards conformance

acctmerg is conformant with:

AT&T SVID Issue 2.

accton

turn on accounting

Syntax

/usr/lib/acct/accton [*file*]

Description

accton turns process accounting on and off. If no *file* is given then accounting is turned off. If *file* is given, the kernel appends process accounting records. (See **acct**(S) and **acct**(FP)).

Files

/etc/passwd	Used for login name to user ID conversions
/usr/adm/pacct	Current process accounting file
/usr/adm/sulogin	Super user login history file
/etc/wtmp	Login/logout history file

See also

acctcom(ADM), **acct**(S), **acct**(FP), **su**(C), **utmp**(F)

Value added

accton is an extension to AT&T System V developed by The Santa Cruz Operation, Inc.

acctprc: acctprc1, acctprc2

process accounting

Syntax

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

Description

acctprc1 - generates per process accounting records

acctprc2 - generates accounting total records

acctprc1 reads input in the form described by **acct**(FP), adds login names corresponding to user ID, then writes, for each process, an ASCII line detailing user ID login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If *ctmp* is given, it is expected to contain a list of login sessions, in the form described in **acctcon**(ADM), 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 between 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

File

/etc/passwd

See also

acct(ADM), **acct**(S), **acct**(FP), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctsh**(ADM), **cron**(C), **fwtmp**(ADM), **runacct**(ADM), **utmp**(F)

Notes

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

Standards conformance

acctprc1 and **acctprc2** are conformant with:

AT&T SVID Issue 2.

acctsh: chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct

shell procedures for accounting

Syntax

/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 [*file* ...]

/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**(C)). It periodically checks the size of */usr/adm/pacct*. If the size exceeds ***blocks***, 1000 by default, **turnacct will** be invoked with argument **switch**. If the number of free disk blocks in the */usr* file system falls below 500, **ckpacct** will automatically turn off the collection of process accounting records via the **off** argument to **turnacct**. When at least this number of blocks is restored, the accounting will be activated again. 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/default/filesys*. 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 file systems only. If the **-o** flag is used, ***files*** should be mount points of mounted filesystems. If omitted, they should be the special file names of mountable file systems.

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**(C) 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 ensures that 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 **acctcon**(ADM)). It takes one or more file names as arguments; otherwise it reads from the standard input.

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**(ADM)).

shutacct is invoked during a system shutdown to turn process accounting off and append a ***"reason"*** record to */etc/wtmp*.

startup is called by */etc/init.d/acct* to turn the accounting on whenever the system is brought to a multi-user state.

turnacct is an interface to **accton** (see **acct**(ADM)) 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 num-

ber 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. **acct** starts and stops process accounting via **init** and **shutdown** accordingly.

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 (ADM)
/usr/adm/acct/sum	summary directory, should be saved

See also

acct(ADM), **acct**(FP), **acct**(S), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **cron**(C), **diskusg**(ADM), **fwtmp**(ADM), **runacct**(ADM), **utmp**(F)

Standards conformance

chargefee is conformant with:

ANSI X3.159-1989 Programming Language — C.

ckpacct, lastlogin, prctmp, runacct and **shutacct** are conformant with:

AT&T SVID Issue 2.

addxusers

create new user accounts given a traditional password file

Syntax

/tcb/bin/addxusers [**-esuv**] [**-t** *type*] [*file*]

Description

addxusers reads the specified *file*, which should be in traditional **passwd**(FP) format (as found on XENIX systems), and creates the indicated accounts by making equivalent entries in the system's */etc/passwd* file and Protected Password database. The *auth* subsystem and *chown* kernel authorizations are required to run **addxusers**. If no *file* is given, **addxusers** does not attempt to add any new users and only performs certain consistency checks on the existing user accounts. A *file* of "-" means that the standard input should be read.

Login names must begin with a lowercase letter, must not already exist, must not contain a slash (/), and must not be longer than 8 characters.

Numeric user IDs must not be already assigned, and must be in the range 0 to 60000 (inclusive).

Numeric group IDs must be in the range 0 to 60000 (inclusive). Groups which are missing from the file */etc/group* generate a warning, as does membership in a group associated with a protected subsystem.

Encrypted passwords are preserved; that is, users will be able to use their old XENIX passwords to log onto the new system.

Any password-aging information which is present is translated into the equivalent expiration parameters.

The comment field, initial working directory (home directory), and shell program are preserved. Missing or inaccessible directories and shells are warned about, as are non-absolute pathnames. Users should not share home directories.

With the **-u** option, **addxusers** expects *file* to contain a list (one per line) of usernames to add to the Protected Password database. Each user must already have an entry in */etc/passwd* in XENIX format, which is used to make an equivalent entry for the user in the Protected Password database. This allows the system administrator to manually add entries to the */etc/passwd* file, then easily correct the protected password database to reflect these additions.

The **-v** option displays a "being processed" message (which includes the username) for each user **addxusers** attempts to add to the system.

The **-t** option sets the ***type*** of each created user; if omitted, each user is classified as an "individual" person. The legal ***type*** values are:

Number	Equivalent names		Comments
0	root	superuser	All-powerful user (numeric ID 0).
1	operator		Various classifications of
2	sso	security officer	anonymous system administration
3	admin	administrator	accounts.
4	pseudo	pseudo-user	General-purpose anonymous user.
5	general	individual	An individual's personal account.
6	retired		An account which is no longer used.

Normally, only minimal checks for corruption are carried out on the existing */etc/passwd* file before the new users are added: checks are only performed for duplicated login names or numeric user IDs, and bad format. (These are all fatal errors, and prevent any new users from being added.) The **-e** option causes the same checks which are applied to new users to be applied to the existing users (except for membership in a protected subsystem group). The **-s** option checks the existing users for membership of a protected subsystem group. As with new user accounts, not all of the problems which may be discovered are fatal (many are only warnings).

Duplicated group names or numeric group IDs in the */etc/group* file are warned about. However, if a protected subsystem group is corrupted in this way, this is a fatal error (no users are added).

Example

The following steps should be performed when migrating a community of users from a XENIX system:

1. Back up the home directories of the users on the XENIX system using **cpio**(C) or **tar**(C). (Do not back up these files using absolute pathnames. For example, if your accounts are in */usr*, run your backup command from that directory, not from /.)

2. Make a copy of */etc/passwd* and */etc/group* from the XENIX system. (Do not back these files up with absolute pathnames either.)

3. After making certain you are in single user mode, extract the backup of the user's home directories on the new system. For example, if your user accounts reside in */usr*, the files should be extracted in */usr* on the new system. (Note that if you are using a mounted filesystem for your accounts, you must mount it before extracting your backups.)

4. Extract the copy of the *passwd* and *group* files in a temporary directory; for example, */tmp/passwd* and */tmp/group*. Be careful not to overwrite the */etc/passwd* and */etc/group* files on the new system.

5. Edit */tmp/passwd* to remove "system" accounts (such as *root* and *bin*) and any accounts that already exist on the new system.

6. Separate the remaining accounts in */tmp/passwd* (which are to be added to the new system) into different files by user type. For example, place all "pseudo-users" in a file called */tmp/pseudo* and all "individual" users in */tmp/individual*.

7. In your sorted */tmp* account files, you should change login names, numeric user IDs, numeric group IDs, initial working directories, and shell programs as necessary to prevent conflicts with any accounts already on the new system. (If any numeric user or group IDs are changed, it may be desirable to **chown**(C) or **chgrp**(C) the appropriate home directories and their contents on the new system.)

8. Merge */tmp/group* (the saved copy of the XENIX system's */etc/group*) with the new system's */etc/group*; see *group*(F). Again, make certain you are still in single-user mode; if */etc/group* is modified while in multi-user mode, no-one will be allowed to login.

9. Run **addxusers**:

```
addxusers -t pseudo-user /tmp/pseudo 2>&1 | tee -a /tmp/errors
addxusers -t individual /tmp/individual 2>&1 | tee -a /tmp/errors
...
```

(If the */tcb/bin* is not in the *root* **PATH** variable, you must specify the full pathname.) It is advisable to save the standard output and error output of **addxusers** (as shown above) for later analysis and correction.

Finally, use the Accounts ⇨ User ⇨ Examine menu of **sysadmsh**(ADM) to customize the newly-created accounts as needed.

The authorizations may need customization, and accounts which are neither individuals nor retired should have an "account which may su" assigned.

See also

authcap(F), **chgrp**(C), **chown**(C), **cpio**(C), **group**(F), **passwd**(FP), **rmuser**(ADM), **su**(C), **sysadmsh**(ADM), **tar**(C), **tee**(C), **unretire**(ADM)

Notes

When logging in, XENIX truncates passwords to eight (8) characters; SCO System V does not. Therefore, the user must not type more than eight characters when the password from the XENIX system is in effect.

Passwordless accounts and other liberties XENIX allows are more restricted in SCO System V. To continue to use such poor security practices requires customizing the system defaults or the unsecure accounts.

Some standard accounts shipped with the system provoke warnings when the **-e** or **-s** options are specified.

Some vendor's systems support specifying a **nice**(S) value in the comment field, or doing a **chroot**(S) to the home directory (called a sublogin). Both constructions are understood by **addxusers**, and the **nice** value is supported, but sublogins are not in SCO System V and cause a warning.

Value added

addxusers is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

aioinfo

print out AIO statistics

Syntax

aioinfo [*raw-device*]

Description

aioinfo prints out information and internal statistics about AIO configuration and use. By default, it opens */dev/rroot* to get a file descriptor for the AIO **ioctl**; if this is not accessible, or if support for AIO is not linked into the kernel, an alternate AIO disk partition name must be specified.

The following is sample output:

```
total aio_info slots          5
active aio_info slots         2
number of memory locks        1
total locked memory           65536
total ureq structures         120
active ureq structures        23
free ureq structures          97
cumulative ureq usage         51843
maximum ureq usage            73
total aio buf structures      120
free aio buf structures       97
```

In this output, two processes are currently doing AIO. One has 65536 bytes of memory locked, the other does not have a memory lock. 23 AIO requests are currently pending. 51843 AIO requests have been issued since the machine was booted, and the maximum number of simultaneous pending AIO requests since boot was 73.

See also

aio(M), **aiolkinit**(ADM), **aiomemlock**(F)

aiolkinit

set up AIO memory locking permissions

Syntax

/etc/aiolkinit [*raw-device*]

Description

The **aiolkinit** utility allows a system administrator to control which users may lock memory for AIO use, and how much can be locked. This utility reads the */usr/lib/aiomemlock* file, and sets up an internal kernel table entry for each line.

Typically, **aiolkinit** is not invoked directly, but is called by a script in the */etc/rc2.d* directory. Note that this script should be invoked before starting up any program that uses AIO. Invoking **aiolkinit** after boot causes entries that have been added to */usr/lib/aiomemlock* to be revised. Removing entries does not affect their memory locking ability until reboot.

The **aiolkinit** program defaults to opening */dev/rroot* to call the appropriate AIO I/O control command (**ioctl**). If */dev/rroot* does not support AIO, an alternate AIO device must be provided as an argument.

Notes

This command can be run only by the super user.

File

/usr/lib/aiomemlock

See Also

aio(M), **aioinfo**(ADM), **aiomemlock**(F)

ale

lock and update authentication files

Syntax

/tcb/bin/ale ***file program*** [***arguments***]

Description

ale allows the authentication administrator to execute shell scripts that update authentication files while in multiuser mode. The *auth* subsystem and *chown* kernel authorizations are required to run **ale**.

file is the absolute pathname of the authentication file to be locked during the update. ***program*** is the name of the shell script to perform the update, which must reside in the */tcb/lib/auth_scripts* directory. ***arguments*** are the arguments to be passed to the script.

ale participates in the TCB locking protocol in attempting to create a lockfile named ***file-t***. If it is successful, the shell script is executed by the Bourne shell. The script can then edit ***file***, putting the results into ***file-t***. If the script successfully completes its updates, it will exit with a code of 0. This signals **ale** to unlock the file. It renames ***file*** to ***file-o***, ***file-t*** to ***file***, and finally removes ***file-o***. While the ***file-t*** is present, no other utility observing the TCB locking protocol will update ***file***.

If the shell script cannot complete the update it should exit with a code of 1, which tells **ale** a problem has occurred. **ale** then displays an error message, removes ***file-t*** and leaves ***file*** unchanged. If the shell script finds there is no updating to be done it should exit with a code of 2, and **ale** removes ***file-t*** and leaves ***file*** unchanged.

To access authentication files, **ale** executes the shell scripts with both real and effective group IDs set to *auth*, and the user IDs set to the real user ID of the user who called *ale*.

Files

/etc/auth/system/files	File Control database
/etc/group	Group file
*/tcb/files/auth/?/**	User Authentication database
*/etc/auth/**	System Authentication database

See also

authcap(F), **rmuser**(ADM), **ttyupd**(ADM), **unretire**(ADM)

Diagnostics

If **ale** detects an error, it displays an appropriate error message and exits with code 1. Otherwise **ale** returns the exit status of ***program***.

Notes

ale checks the permissions on the complete paths of ***file***, ***program*** and the File Control database itself against their entries in the File Control database. If any discrepancies are found an appropriate "may be compromised" message (including the pathname) is displayed and an entry is written to the audit trail. **integrity**(ADM) and fixmog ADM can be used to analyze and fix the problem.

Care should be taken when writing scripts which update authentication data. If files are incorrectly updated it could cause the system to refuse further log-ins.

Value added

ale is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

ap

generate account profile for propagation to other machines

Syntax

ap -d [-v] [*usernames*]

ap -r -f *file* [-o] [-v] [*usernames*]

Description

ap allows the propagation of user accounts by generating an archive that can be loaded on other machines.

ap -d writes an account profile entry to the standard output for each username specified. If no ***usernames*** are specified, account profiles are written for all users listed in the password file.

ap -r restores account profile information from the file specified by the **-f** option, which is assumed to be the product of a previous **ap -d**. If no ***usernames*** are specified, all the account profiles contained in the file are restored: otherwise only the account profiles for the specified users are restored.

An account profile entry consists of the user's line from the password file followed by all relevant parts of their Protected Password database entry. The following Protected Password database fields are irrelevant and are not copied:

Time of last unsuccessful password change.
Time of last successful and last unsuccessful login.
Terminal of last successful and last unsuccessful login.
Number of consecutive unsuccessful logins.

The **-v** (verbose) option causes **ap** to output a message to the standard error for each account profile dumped or restored.

The **-o** (overwrite) option causes **ap** to overwrite an existing account profile which has the same username and user ID as one being restored. If the **-o** option is not specified a message is output and existing entries are not overwritten.

Examples

To dump the account profiles for users *root* and *guest* to a file called *profiles* and display a message after each account profile is dumped:

ap -dv root guest > profiles

This file can then be transferred to another machine. To restore the account profile for user *root*, overwriting any existing profile:

ap -ro -f profiles root

Files

/etc/passwd	Password file
/etc/shadow	Shadow Password file
*/tcb/files/auth/?/**	Protected Password database
*/etc/auth/subsystems/**	Subsystem Authorizations database

See also

addxusers(ADM), **authck**(ADM), **authcap**(F), **fields**(S), **getprpwent**(S), **getpwent**(S), **passwd**(FP), **subsystems**(S)

Diagnostics

If **ap** detects a fatal error, it displays an appropriate error message and exits with status greater than zero. If no errors are encountered, **ap** exits with status zero.

Notes

ap requires the invoking user to be the super user or have the *auth* subsystem authorization, and have both the *chown* and *execsuid* kernel authorizations.

As different machines may have different System Default values, the same profile transferred to another machine may give the user different capabilities simply because different default values are picked up for fields not present in the user's Protected Password database entry.

As the file containing the dumped account profile information is used to update the password and Protected Password database, it must be protected from unauthorized access in the same way the Protected Password database entries themselves are protected.

Value added

ap is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

asktime

prompt for the correct time of day

Syntax

/etc/asktime

Description

asktime - prompt for the correct time of day

asktimer - is a link to **/etc/asktime**.

This command prompts for the time of day. You must enter a legal time according to the proper format as defined below:

[[*yy*]*mmdd*]*hhmm*

Here the first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24-hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. The current year is the default if no year is mentioned.

Examples

This example sets the new time, date, and year to "11:29 April 20, 1995".

```
Current system time is Wed Nov  3 14:36:23 PST 1994
Enter time ([yymmdd]hhmm): 9504201129
```

Diagnostics

If you enter an illegal time, **asktime** prompts with:

```
Try again:
```

Notes

asktime is normally performed automatically by the */etc/rc2* system startup scripts immediately after the system is booted; however, it may be executed at any time. The command is privileged, and can only be executed by the super user.

Systems which autoboot will invoke **asktime** automatically on reboot. On these systems, if you don't enter a new time or press ⟨Return⟩ within 1 minute of invoking **asktime**, the system will use the time value it has. If ⟨Return⟩ alone is entered, the time is unchanged.

Value added

asktime is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

asroot

run a command as root

Syntax

/tcb/bin/asroot *command* [*args*]

Description

asroot allows an authorized user to run a command as superuser (*root*). Commands that can be used with **asroot** are defined by the super user (see "Making a command executable under asroot") and must be present in the */tcb/files/rootcmds* directory. Only *root* can make entries in this directory.

To use **asroot**, the user must have either the *root* primary subsystem authorization (which allows any command in the *rootcmds* directory to be run) or have a secondary subsystem authorization with the same name as the command. In addition to one of these the user must also have the *execsuid* kernel authorization.

By default, **asroot** asks the user for their account password before executing the command. (This prevents an unauthorized user from using a terminal which an authorized user has left without logging out.) This feature can be turned off by entering the line "**ASROOTPW=NO**" in */etc/default/su*. **asroot** also logs its use by making entries in the **SULOG** logfile as configured in */etc/default/su*.

If the command to run is a shell script then it will be executed by the Bourne (**/bin/sh**) shell. The setting of the **SHELL** environment variable is not considered.

Making a command executable by asroot

To make a command executable by **asroot**, log in as *root* and do the following:

1. Copy the desired command into the */tcb/files/rootcmds* directory. Do not create a link if the permissions on the file are less restrictive than those listed in the File Control database.
2. Change the permissions on the file to match those specified in the File Control database. This can be done most conveniently with the **fixmog**(ADM) command.
3. Edit the authorizations file */etc/auth/system/authorize* and add a comma and the name of the new command to the end of the line beginning with "root:". This declares a new secondary subsystem authorization that can be given to users like any other authorization with the **sysadmsh**(ADM) Accounts ⇨ User ⇨ Examine:Privileges selection. Users can only execute the command with **asroot** if they have the *root* authorization or the authorization corresponding to the name of the command.

Default asroot commands

By default one command is shipped in the */tcb/files/rootcmds* directory: the **shutdown**(ADM) command. Only trusted users should be given the *root* authorization.

Files

/tcb/files/rootcmds	asroot commands
/etc/auth/system/authorize	subsystem authorizations
/etc/auth/system/files	File Control database
/etc/default/su	**ASROOTPW** and **SULOG** settings

See also

authsh(ADM), **fixmog**(ADM), **integrity**(ADM), **subsystems**(S)

Diagnostics

asroot returns an exit code of 1 when:

1. the length of the command name is greater than 16 characters
2. the user is not authorized to run the command
3. the command's execution bits in the */tcb/files/rootcmds* directory are not set properly
4. an integrity violation is detected
5. an authentication error is detected
6. an incorrect user password is entered

asroot will also return an exit code of 2 when no command name is given or exit code of 3 if the command cannot be executed.

Notes

asroot checks the permissions of the complete pathname of all files it uses. If any component of a path does not match its entry in the File Control database, an integrity violation is reported. Run **integrity**(ADM) or **fixmog**(ADM) to discover where the integrity violation has occurred.

Care must be taken, when choosing commands to be executed by **asroot**, that the *root* privilege is not given away accidentally. For example, if **sysadmsh**(ADM) were to be run via **asroot** then any shell escapes would also run as *root*.

A line in */etc/auth/system/authorize* cannot exceed 1024 characters in length and the sum of the number of primary and secondary authorizations cannot exceed 32.

Value added

asroot is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

atcronsh

at and cron administration utility

Syntax

/usr/lib/sysadm/atcronsh

Description

atcronsh is the screen interface invoked by the **sysadmsh**(ADM) Jobs ⇨ Authorize selection. It is used to specify users allowed to use the **cron**(C), **at**(C) and **batch** (see **at**(C)) commands. It also allows the **at**(C) and **batch** prototype files to be edited.

The program allows a system default for **cron**(C), **at**(C) and **batch** to be given. The defaults can be:

none — No user authorized

allow — All users allowed to use the commands unless a user is specifically denied

deny — All users denied to use the commands unless a user is specifically authorised

The default setting decides whether an allow or deny file is to be used (deny file means */usr/lib/cron/cron.deny* or *at.deny*, allow file means *at.deny* or */usr/lib/cron/cron.deny*).

For each user (unless the none system default has been chosen), a specific authorization for **cron**(C), **at**(C) and **batch** may be given. The allow and deny files are interpreted as follows:

- if an allow file exists, and the user name appears in it, the user is allowed access.
- if an allow file exists, access is denied
- if a deny file exists and the user name appears in it, access is denied
- if a deny file exists, access is allowed
- access is denied

Files

/usr/lib/cron/cron.allow
/usr/lib/cron/cron.deny
/usr/lib/cron/at.allow
/usr/lib/cron/at.deny

See also

at(C), **auditsh**(ADM), **authsh**(ADM), **backupsh**(ADM), **cron**(C), **lpsh**(ADM), **sysadmsh**(ADM)

Notes

Invoking **atcronsh**(ADM) is not recommended; use the **sysadmsh**(ADM) Jobs ⇨ Authorize selection.

Value added

atcronsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

auditcmd

command interface for audit subsystem activation, termination, statistic retrieval, and subsystem notification

Syntax

/tcb/bin/auditcmd [-e] [-d] [-s] [-c] [-m] [-q]

Description

The **auditcmd** utility is used to control the audit subsystem. This command may only be executed by processes with the **configaudit** kernel authorization since the audit device is used.

auditcmd allows the specification of the following options:

-e Enable the audit subsystem for audit record generation. The enabling of the audit subsystem initializes subsystem parameters from the */tcb/files/audit/audit_parms* file. This file is established using the **sysadmsh**(ADM) Audit selections.

-s Inform the audit subsystem that a system shutdown is in progress. The subsystem will continue audit record generation to a temporary directory on the root file system. The audit daemon is also modified so that it will survive the shutdown. The subsystem will continue to generate audit records until disabled.

-d Disable the audit subsystem. All audit record generation ceases and a termination record is written to the audit trail. This record results in the termination of the audit daemon. The subsystem properly synchronizes to ensure that the audit daemon has read all records from the audit trail before the system is allowed to terminate.

-m Inform the audit subsystem that multi-user run state has been achieved and that alternate audit directories specified by the administrator using **sysadmsh** are now mounted and available.

-c Retrieve audit subsystem statistics from the audit device.

-q Perform the specified option silently. Do not report errors attributable to the audit subsystem not being enabled at the moment.

See also

audit(HW)

"Using the audit subsystem," chapter of the *System Administrator's Guide.*

Diagnostics

auditcmd returns 0 on success, 1 on command line argument error, and -1 on failure actions. Reasons for failure include parameter file inconsistencies, lack of permission, and security database inconsistency.

Value added

auditcmd is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

auditd

read audit collection files generated by the audit subsystem and compact the records

Syntax

/tcb/bin/auditd [-y] [-n]

Description

auditd is the audit daemon process which is spawned whenever the audit subsystem is enabled. The audit subsystem continually generates audit records writing them to intermediate files called audit collection files. At any time, there may be many collection files since the subsystem continually switches files to ensure that no single file grows excessively large.

The daemon is responsible for reading the audit collection file records from the subsystem, compacting them to provide space savings, and writing the compacted records to files which will later be used for reduction. To read the records from the subsystem, the daemon uses the */dev/auditr* device. The daemon exclusively reads this file which is managed by the subsystem. Each read request returns a block of data from a collection file. The audit subsystem insures that the data is returned in the proper order and also handles file management associated with the multiple collection files. This provides the daemon with a single read focal point.

As a block of data is returned to the daemon, it is optionally compacted and the record, with its size prepended, is written to the current audit output file. Like the audit subsystem, the daemon is capable of writing many different output files in a number of administrator-specified directories to avoid overflowing any one file system. As each output file is written, the daemon records the name in a log file which is used by the reduction program. This log file provides an output file trail alleviating the need for the administrator to keep up with file generation or to recreate the sequence of output file writing. The compaction of output files and the selection of audit directories is controlled by the administrator interface utility **auditsh**(ADM).

Each time the audit subsystem is enabled, a new audit session is created. The session is identified by a session ID which is used to stamp the output files generated by the audit daemon and the log file that identifies them. **auditif** is used to examine daemon log files in the */tcb/files/audit* directory to identify the session and the date/time of the start and end of the session. In this manner, the administrator need not know the session ID but only the dates for which data reduction is desired.

When the daemon is started, a recovery mechanism is invoked to determine if the previous audit session was terminated normally. If abnormal termination occurred, there may be audit records written by the subsystem to collection files that were not read by the daemon and compacted to an audit output file.

The daemon recovery mechanism provides the capability to recover these records and update the output files from the previous session as necessary. The recovery mechanism will interactively query whether recovery is desired if abnormal termination occurred. The **-y** and **-n** options may be used to avoid the interactive question.

The daemon also provides a mechanism whereby applications that are not privileged to open and write audit records to the audit device are able to send the daemon audit records. These are, in turn, written to the audit subsystem. To provide this service, the daemon creates a message queue which only certain applications with specific permission are able to send messages to. When one of the applications wishes to generate an audit record using this mechanism, the record is first constructed and then written to the message queue. The specific message queue is identified in the file */tcb/files/audit/audit_dmninfo*. This file contains the **audit_dmninfo** structure which is defined in the include file *sys/audit.h*. The first field is the process ID of the daemon and the second is the message queue identifier. After the message has been written to the queue by the application, the application will generate a **SIGUSR1** to the daemon indicating a message is waiting. The daemon responds by reading the message queue and writing the record to the audit subsystem device.

Files

/dev/auditr
/dev/auditw
/tcb/files/audit/audit_dmninfo
/tcb/files/audit/CAFLOG.xxxxxx

See also

audit(HW)

"Using the audit subsystem," chapter of the *System Administrator's Guide*

Diagnostics

Upon successful completion at the termination of auditing by the subsystem, the program exits with a status of 0. Otherwise, a diagnostic message is printed and the program exits with a status of -1.

Value added

auditd is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

auditsh

menu driven audit administration utility

Syntax

/usr/lib/sysadm/auditsh

Description

auditsh is the screen interface invoked by the **sysadmsh**(ADM) System ⇨ Audit selection. This selection controls the audit subsystem, allowing establishment of audit subsystem initialization parameters, specification of criteria for selecting output records during reduction, report generation, dynamic changing of subsystem parameters, and backup and restore of compacted audit output files.

If the environment variable **PAGER** is set, the specified program is used to display reports sent to the terminal.

See also

atcronsh(ADM), **auditcmd**(ADM), **auditd**(ADM), **authsh**(ADM), **backupsh**(ADM), **lpsh**(ADM), **reduce**(ADM), **sysadmsh**(ADM)

Note

Invoking **auditsh**(ADM) is not recommended; use the **sysadmsh**(ADM) System ⇨ Configure ⇨ Audit selection.

Value added

auditsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

authck

check internal consistency of authentication database

Syntax

/tcb/bin/authck [-p] [-t][-a][-s [-n | -y]][-v]

Description

authck checks both the overall structure and internal field consistency of all components of the Authentication database. It reports all problems it finds. The options and tests are as follows:

-p Check the Protected Password database. A number of tests are performed. The Protected Password database and */etc/passwd* are checked for completeness such that neither contains entries not in the other. Once this is done, the fields common to the Protected Password database and */etc/passwd* are checked to make sure they agree. Then, fields in the Protected Password database are checked for reasonable values. For instance, all time stamps of past events are checked to make sure they have times less than that returned by **time**(S).

-t The fields in the Terminal Control database are checked for reasonable values. All time stamps of past events are checked to make sure they have times less than returned by **time**.

-s The Protected Subsystem database files are checked to ensure they correctly reflect the subsystem authorization entries in the Protected Password database. Each name listed in each subsystem file is verified against the Protected Password entry with the same name, so that no authorization is inconsistent between the files. Also, each Protected Password entry is scanned to ensure that all the privileges listed do in fact get reflected in the Protected Subsystem database. If any inconsistencies are found and neither the **-n** or **-y** flags have been given, the administrator is asked whether **authck** should repair the Subsystem database. The **-y** flag makes **authck** repair the database without asking first and the **-n** flag makes **authck** abort the repair phase.

-a This option is shorthand for turning on all the **-p**, **-t**, and **-s**, options.

-v This option provides running diagnostics as the program proceeds. It also produces warnings on events that should not occur but otherwise do not harm the Authentication database and the routines operating on it.

Files

/etc/passwd	System password file
*/tcb/files/auth/?/**	Protected Password database
/etc/auth/system/ttys	Terminal Control database
/etc/auth/system/files	File Control database
*/etc/auth/subsystems/**	Protected Subsystem database
/etc/auth/system/default	System Defaults database

See also

authcap(F), **getprpwent**(S), **getprtcent**(S), **getprfient**(S), **getprdfent**(S), **integrity**(ADM), **subsystem**(S)

"Maintaining System Security" chapter of the *System Administrator's Guide*

Notes

authck requires the invoking user to be root or have the **auth** subsystem authorization. The **chown** kernel authorization is also required for **authck** to repair the subsystem databases.

Value added

authck is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

authsh

administrator interface for authorization subsystem

Syntax

/usr/lib/sysadm/authsh

Description

authsh is the screen interface invoked by the **sysadmsh**(ADM) Accounts selection to administer the authorization subsystem. It is a full screen menu-driven interface that provides the functions necessary to control the generation and maintenance of user and system passwords, the terminal database configuration, terminal and account locking, and the generation of administrator reports on system activity.

The functions supported by the main level menu are:

User — This category of screen interfaces is provided for the setup and maintenance of user accounts and user account passwords. The screens are used to add, update, display, and delete user accounts from the system. Also, modifications to user account passwords or modifications to the various criteria controlling the generation of account passwords is accomplished using this menu option.

Defaults — These options are provided for the maintenance of system-wide parameters like default privileges, password expiration, password lifetime, single-user password requirement, restrictive password generation, and the delay time between login attempts. These parameters apply on a global system basis rather than a user account basis.

Terminal — The terminal database interface screens are used for the maintenance of the database entries to support the addition, deletion, and update of terminal information. Additionally, this category includes the necessary screens for setting and clearing locks on specific terminals.

Report — This category provides the administrator with a method of generating various reports on system activity. Report types include password database, terminal database, and login activity reports.

Check — This option provides the administrator with a consistency check on databases (protected password, terminal control database, and subsystem database) and the password file (*/etc/passwd*). The password check returns system account warning messages. This option is not normally used.

See also

passwd(C)

"Maintaining system security," chapter of the *System Administrator's Guide*

Files

/etc/group
/etc/passwd
*/tcb/files/auth/[a-z]**
*/etc/auth/subsystems/**
*/etc/auth/system/**
/etc/default/authsh

/etc/default/authsh fields

The field values of */etc/default/authsh* are:

LOGIN_GROUP Name of default login group. Must exist in */etc/group*.

OTHER_GROUPS List of groups the user is to be a member of. Each group listed must exist in */etc/group*. The **LOGIN_GROUP** does not need to be included in this list. The groups in the list may be separated by commas (,) or spaces.

SHELL Name of default login shell, either the name of a shell defined in */usr/lib/mkuser*, or the full pathname of an executable file. Note that the empty name is legal but is not equivalent to either **sh** or **/bin/sh**.

HOME_DIR Default absolute pathname of parent directory of user's home directory. The home directory itself has the same name as the user. This parent directory must be r/w/x by group *auth*.

HOME_MODE Default permissions for the user's home directory. Note that both **HOME_DIR** and **HOME_MODE** default settings can be overridden on a shell-specific and/or path-specific basis.

USER_TYPE Default type of user:

Individual — Individual's personal account, used by one person, and one person only.

Operator

Administrator

Security Officer — Various classifications of accounts potentially used by more than one individual.

Pseudo-user — Anonymous account never directly used by a user.

All user types except Individual must have an associated account which is allowed to **su**(C) to the user.

UID — **MIN_ADMIN_UID** to **MAX_ADMIN_UID**, inclusive: UID values the administrator may choose.

MIN_SUGGEST_UID to **MAX_SUGGEST_UID**, inclusive: UID values the system may suggest.

Note that UIDs less than 200 are reserved and should not be used.

GID — Similar to UID ranges.

Note that GIDs less than 100 are reserved and should not be used.

MIN_USER_NAME — Minimum length of an acceptable user name (default: 3 characters).

MAX_USER_NAME — Maximum acceptable length of a user name (default: 8 characters).

MIN_GROUP_NAME — Minimum length for a group name (default: 3 characters).

MAX_GROUP_NAME — Maximum length for a group name (default: 8 characters).

Note

Invoking **authsh**(ADM) is not recommended; use the **sysadmsh**(ADM) Accounts selection.

Value added

authsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

autoboot

automatically boot the system

Description

The system can be set up to go through the boot stages automatically (as defined in */etc/default/boot*) when the computer is turned on (booted), provided no key is pressed at the **boot**(HW) prompt.

If **boot** times out and **AUTOBOOT=YES**, then the word "auto" is passed in the boot string and **init**(M) is passed a **-a** flag.

In addition, the **TIMEOUT** entry can be set to specify the number of seconds to wait before timing out.

The **autoboot** procedure checks the file */etc/default/boot* for the following instructions on autobooting:

AUTOBOOT=YES or **NO** — Whether or not **boot**(HW) times out and loads the kernel. **boot** looks for this variable in the */etc/default/boot* file on its default device.

MULTIUSER=YES or **NO** — Whether or not **init**(M) invokes **sulogin** or proceeds to multi-user mode.

PANICBOOT=YES or **NO** — Whether or not the system reboots after a **panic**(). This variable is read from */etc/default/boot* by **init**.

RONLYROOT=YES or **NO** — Whether or not the root filesystem is mounted **readonly**. This must be used only during installation, and not for a normal boot. It will effectively prevent writing to the filesystem.

DEFBOOTSTR=*bootstring* — Set default bootstring to ***bootstring***. This is the string used by **boot** when the user presses ⟨Return⟩ only to the "Boot:" prompt, or when **boot** times out.

SYSTTY=*x* — If ***x*** is **1**, the system console device is set to the serial adapter at COM. If ***x*** is **0**, the system console is set to the main display adapter.

SLEEPTIME=*n* — Sets the time (in seconds) between calls to **sync**.

TIMEOUT=*n* — Where ***n*** is the number of seconds to timeout at the "Boot:" prompt before booting the kernel (if **AUTOBOOT=YES**). If **TIMEOUT** is unspecified, defaults to one minute.

If either the */etc/default/boot* file or the variable needed cannot be found, the variable is assumed to be **NO**. However, if the filesystem cannot be found, **PANICBOOT** is set to **YES**.

If the UNIX mail system, **mail**(C), is installed on the system, the output of the boot sequence is mailed to *root*. Otherwise, the system administrator should check the file */etc/bootlog* for the boot sequence output. The output of **fsck**(ADM) is temporarily saved in the file */dev/recover* before it is moved to */etc/bootlog* and finally may be sent to the system administrator via **mail**.

Other boot options which take affect during **autoboot** are documented on the **boot**(HW) manual page.

Files

/etc/bootlog	**boot** output log for autobooting systems
/etc/default/boot	**boot** information file
/etc/rc2	instructions for entering multi-user mode, includes mounting and checking additional filesystems
/etc/sulogin	executed at startup, prompts the user to press ⟨Ctrl⟩**d** for multi-user mode or to enter the root password for maintenance mode
/dev/recover	allows saving of **fsck** output
/dev/scratch	temporary **fsck** file for large filesystems

See also

boot(HW), **fsck**(ADM), **init**(M)

Notes

The utilities invoked during the boot procedure are passed the *-a* flag and time out only when the system autoboots. For example, **asktime**(ADM) times out after 30 seconds when the system **autoboots**, but waits for a response from the user any other time it is invoked.

The previous **boot** modes of **AUTO=CLEAN, DIRTY, NEVER** have been retained for backwards compatibility, but are ignored if any of the newer modes are present.

Value added

autoboot is an extension to AT&T System V developed by The Santa Cruz Operation, Inc.

backup

performs UNIX backup functions

Syntax

backup [**-t**] [**-p** | **-c** | **-f** *files* | **-u** "*user1* [*user2*]"] **-d***device*

backup -h

Description

The UNIX **backup** utility is a front-end for the **cpio**(C) utility. Use **restore**(ADM) to restore backups made with this utility. It is not recommended for routine system backups; use the **sysadmsh**(ADM) interface for system backups.

-h produces a history of backups. Tells the user when the last complete and incremental/partial backups were done.

-c complete backup. All files changed since the system was installed are backed up.

-p incremental/partial backup. This option backs up only the files that have been modified since the date of the last backup. A complete backup must be performed before a partial backup.

-f backup files specified by the <*files*> argument. File names may contain characters to be expanded (that is, *, .) by the shell. The argument must be in quotes.

-u backup a user's home directory. All files in the user's home directory will be backed up. At least one user must be specified but it can be more. The argument must be in quotes if more than one user is specified. If the user name is "all", then all the user's home directories will be backed up.

-d used to specify the device to be used. It defaults to */dev/rdsk/f0q15d* (the 1.2M floppy).

-t used when the device is a tape. This option must be used with the **-d** option when the tape device is specified.

A complete backup must be done before a partial backup can be done. Raw devices rather than block devices should always be used. The program can handle multi-volume backups. The program will prompt the user when it is ready for the next medium. The program will give you an estimated number of floppies/tapes that will be needed to do the backup. Floppies MUST be formatted before the backup is done. Tapes do not need to be formatted, except mini-cartridge tapes. If backup is done to tape, the tape must be rewound.

xbackup is the equivalent utility for XENIX filesystems.

See also

restore(ADM)

backupsh

menu driven backup administration utility

Syntax

/usr/lib/sysadm/backupsh

Description

backupsh is the screen interface invoked by the **sysadmsh**(ADM) Backups selection to administer the backup subsystem. **backupsh** allows scheduled and non-scheduled backups to be taken. Complete filesystems or single files or directories may also be restored. It also allows the */usr/lib/sysadmin/schedule* file to be edited.

backupsh can be used with both UNIX and XENIX filesystems. If a UNIX filesystem is being used then **backupsh** calls **cpio**(C): if a XENIX filesystem is being used then **backupsh** calls **xbackup**(ADM) or **xrestore**(ADM).

Refer to **atcronsh**(ADM) for details of environment variables that **backupsh** uses, the usage is the same except that **backupsh** uses the specific variable **BACKUP** instead of **ATCRON**.

File

/usr/lib/sysadmin/schedule

See also

at(C), **atcronsh**(ADM), **auditsh**(ADM), **authsh**(ADM), **backup**(ADM), **cpio**(C), **cron**(C), **lpsh**(ADM), **restore**(ADM), **sysadmsh**(ADM)

Note

Invoking **backupsh**(ADM) is not recommended; use the **sysadmsh**(ADM) Backups selection.

Value added

backupsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

badtrk

scan fixed disk for flaws and creates bad track table

Syntax

/etc/badtrk [-e [-m *max*]] [-s *qtdn*] [-v] [-f *device*]

Description

Used chiefly during system installation, **badtrk** scans the media surface for flaws, creates a new bad track table, prints the current table, and adds and deletes entries in the table. Bad tracks listed in the table are "aliased" to good tracks, such that when a process tries to read or write a track listed in the bad track table, one of the replacement tracks is used instead. These replacement tracks are allocated when **badtrk** is run during installation. Changing the number of replacement tracks allocated may require re-installation of the operating system, so the number of replacement tracks allocated should be fairly large.

Options

-f *device*
Opens the partition ***device*** and reads the bad track table associated with that partition. ***device*** must be a UNIX partition of a fixed disk: */dev/rhd0a* for the first drive, */dev/rhd1a* for the second, and so on. The default is */dev/rhd0a.*

-e
Used by the installation procedure, the **-e** flag causes **badtrk** to change the size of the bad track table.

WARNING: The **-e** flag should not be invoked by the user. Use of the **-e** may restructure the hard disk, rendering much of the information stored on it unusable.

-m *max*
Used only in non-interactive mode in conjunction with **-e**, **-m** sets the maximum number of bad tracks to ***max***.

-s *arguments*
Invokes **badtrk** non-interactively, causing it to scan the disk for bad tracks and enter any errors found in the bad track table. The ***arguments*** specify either quick or thorough, and either destructive or non-destructive scan:

[q]uick
[t]horough
[d]estructive
[n]on-destructive

The user should specify either **q** or **t**, and either **d** or **n**.

-v Used only in non-interactive mode in conjunction with **-e**, **-v** displays progress messages indicating how much of the disk has been scanned.

Usage

When **badtrk** is executed interactively, the program first displays the main menu:

```
1.      Print Current Bad Track Table
2.      Scan Disk (You can choose Read-Only or Destructive later)
3.      Add Entries to Current Bad Track Table by Cylinder/Head Number
4.      Add Entries to Current Bad Track Table by Sector Number
5.      Delete Entries Individually From Current Bad Track Table
6.      Delete All Entries From Bad Track Table

        Enter your choice or q to quit:
```

You are prompted for option numbers, and, depending upon the option, more information may be queried for later.

A bad track table (option 1) might look like this:

```
Defective Tracks

        Cylinder        Head            Sector Number(s)
-------------------------------------------------------
1.      190             3               12971-12987
```

Option 2 scans the disk for flaws. If changes have been made to your bad track table since you last updated the table on disk (or since you entered **badtrk**), you will be asked if you want to update the disk with the new table before scanning. You should answer "y" to save your changes, "n" if you don't want to save changes made up to this point. Next you are prompted to specify the kind of scan you wish to perform: either quick or thorough, and either destructive or non-destructive. Choosing a destructive scan will cause all data in the scanned region to be lost. After you respond to these prompts, **badtrk** begins its scan. You can interrupt a scan by typing "q" at any time. You are then prompted to continue the scan or return to the main menu.

As the program finds flawed tracks, the location of each bad track is displayed. An example error message might be:

```
wd: ERROR : on fixed disk ctlr=0 dev=0/47 block=31434 cmd=00000020
    status=00005180, sector = 62899, cylinder/head = 483/4
```

(You may see this kind of message if there is a read or write error during the scanning procedure.)

When the scan is complete, the main menu reappears. The program automatically enters any detected flaws in the bad track table.

If your disk is furnished with a flaw map, you should enter these flaws into the bad track table. Select either option 3 or 4, depending upon the format of the flaw map furnished with your disk. Enter the defective tracks, one per line. (This should only be done on non-remapped drives; see cautions under "Notes".)

When you are satisfied that **badtrk** contains a table of the desired flaws, quit the **badtrk** program by entering "q" at the main menu.

If **badtrk** was invoked with the **-e** flag (which should only occur when called by **mkdev hd** during the installation procedure), and the disk contains a valid division table, the following message is displayed prior to the **badtrk** menu:

```
This device contains a valid division table.  Additional
(non-root) filesystems can be preserved across this reinstallation.
If you wish to be able to preserve these file systems later, you must
not change the current limit of the bad track table, which is
n bad tracks.  Do you wish to leave it unchanged? <y/n>:
```

If you respond "y", you will not be prompted later to enter a new limit for the size of your bad track table. You can add or delete entries, but you will not be allowed to increase the maximum number of bad tracks allocated. If you respond "n" and the size of your bad track table is changed, your disk division table will be destroyed.

If you do not have a valid disk table or you selected "n" when prompted, you are prompted for the number of replacement tracks to allocate. There will be a recommended number of replacement tracks to allocate based on the number of known bad tracks plus an allowance for tracks that may go bad in the future. You should choose to allocate at least the recommended number of replacement tracks. Make your choice carefully, because if you want to change this amount later, you will have to reinstall.

Before exiting, **badtrk** will ask whether you wish to update the device with the new bad track table. If you wish to save you changes, answer "y". If you wish to leave the bad track table as it was before running **badtrk**, answer "n".

Notes

This utility only applies to standard disk controllers and not SCSI host adapters or IDA controllers. **badtrk** can only be used in single-user mode.

If a bad spot develops in the boot blocks or system tables at the very beginning of the partition, reinstallation is required.

Some disk controllers support alternate modes known as "translation", "mapping" or "63-sector" modes, that change the apparent shape of the drive. This is often used to make a drive that has more than 1024 cylinders appear to have less cylinders in order to make it compatible with MS-DOS. If your drive has been formatted using one of these options, do not use options 3 and 4 to manually add entries to the bad track.

File

/etc/badtrk

Value added

badtrk is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

brc, bcheckrc

system initialization procedures

Syntax

/etc/bcheckrc [-a]

/etc/brc

Description

brc - clears mounted filesystem table and adds entry for root filesystem

bcheckrc - checks root filesystem

These shell procedures are executed via entries in */etc/inittab* by **init**(M) whenever the system is booted (or rebooted).

First, the **bcheckrc** procedure checks the status of the root filesystem. If the root filesystem is found to be bad, **bcheckrc** repairs it. When invoked with the **-a** (autoboot) flag, **bcheckrc** will run without operator intervention. **init** calls **bcheckrc** with the **-a** flag when the system autoboots.

Then, the **brc** procedure clears the mounted filesystem table, */etc/mnttab*, and puts the entry for the root filesystem into the mount table.

After these two procedures have executed, **init** checks for the "initdefault" value in */etc/inittab*. This tells **init** in which run-level to place the system. Since "initdefault" is initially set to **2**, the system will be placed in the multi-user state via the **/etc/rc2** procedure.

Note that **bcheckrc** should always be executed before **brc**. Also, these shell procedures may be used for several run-level states.

See also

boot(HW), **fsck**(ADM), **init**(M), **rc2**(ADM), **shutdown**(ADM)

btldinstall

install boot-time loadable device drivers into the Link Kit

Syntax

/etc/btldinstall ***mount_dir***

Description

This command asks the user which packages on a boot-time loadable device driver diskette (the **btld**(F) diskette) are to be installed, and then adds the appropriate drivers to the Link Kit. A "package" is a directory hierarchy which contains one or more drivers, and associated files.

mount_dir is expected to be the root directory of a mounted btld diskette. **btldinstall** is typically run by a Bourne shell (**sh**(C)) script (**/install/INSTALL**) which is always present on a btld diskette. **/install/INSTALL** is itself run automatically by **installpkg**(ADM).

btldinstall performs the following actions, in order:

a) Ensure the Link Kit is installed. If it is not, the user is asked whether to install it using **custom**(ADM), or to abandon the installation of boot-time drivers.

b) Asks the user which boot-time loadable device driver packages to install. The default behavior is that **btldinstall** installs all the packages on the btld diskette which are also in the *package string* (see **string**(M)).

Then once per required package:

c) If the Bourne shell script **/pkg/install/copyright** exists and is executable it is run.

d) If the Bourne shell script **/pkg/install/preinstall** exists it is run as though it were part of **btldinstall**.

e) The list of drivers in */pkg/install/drivers* is checked to determine if a driver with the same name already exists in the Link Kit. The user is asked to resolve this conflict, either by replacing the driver in the Link Kit or by choosing not to install the boot-time loadable driver.

f) The drivers are installed with **idinstall**(ADM).

g) If the hierarchy */pkg/new* exists, it is copied to the hard disk as if it were "**/**".

h) If the file */pkg/install/pkg.name* exists it is copied to */usr/options/pkg.name*.

i) If the Bourne shell script **/pkg/install/postinstall** exists it is run as though it were part of **btldinstall**.

The following environment variables are available for use in scripts run by **btldinstall**:

$pkg	The current package being installed.
$pkginst	All packages to be installed.
$okdrivers	All drivers to be installed for this package.

See also

boot(HW), **btld**(F), **idinstall**(ADM), **installpkg**(ADM), **string**(M)

Notes

If an error occurs during the installation, **btldinstall** unwinds changes to the Link Kit to leave it in a working state.

Value added

/etc/btldinstall is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

captoinfo

convert a termcap description into a terminfo description

Syntax

captoinfo [-v ...] [-V] [-1] [-w *width*] *file* ...

Description

The **captoinfo** command looks in ***file*** for **TERMCAP** descriptions. For each one found, an equivalent *terminfo*(F) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the **TERMCAP** "tc=" field) will be reduced to the minimum superset before being output.

If no ***file*** is given, then the environment variable **TERMCAP** is used for the file name or entry. If **TERMCAP** is a full pathname to a file, only the terminal whose name is specified in the environment variable **TERM** is extracted from that file. If the environment variable **TERMCAP** is not set, then the file */etc/termcap* is read.

-v print out tracing information on standard error as the program runs. Specifying additional **-v** options will cause more detailed information to be printed.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out, one to a line. Otherwise, the fields will be printed several to a line, up to a maximum width of 60 characters.

-w change the output to ***width*** characters.

File

*/usr/lib/terminfo/?/** compiled terminal description database

Notes

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo* **bel**) is assumed to be ^G. The linefeed capability (*termcap* **nl**) is assumed to be the same for both **cursor_down** and **scroll_forward** (*terminfo* **cud1** and **ind**, respectively). Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for *termcap* fields such as **cursor_position** (*termcap* **cm**, *terminfo* **cup**) will sometimes produce a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation **%n** will produce strings that are especially long. Most occurrences of these non-optimal strings will be flagged with a warning message and may need to be recoded by hand.

The short two-letter name at the beginning of the list of names in a *termcap* entry, present for backwards compatibility, has been removed.

Diagnostics

`tgetent failed with return code` ***n*** `(`***reason***`).`
The *termcap* entry is not valid. In particular, check for an invalid "tc=" entry.

`unknown type given for the termcap code` ***cc***`.`
The *termcap* description had an entry for *cc* whose type was not Boolean, numeric, or string.

`wrong type given for the Boolean (numeric, string) termcap code` ***cc***`.`
The Boolean *termcap* entry *cc* was entered as a numeric or string capability.

`the Boolean (numeric, string) termcap code` ***cc*** `is not a valid name.`
An unknown *termcap* code was specified.

`tgetent failed on TERM=`***term***`.`
The terminal type specified could not be found in the *termcap* file.

`TERM=`***term***`: cap` ***cc*** `(info` ***ii***`) is NULL:REMOVED.`
The *termcap* code was specified as a null string. The correct way to cancel an entry is with an "@", as in ":bs@:". Giving a null string could cause incorrect assumptions to be made by the software which uses *termcap* or *terminfo*.

`a function key for` ***cc*** `was specified, but it already has the value` ***vv***`.`
When parsing the "ko" capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

`the unknown termcap name` ***cc*** `was specified in the ko termcap capability.`
A key was specified in the "ko" capability which could not be handled.

`the vi character` ***v*** `(info` ***ii***`) has the value` ***xx***`, but ma gives` ***n***`.`
The "ma" capability specified a function key with a value different from that specified in another setting of the same key.

`the unknown vi key` ***v*** `was specified in the ma termcap capability.`
A vi(C) key unknown to **captoinfo** was specified in the "ma" capability.

`Warning: termcap sg (`***nn***`) and termcap ug (`***nn***`) had different values.`
terminfo assumes that the "sg" (now "xmc") and "ug" values were the same.

`Warning: the string produced for` ***ii*** `may be inefficient.`
The parameterized string being created should be rewritten by hand.

`Null termname given.`
: The terminal type was null. This is given if the environment variable **TERM** is not set or is null.

`cannot open` ***file*** `for reading.`
: The specified file could not be opened.

See also

curses(S), **infocmp**(ADM), **terminfo**(F), **tic**(C)

chg_audit

enable and disable auditing for the next session

Syntax

/tcb/lib/chg_audit [on]

Description

chg_audit enables and disables auditing for the next session (next reboot). It edits the */etc/inittab* and */etc/conf/cf.d/init.base* files to add or remove the audit startup command when the system is rebooted. The command is normally invoked by the **auditsh**(ADM).

If **on** is specified, then auditing is enabled. If no argument is given, then the audit lines are removed from the *inittab* files.

Files

/etc/inittab
/etc/conf/cf.d/init.base

See also

auditsh(ADM), **sysadmsh**(ADM)

Value added

chg_audit is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

checkaddr

MMDF address verification program

Syntax

/usr/mmdf/bin/checkaddr [-w] [*addresses...*]

Description

The **checkaddr** program is used to check the validity of an address within the local mail system (MMDF). **checkaddr** can be given addresses either on the command line, one address per argument, or a list of addresses can be given to **checkaddr** on the standard input, one address per line. The latter mode is used for checking the addresses in a mailing list as in **checkaddr** < ***mailing-list-file***. **checkaddr** announces each address on a separate line and follows the address with its status (normally "OK"). **checkaddr** uses **submit**(ADM) to do the address verification.

If the **-w** option is given, **checkaddr** causes **submit** to generate a detailed submission tracing. This can sometimes be useful to help find problems in alias files or mailing lists.

See also

submit(ADM)

Credit

MMDF was developed at the University of Delaware and is used with permission.

checkque

MMDF queue status report generator

Syntax

/usr/mmdf/bin/checkque [-fpsz] [-*tage* [m]] [-c *channel channel* ...]

Description

checkque reports on the amount of mail waiting in the MMDF distribution queue. It indicates the total number of messages and the size of the queue directory. It then lists the number of messages waiting for each transmission channel.

The **-c** option allows one or more channel names to be specified. If present, **checkque** restricts its report to the named channels.

The **-f** option causes **checkque** to print the name of the oldest queued message for each channel. **-p** causes only channels with "problems" to be listed. Problems are defined as channels with mail waiting for over some "problem threshold". The default problem threshold is 24 hours. The **-t** option is used to change the problem threshold. A number of hours (or minutes, if **m** is appended) should appear without a space after the **-t**. **-s** forces an abbreviated summary listing instead of the normal multi-line report. **-z** causes channels with no messages queued to be skipped in the report.

Because the mail queue usually is protected from access by any uid, except MMDF, **checkque** should be run under *root* or *mmdf* uid. It should not be made **setuid**() to *mmdf* unless you want to allow non-staff members to see the queue status.

Most configurations will have only two channels. One is for local delivery and the second is for off-machine relaying, such as by calling out or by being called up, or by attaching to ArpaNet hosts. Local delivery usually happens at the time of submission, so it is rare that any mail is waiting in it. Mail in other outbound queues is processed by **deliver** according to your site parameters, either by running **deliver** as a background daemon or by periodically firing it up via **cron**.

Files

quedfldir[]/addr
quedfldir[]/msg
quedfldir[]/q.*
phase-directory/channel/*

See also

deliver(ADM)

Credit

This utility was written by Dave Crocker, Dept. of E.E., Univ. of Delaware.

MMDF was developed at the University of Delaware and is used with permission.

checkup

report on MMDF problems

Syntax

/usr/mmdf/bin/checkup [**-p -v** [*digit*]]

Description

The **checkup** command is used to check aspects of the MMDF system configuration. Normally, **checkup** reports on all problems that are encountered, including correct states. Displayed problems are prefixed by two asterisks (**); information that is advisory is enclosed in square brackets ([ ]).

The two optional flags to **checkup** specify how much information is displayed. The **-p** option reports only problems detected by **checkup**. This is useful for day-to-day checking of the system, such as mailing the output to the postmaster alias.

The **-v** flag takes an optional *digit* which ranges from 1 (the same as the **-p** option), to level 7 which displays all information.

Some of the displayed information, such as that about permissions modes varies by site conventions and may not have widespread significance. In particular it is common for sites to allow group read, write, or execute on files that **checkup** expects to be protected more carefully. Use of group permissions can greatly ease administration efforts for system administrators without compromising security. Warnings regarding "others" permissions should be examined.

Credit

MMDF was developed at the University of Delaware and is used with permission.

chroot

change root directory for command

Syntax

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*. In addition, the initial working directory is *newroot*.

Notice that:

chroot *newroot command* > *x*

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

This command is restricted to the super user.

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. Note that it is not possible to change directories to what was formerly the parent of the new root directory; that is, the **chroot** command supports the new root as an absolute root for the duration of the *command*. This means that "/.." is always equivalent to "/".

See also

chdir(S), **cd**(C)

Notes

Exercise extreme caution when referencing special files in the new root file system.

command must be under *newroot* or "*command*: `not found`" is reported.

Standards conformance

chroot is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

cleanque

send warnings and return expired mail

Syntax

/usr/mmdf/bin/cleanque [-w]

Description

cleanque removes extraneous files from the *tmp* and *msg* subdirectories of the MMDF "home queue" directory. It also sends warnings for mail which has not been fully delivered after "warntime" hours following submission. Finally, it returns mail which has not been fully delivered after "failtime" hours after submission. "Warntime" and "failtime" are defined in the MMDF *mmdftailor*(F) file.

Generally, **cleanque** should be run by **cron**, once a day, but may be run at any time to free up space.

The optional argument, **-w**, can be used if you are running **cleanque** manually and want to see what the program is doing.

See also

deliver(ADM), **queue**(F)

Notes

cleanque does not currently remove extraneous files from the individual queues (*q.** subdirectories).

Credit

MMDF was developed at the University of Delaware and is used with permission.

cleantmp

remove temporary files in directories specified

Syntax

/usr/lib/cleantmp

Description

cleantmp removes temporary files in directories specified in */etc/default/cleantmp* under the variable **TMPDIRS**. By default, */tmp* and */usr/tmp* are examined. Users can add to the list of directories, separating each directory with a space. Files in these directories which are not accessed within the last ***n*** days will be removed, where ***n*** is the number of days specified under the variable **FILEAGING** in */etc/default/cleantmp*. By default, **FILEAGING** is 7. Users can change the number of days for **FILEAGING**. */usr/lib/cleantmp* is run as a **cron** job every day at 3:00a.m. Refer to */usr/spool/cron/crontabs/root* on the system. The super user can edit this file to change the frequency and time at which **/usr/lib/cleantmp** is run. If the directories specified do not exist or if they are mount points and the file system is not mounted, **cleantmp** will send mail to root saying that the directory does not exist.

The format of */etc/default/cleantmp* is as follows:

FILEAGING=7
TMPDIRS=/tmp/usr/tmp

File

/etc/default/cleantmp

See also

rc2(ADM)

Value added

cleantmp is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

clri

clear inode

Syntax

/etc/clri *filesystem i-number ...*

Description

clri writes zeros on the 64 bytes occupied by the inode numbered ***i-number***. ***Filesystem*** must be a special filename referring to a device containing a filesystem. After **clri** is executed, any blocks in the affected file will show up as "missing" if the filesystem is checked with **fsck**(ADM). Use **clri** only in emergencies and exercise extreme care.

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

The primary purpose of this command is to remove a file which, for some reason, does not appear in a directory. If you use **clri** to destroy an inode which does appear in a directory, track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to this file. At that point, removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated.

See also

fsck(ADM), **ncheck**(ADM)

Notes

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

This utility does not work on DOS filesystems.

cnvtmbox

convert XENIX-style mailboxes to MMDF format

Syntax

/usr/mmdf/bin/cnvtmbox [**-c** | **-o**] ***old_mailbox*** [***new_mailbox***]

Description

cnvtmbox converts a mailbox (***old_mailbox***) either from the XENIX-style (the older UNIX-style) format to MMDF format or from MMDF format to XENIX format. Generally, mailboxes in MMDF format use ⟨Ctrl⟩**A** to delimit messages; XENIX format uses lines beginning with "From <Space>" to delimit between messages. (You can change the message-delimiter character using the **MMBXPREF** and **MMBXSUFF** keywords in the */usr/mmdf/mmdftailor* file. For more information, see the **mmdftailor**(F) manual page.)

If ***new_mailbox*** is specified, **cnvtmbox** places the converted mailbox in this folder: otherwise, this utility writes the converted mailbox to *stdout*.

The options to **cnvtmbox** are:

c Converts XENIX-style or mixed-format mailbox to MMDF (generally ⟨Ctrl⟩**a**-delimited) format. If no options are specified, **-c** is the default.

o Converts MMDF or mixed-format mailbox to XENIX-style (or old UNIX-style) format.

File

/usr/mmdf/bin/cnvtmbox

See also

"Setting up electronic mail" in the *System Administrator's Guide*

Value added

cnvtmbox is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

configure

kernel configuration program

Syntax

cd /etc/conf/cf.d
configure [*options*] [*resource=value* ...]

Description

The **configure** program determines and alters different kernel resources. For end users, using **configure** is easier than modifying the system configuration files directly. For device driver writers, **configure** avoids the difficulties of editing configuration files that have already been edited by an earlier driver configuration script.

You must move to */etc/conf/cf.d* to execute **configure**.

Resources are modified interactively or with command-line arguments. Adding or deleting device driver components requires the command-line options.

The next paragraphs discuss how to use **configure** interactively. Command-line options are discussed in the "Options" section.

Before using **configure** to modify the system configuration files, use the following command to make a backup copy of the kernel:

cp /unix /unix.old

Interactive Usage

configure operates interactively when no options (including ***resource=value***) are given or when **-f** is the only option specified on the command line.

When you invoke **configure** interactively, you first see a category menu similar to the following:

```
1.  Disk and Buffers
2.  Character Buffers
3.  Files, Inodes, and Filesystems
4.  Processes, Memory Management and Swapping
5.  Clock
6.  MultiScreens
7.  Message Queues
8.  Semaphores
9.  Shared Data
10. System Name
11. Streams Data
12. Event Queues and Devices
13. Hardware Dependent Parameters
14. Remote File sharing Parameters
15. Security Parameters

Select a parameter category to reconfigure
by typing a number from 1 to 15, or type 'q' to quit:
```

To choose a category, enter its number (for example, "1" for "Disk and Buffers"), then press ⟨Return⟩.

Each category contains a number of configurable resources. Each resource is presented by displaying its name, a short description, and its current value. For example, for the "Disk and Buffers" category you might see:

```
NBUF: total disk buffers.
Currently determined at system start up:
NSABUF: system-addressable (near) disk buffers.
Currently 10:
NHBUF: hash buffers (for disk block sorting).
Currently 128:
```

To keep the current value, simply press ⟨Return⟩. Otherwise, enter an appropriate value for the resource, then press ⟨Return⟩. **configure** checks each value to make sure that it is within an appropriate range. If it is not, **configure** warns you that the value is inappropriate and asks you to confirm that you want to override the recommended value.

To exit from **configure**, enter **q** at the category menu prompt. If any changes are made, **configure** asks if it should update the configuration files with the changes. To keep the old configuration values, enter **n** at this prompt, and no changes are made. Otherwise, enter **y** and **configure** updates the required configuration files. After **configure** has completed, the kernel is ready for linking.

To link the kernel, enter:

cd /etc/conf/cf.d
./link_unix

Linking may take a few minutes. After the kernel is linked, enter the following command to reboot the system to run the new kernel:

/etc/shutdown

Follow the prompts for shutting the system off. Next, you see the boot prompt:

```
Boot
:
```

Press ⟨Return⟩. The system is now running the new kernel.

Options

The command line options are designed for writers of driver-installation shell scripts. You can configure drivers, query driver configurations, remove driver definitions from the configuration files, and modify certain driver attributes, plus query and alter kernel parameters, all from the command line. There are also options for querying the current driver configuration.

configure uses the following options:

-a	[*func1 func2* ...]
-b	
-c	
-d	[*func1 func2* ...]
-f	*master_file* [*dfile*]
-g	*dev_name handler* | *dev_name*
-h	*dev_name*
-i	
-j	[*prefix*] [**NEXTMAJOR**]
-l	*priority_level*
-m	*major_dev_number*
-o	
-p	
-q	
-s	
-t	
-v	*interrupt_vector* [*interrupt_vector2*...]
-w	
-x	
-y	*resource*
-A	*address address*
-C	*channel*
-D	
-G	
-H	
-I	*address address*
-J	*address address*

-M *maximum minimum*
-O
-P
-R
-S
-T *interrupt_scheme*
-U *number_of_subdevices*
-V *interrupt_vector*
-X *offset*
-Y
-Z

-m, **-b**, and **-c**
These options are used to define which driver is being referenced. Following **-m** must be the major device number of the driver. If you are configuring a block driver, **-b** must appear; if you are configuring a character driver, **-c** must appear. Both are used when configuring a driver with both kinds of interfaces.

-s When adding or deleting a streams module, use this option with the **-h** option and instead of **-m**, **-b**, and **-c**. For a streams driver, use it with **-m** and **-c**.

-a and **-d**
Each option is followed by a list of functions to add or delete, respectively. These are the names of the functions that appear within **bdevsw** or **cdevsw**, as appropriate, plus the names of the initialization, clock poll, halt, and interrupt routines, if present, plus the name of the tty structure pointer. **configure** enforces the rules that all of a driver's routines must have a common prefix, and that the prefix must be 2-4 characters long.

-h This option is used to give the driver or streams module name when the name is different from the prefix or when no prefix is specified as in the case of the streams module. The name can be 1-8 characters long.

-i This option with **-a** or **-d** adds or deletes the characteristic that the device is a tty. The default is off.

-j When followed by a *prefix* used by a driver, the major device number is displayed. When followed by the string **NEXTMAJOR**, the smallest unused major device number is displayed.

-v This option modifies the system notion of the vectors on which this device can interrupt.

-l This sets the interrupt priority level of the device, which is almost always the same as the type of **spl** call used: a driver that interlocks using **spl5** almost always has an interrupt priority level of 5. Use of this option should not be required in new drivers.

-f Much of the configuration data is maintained in two files, whose default names are *mdevice* and *mtune*. The **-f** option can be used to specify alternate names. Note that if **-f** is the only option present, the program is still interactive.

-w When specifying a parameter value, this option works in the same way as the **-o** option, but suppresses all warning messages, when a parameter is set outside the current maximum and minimum values.

-o This is the override flag. When invoked non-interactively, this option overrides the minimum and maximum values that are otherwise enforced. This option has no effect on interactive commands.

-p This option with **-a** or **-d** adds or deletes the characteristic that the device is a SCSI peripheral. Default is off.

-q This option with **-a** or **-d** adds or deletes the characteristic that the device is a SCSI host adapter. Default is off.

-x This dumps all the resource prompts known to **configure**. These reveal the name, description, and current value of each parameter capable of being reconfigured. Category prompts are not dumped.

-y The **-y** option displays the current value of the requested parameter.

-t This option displays nothing (except possibly error messages). However, it has a return value of "1" if a driver corresponding to the given combination of **-m**, **-b**, **-c** options is already configured, and "0" if no such driver is present.

-g This option is used to add or remove graphics input (GIN) device handlers. Devices such as mice, bitpads, and keyboards may have handlers to turn their input data into "events". The **-g** flag may be given one argument that is interpreted as a device name. That GIN device is removed from the configuration files. If the **-g** flag has two arguments, the second is a handler for that device, and the device is added to the files. If it was already present, its handler is updated and the user is informed. Multiple devices may be added or removed by specifying **-g** multiple times.

-A This option, followed by two values that are taken to be hexadecimal I/O addresses, returns the name of the device with the I/O address conflict.

-C Followed by an integer, this option used with **-a** indicates the DMA channel that the device uses. The default is not to use DMA.

-D This option used with the **-a** option adds to the device driver the characteristic that the driver can share its DMA channel; **-D** used with the **-d** option deletes this characteristic. The default is not to share.

-G This option with **-a** adds the "G" characteristic to the driver; **-G** with **-d** deletes the "G" characteristic. This characteristic indicates whether or not the device uses an interrupt, even though an interrupt is specified in the *sdevice* file. This is used when you want to associate a device to a specific device group. The default is not to set this characteristic.

-H This option with **-a** or **-d** adds or deletes the characteristic that the driver supports hardware that distinguishes it from those that are entirely software (pseudo devices). The default is to set this characteristic.

-I This option is followed by two values that are the hexadecimal start and end I/O addresses. The default values are zero.

-J The option is followed by two values that are the hexadecimal start and end controller memory addresses. The default values are zero.

-M This option followed by two integers states the maximum and minimum number of devices that can be specified in the *sdevice* file. The default is a maximum of 1 and a minimum of 0.

-O This option with **-a** or**-d** indicates whether or not the IOA range of the device can overlap that of another device. The default is no.

-P When used with **-a** or **-d**, adds or deletes an ignore "I" flag in the device *mdevice* entry. The "I" flag allows the configuration build utilities to ignore a devices *pack.d* directory (useful to the mpt/spt) driver.

-R This option with **-a** or**-d** indicates whether or not the driver is required in the kernel all the time. The default is yes.

-S This option with **-a** or **-d** indicates whether or not the driver has one *sdevice* entry only. The default is no.

-T This option, when followed by an argument, states the type of interrupt scheme the device uses. The possible arguments are:

0 The device does not require an interrupt line.

1 The device requires an interrupt line. If the device supports more than one controller, each controller requires a separate interrupt.

2 The device requires an interrupt line. If the device supports more than one controller, the controllers share the same interrupt.

3 The device requires an interrupt line. If the device supports more than one controller, the controllers share the same interrupt. Multiple device drivers having the same interrupt priority level can share this interrupt.

The default is 0.

-U This option, when followed by an integer, encodes a device-dependent numeric value in the *sdevice* file to indicate the number of subdevices on a controller or a pseudo device. The integer must be a value that lies within the maximum and minimum number of devices specified in the *mdevice* file. The default is 1.

-V This option, followed by a vector value, returns the name of the device with the vector conflict.

-Y This option with **-a** or **-d** indicates whether or not to configure a driver into the kernel. Specifying **-a** puts a "Y" in the configuration field of the driver's *sdevice* entry; specifying **-d** puts an "N" in this field. The default is to put a "Y".

-X *offset*
This option is used to add (**-a**) or delete (**-d**) an extended minor device number entry from *mdevice*. The extended minor *offset* must be a multiple of 256. This option must be used in conjunction with **-m**, **-b**, or **-c**.

-Z This option indicates that a device can have more than one entry in the *mdevice* file. The SCSI driver is an example of a driver that needs this feature. The option is usually used when adding a new entry or deleting a particular entry in the *mdevice* file. When an additional line is added, this option should also be run on the original entry to set this characteristic in that entry. Using **-d** with **-Z** removes only the *mdevice* entry. Using **-d** without **-Z** removes the *mdevice* entry and the *sdevice* entry.

Setting Command-Line Parameters

Any number of arguments can be given on the command line of the form ***resource=value***. These arguments can be given at the same time as an add or delete driver request, but must follow all the driver-configuration arguments on the command line.

If one or more instances of ***resource=value*** are the only arguments on the command line, the changes are made non-interactively. If the values given are outside the permissible range for a parameter, no action is taken unless the **-o** or **-w** options are included to override them.

Some resources have values that are character strings. In this case, their values must be enclosed within the characters \" . The quotes are syntactically necessary for them to be used as C-language strings, and the backslashes protect the quotes from being removed by the shell.

Examples

(Note: these examples are provided for illustrative purposes only. Do not attempt to follow them; the device numbers and vectors specified may already be in use on your system. If this is the case, an attempt to carry out these examples will result in unpredictable errors.)

Print out the current value of **NCLIST**:

configure -y NCLIST

Return 1 if character major device 7 and vector 3 are already configured:

configure -t -v 3 -m 7 -c

Add a clock-time polling and initialization routine to the already configured "foo" driver, a hypothetical character driver at major device #99:

configure -a foopoll fooinit -c -m 99

Delete the hypothetical "foo" driver:

configure -m 99 -d -c

Add a new "hypo" driver, a block driver with a character interface. It absorbs 3 different interrupt vectors, at priority 6:

```
configure -a hypoopen hypoclose hyporead \
 hypowrite hypoioctl  hypostrategy  hypoprint \
 hypointr -b -c -l 6 -v 17 42 49 -m 99
```

Add a new streams module with prefix "grb" and name "garble":

configure -s -a grbinit -h garble

Files

/etc/conf/cf.d/mdevice
/etc/conf/cf.d/sdevice
/etc/conf/cf.d/mtune
/etc/conf/cf.d/stune
/etc/conf/cf.d/mevent
/etc/conf/cf.d/sevent

See also

link_unix(ADM), **majorsinuse**(ADM), **mdevice**(F), **mtune**(F), **sdevice**(F), **stune**(F), **vectorsinuse**(ADM),

"Tuning System Performance" in the *System Administrator's Guide*

Value added

configure is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

consoleprint

print /usr/adm/messages or any file to a serial printer attached to the printer port of a serial console

Syntax

consoleprint [*file*]

Description

consoleprint prints the file */usr/adm/messages* to a printer attached to the printer port of a serial console. If a filename is specified, it is printed instead. **consoleprint** is normally run by a system administrator to get a hardcopy version of the system console messages.

File

/etc/termcap

See also

lprint(C)

Notes

The only terminals currently supported with entries in */etc/termcap* are the Tandy DT-100 and DT-1, and the Hewlett-Packard HP-92.

Terminal communications parameters (such as baud rate and parity) must be set up on the terminal by the user.

Value added

consoleprint is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

crash

examine system images

Syntax

/etc/crash [**-d***dumpfile*] [**-n***namelist*] [**-w***outputfile*]

Description

The **crash** command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to **crash** are ***dumpfile***, ***namelist***, and ***outputfile***.

dumpfile is the file containing the system memory image. The default ***dumpfile*** is */dev/mem*.

The text file ***namelist*** contains the symbol table information needed for symbolic access to the system memory image to be examined. The default ***namelist*** is */unix*. If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the **crash** command is invoked, a session is initiated. The output from a **crash** session is directed to ***outputfile***. The default ***outputfile*** is the standard output.

Input during a **crash** session is of the form:

function [***argument*** ...]

where ***function*** is one of the **crash** functions described in the **Functions** section of this manual page, and ***arguments*** are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

The following function options are available to **crash** functions wherever they are semantically valid.

-e Display every entry in a table.

-f Display the full structure.

-p Interpret all address arguments in the command line as physical addresses.

-s ***process*** Specify a process slot other than the default.

-w ***file*** Redirect the output of a function to ***file***.

Note that if the **-p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions **mode**, **defproc**, and **redirect** correspond to the function options **-p**, **-s**, and **-w**. The **mode** function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; **defproc** sets the value of the process slot argument for subsequent functions; and **redirect** redirects all subsequent output.

Output from **crash** functions may be piped to another program in the following way:

function [***argument ...***] ***! shell_command***

For example:

mount ! grep rw

will write all mount table entries with an **rw** flag to the standard output. The redirection option (**-w**) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table slot arguments are always decimal. Table slot arguments larger than the size of the function table will not be interpreted correctly. Use the **findslot** command to translate from an address to a table slot number. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by "0x" and as octal if it is preceded by "0". Decimal override is designated by "0d", and binary by "0b".

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as **p** for **proc**, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number or a range. A range of slot numbers may be specified in the form ***a-b*** where ***a*** and ***b*** are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be any of the following symbols:

+ - * / & |

An operand which is a number should be preceded by a radix prefix if it is not a decimal number ("0" for octal, "0x" for hexadecimal, "0b" for binary). The expression must be enclosed in parentheses ( ). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to **crash** functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

table_entry = ***table entry*** | ***range***

start_addr = ***address*** | ***symbol*** | ***expression***

Functions

? [**-w** *file*] List available functions.

!cmd Escape to the shell to execute a command.

adv [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the advertise table. (RFS specific)

base [**-w** *file*] number ...
Print **number** in binary, octal, decimal, and hexadecimal. A number in a radix other than decimal should be preceded by a prefix that indicates its radix as follows: "0x", hexadecimal; "0", octal; and "0b", binary.

buffer [**-w** *file*] [*-format*] *bufferslot*
or
buffer [**-w** *file*] [*-format*] [**-p**] *start_addr*
Alias: **b**.
Print the contents of a buffer in the designated format. The following format designations are recognized: **-b**, byte: **-c**, character; **-d**, decimal; **-x**, hexadecimal; **-o**, octal; **-r**, directory; and **-i**, inode. If no format is given, the previous format is used. The default format at the beginning of a **crash** session is hexadecimal.

bufhdr [**-f**] [**-w** *file*] [[**-p**] *table_entry* ...]
Alias: **buf**.
Print system buffer headers.

callout [**-w** *file*]
Alias: **c**.
Print the callout table.

dballoc [**-w** *file*] [*class* ...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed. (Streams specific)

dbfree [**-w** *file*] [*class* ...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed. (Streams specific)

dblock [**-e**] [**-w** *file*] [**-c** *class* ...]
or
dblock [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print allocated streams data block headers. If the class option (**-c**) is used, only data block headers for the class specified will be printed. (Streams specific)

defproc [**-w** *file*] [**-c**]
or
defproc [**-w** *file*] [*slot*]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**-c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a **crash** session, the process slot is set to the current process.

dis [**-w** *file*] [**-a**] *start_addr* [*count*]
Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**-a**) specifies a non-symbolic disassembly.

ds [**-w** *file*] *virtual_address* ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Alias: **f**.
Print the file table.

findaddr [**-w** *file*]*tableslot*
Print the address of slot in table. Only tables available to the **size** function are available to **findaddr**.

findslot [**-w** *file*] *virtual_address* ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the **size** function are available to **findslot**.

fs [**-w** *file*] [[**-p**] *table_entry* ...]
Print the file system information table.

gdp [**-e**] [**-f**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the gift descriptor protocol table. (RFS specific)

gdt [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the global descriptor table.

help [**-w** *file*] *function* ...
Print a description of the named function, including syntax and aliases.

idt [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the interrupt descriptor table.

inode [**-e**] [**-f**] [**-w** *file*] [[**-p**] *table_entry* ...]
Alias: **i**.
Print the inode table, including file system switch information.

kfp [**-w** *file*] [*value*]
Print the frame pointer for the start of a kernel stack trace. If the value argument is supplied, the **kfp** is set to that value.

lck [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Alias: **l**.
Print record-locking information. If the **-e** option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

ldt [**-e**] [**-w** *file*] [**-s** *process*] [[**-p**] *table_entry* ...]
Print the local descriptor table for the given process, or for the current process if none is given.

linkblk [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the linkblk table. (Streams specific)

map [**-w** *file*] *mapname* ...
Print the map structure of *mapname*.

mbfree [**-w** *file*]
Print free streams message block headers. (Streams specific)

mblock [**-e**] [**-w** *filename*] [[**-p**] *table_entry* ...]
Print allocated streams message block headers. (Streams specific)

mode [**-w** *file*] [*mode*]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a **crash** session, the mode is virtual.

mount [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Alias: **m**.
Print the mount table.

nm [**-w** *file*] *symbol* ...
Print value and type for the given symbol.

od [-p] [-w *file*] [*-format*] [*-mode*] [-s *process*] *start_addr* [*count*]
Alias: **rd**.
Print **count** values starting at the start address in one of the following formats: character (**-c**), decimal (**-d**), hexadecimal (**-x**), octal (**-o**), ASCII (**-a**), or hexadecimal/character (**-h**), and one of the following modes: long (**-l**), short (**-t**), or byte (**-b**). The default mode for character and ASCII formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format **-h** prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a **crash** session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

Example:

```
> od 0 4
00000000:  0008014c   256471ed   00000000   00000000
> vtop 0
VIRTUAL  PHYSICAL SECT SDT   SRAM   PDT
       0   1eb000    0   0        0   0
> od -p 1eb000 4
001eb000:  0008014c   256471ed   00000000   00000000
> mode p
Mode = physical
> od 1eb000 4
001eb000:  0008014c   256471ed   00000000   00000000
>
```

panic Print the latest system notices, warnings, and panic messages from the limited circular buffer kept in memory.

pcb [-w *file*] [*process*]
Print the process control block (TSS) for the given process. If no arguments are given, the active TSS for the current process is printed.

pdt [-e] [-w *file*] [-s *process*] [-p] *start_addr* [*count*]
The page descriptor table of the designated memory **section** and **segment** is printed. Alternatively, the page descriptor table starting at the start address for **count** entries is printed. If no count is entered, 1 is assumed.

pfdat [-e] [-w *file*] [[-p] *table_entry* ...]
Print the pfdata table.

proc [-e] [-f] [-w *file*] [[-p] *table_entry* ... *#procid* ...]
or
proc [-f] [-w *file*] [-r]
Alias: **p**.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and

process ids may be entered. Each process id must be preceded by a "#". Alternatively, process table information for executable processes may be specified with the executable option (**-r**). The full option (**-f**) details most of the information in the process table as well as the region table for that process.

qrun [**-w** *file*] Print the list of scheduled streams queues. (Streams specific)

queue [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print streams queues. (Streams specific)

quit Alias: **q**.
Terminate the **crash** session.

rcvd [**-e**] [**-f**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the receive descriptor table. (RFS specific)

redirect [**-w** *file*] [**-c**]
or
redirect [**-w** *file*] [*file*]
Used with a filename, redirects output of a **crash** session to the named file. If no argument is given, the filename to which output is being redirected is printed. Alternatively, the close option (**-c**) closes the previously set file and redirects output to the standard output.

region [**-e**] [**-w** *file*] [[**-p**] *table_entry* ...]
Print the region table.

sdt [**-e**] [**-w** *file*] [**-s** *process*] *section*
or
sdt [**-e**] [**-w** *file*] [**-s** *process*] [**-p**] *start_addr* [*count*]
The segment descriptor table for the current process is printed.

search [**-p**] [**-w** *file*] [**-m** *mask*] [**-s** *process*] *pattern start_addrcount*
Print the long words in memory that match ***pattern***, beginning at the start address for ***count*** long words. The mask is "anded" (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

Example:

```
> od 0 4
00000000:  0008014c   256471ed   00000000   00000000
> se -m ff ed 0 4
MASK = 0xff, PATTERN = 0xed, START = 0x0, LENGTH = 0x4

MATCH AT        4: 256471ed
>
```

size [**-w** *file*] [**-x**] [***structure_name*** ...]
Print the size of the designated structure. The (**-x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**-e**] [**-f**] [**-w** *file*] [[**-p**] ***table_entry*** ...]
Print the send descriptor table. (RFS specific)

srmount [**-e**] [**-w** *file*] [[**-p**] ***table_entry*** ...]
Print the server mount table. (RFS specific)

stack [**-w** *file*] [***process***]
Alias: **s**.
Dump stack. If no arguments are entered, the kernel stack for the current process is printed. Neither the u-area stack associated with the current process or the process's own stack are accessible on a running system.

stat [**-w** *file*] Print system statistics.

stream [**-e**] [**-f**] [**-w** *file*] [[**-p**] ***table_entry*** ...]
Print the streams table. (Streams specific)

strstat [**-w** *file*] Print streams statistics. (Streams specific)

trace [**-w** *file*] [**-r**] [***process***]
Alias: **t**.
Print kernel stack trace. The **kfp** value is used with the **-r** option.

ts [**-w** *file*] ***virtual_address*** ...
Print closest text symbol to the designated address.

tty [**-e**] [**-f**] [**-w** *file*] [**-t** ***type*** [[**-p**] ***table_entry*** ...]]
Valid types: **cn**, **sio** (console, serial ports).
Print the tty table. If no arguments are given, the tty table for the console is printed. If the **-t** option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [**-f**] [**-w** *file*] [***process***]
Alias: **u**.
Print the ublock for the designated process.

var [**-w** *file*] Alias: **v**.
Print the tunable system parameters.

vtop [**-w** *file*] [**-s** ***process***] ***start_addr*** ...
Print the physical address translation of the virtual start address.

Files

/dev/mem	system image of currently running system
/unix	namelist for currently running system

custom

install software products and components

Syntax

custom [**-od**] [**-abilrv** [*package*]] [**-s** *set*] [**-m** *device*] [**-f** [*file*]]

Description

With **custom** you can create a custom installation by selectively installing or deleting portions of the UNIX system or other products. **custom** is executable only by the super user and is either interactive or can be invoked from the command line with several options.

custom has three levels of operation: Complete Product, Service, and Service Component. At the Complete Product level, the entire product distribution (also known as a "bundle") is installed, which can consist of several products. At the Service level, groups of products that comprise a functional area are installed. At the Service Component level, an individual product, its packages, or individual files are installed. For example, a Complete Product could consist of several Services. In turn, a Service Component would include a number of packages. Files are extracted or deleted in *packages*. A package is a collection of individual files.

You can also install additional products. You can list the available packages by using the **custom** command as described next.

Usage

To use **custom** interactively, enter:

custom

The **custom** main menu appears with the following options:

Install — Allows a product or system to be added.

A window is first opened to select a "New Product" or a system set. When a new product is selected, you are given the choice of adding the "Entire Product", "Packages" or "Files". When "Entire Product" is chosen, **custom** calculates which installation volumes (distribution media) are needed, then prompts for the correct volume numbers.

If "Packages" is chosen, a list of all available packages in the currently selected set is displayed. Each line describes the package name, whether the package is fully installed, not installed or partially installed, the size of the package (in 512 byte blocks), and a one line description of the package contents.

Multiple packages can be specified by marking them with the space bar. The selected packages will appear with asterisks. When executed, **custom** will prompt for insertion of the necessary volumes. (You cannot use **custom** to install the entire RTS package if that package is already partially installed. If this situation comes up, use **fixperm**(ADM) to determine which files are missing, and then use **custom** to install each file individually.)

If "Files" is chosen, you are prompted to select the package and then the filenames. **custom** then prompts for volumes.

If a system set is selected, **custom** operates at the product level. You are given the option of installing the "Complete Product" (complete distribution), "Services" (a specific group of products), or "Service Components" (individual products).

Remove — Deletes the correct files in the specified package/product. Select the product or package to be deleted just as you select a product or package to install.

List — Lists all files in the specified package or all packages in a product set.

Quit — Leaves **custom**.

Options

Three arguments are required for a completely non-interactive use of **custom**:

A set identifier (**-o** or **-d**)

A command (**-i**, **-r**, **-l**,**-f**, **-a**, **-v**, **-b**, or **-s**)

And either one or more package names, or a filename

If any information is missing from the command line, **custom** prompts for the missing data.

Only one of **-o**, or **-d** may be specified. These stand for:

-o Operating System

-d Development System

Only one of **-a**, **-b**, **-f**, **-i**, **-l**, **-r**, **-s**, or **-v** may be specified, followed by an argument of the appropriate type (one or more package names, or a filename). These options perform the following:

-i Install the specified package(s)

-r Remove the specified package(s)

-v Verbose output for installing and removing packages. For example, it gives information on command being run and on the size of the packages.

-s *set* To install a specified ***set***. A ***set*** is a collection of packages listed in a permsfile entry for a product. For example, "ext" is the Extended Utilities set of the operating system.

-l List the files in the specified package(s)

-b Enforces product dependencies specified in the "bundle" file for "bundled" products.

-f Install the specified file

-a Add a new product

The **-m** flag allows the media device to be specified. The default is */dev/install* (which is always the 0 device, as in */dev/fd0*). This is very useful if the system has a 5.25-inch drive on */dev/fd0* and a 3.5-inch floppy on */dev/fd1*, and it is necessary to install 3.5-inch media. For example:

custom -m /dev/rfd196ds9

this will override the default device and use the one supplied with the **-m** flag.

File

*/etc/perms/**

See also

df(C), **du**(C), **fixperm**(ADM), **xinstall**(ADM)

Notes

If you upgrade any part of your system, **custom** detects if you have a different release and prompts you to insert the floppy volume that updates the custom data files. Likewise, if you insert an invalid product or a volume out of order, you will be prompted to reinsert the correct volume.

Upon installation of the operating system, the RTS package is always entirely installed.

Value added

custom is an extension to AT&T System V developed by The Santa Cruz Operation, Inc.

dbmbuild

build the MMDF hashed database of alias and routing information

Syntax

/usr/mmdf/table/dbmbuild [-nvdk] [*database* [*table* ...]]

Description

dbmbuild reads the tables specified in the MMDF tailor file into a hashed database for use in quickly verifying addresses and efficiently assigning channels to submitted messages. Whenever you change MMDF alias or routing information in any way, you must rebuild the hashed database by logging in as **mmdf** and running **dbmbuild** from the */usr/mmdf/table* directory.

If no database file is specified, the default database *mmdfdbm* is used. If no table files are specified, all tables listed in the tailor file are used. In particular, three tables are read for each channel definition: the list of authorized sources, the list of authorized destinations, and the table of names/aliases for that channel. Also, the remaining tables (MTBL and MDMN) are read.

The options are:

n Create a new database. If this option is omitted, **dbmbuild** updates an existing database. If no options at all are specified, **-n** is assumed; however, if you give any options (even **-v**), you must specify the **-n** option if you want to create a new database.

v Run in verbose mode, displaying information during table processing.

d Run in debug mode, reporting everything that happens.

k Keep going. If a file is mentioned that does not exist, ignore it. This option might be an appropriate default at some sites.

Appropriate locks are placed on the database so that **dbmbuild** can safely be run while MMDF is in operation.

Files

/usr/mmdf/mmdftailor	
/usr/mmdf/table/alias.list	
/usr/mmdf/table/alias.user	
/usr/mmdf/table/.chn*	
/usr/mmdf/table/.dom*	
$(tbldbm).dir	database directory
$(tbldbm).pag	database pages
$(tbldbm).lck	database locking file
*$(tbldfldir)/**	various tables that form the database

See also

dbm(S), **mmdftailor**(F), **tables**(F)

"Setting up electronic mail" in the *System Administrator's Guide*

Credit

MMDF was developed at the University of Delaware and is used with permission.

dbmedit

edit the MMDF database file

Syntax

/usr/bin/dbmedit [-v] [-d*database* **] [** ***cmd*...**]**

Description

The **dbmedit** command lets you edit the **dbm**(S) database used by MMDF. Use this command for quick and simple changes to the database or with careful use of **setuid** programs to make controlled changes on behalf of users. For example, a **forwardmail** command (that you create) might use **dbmedit** to change a user's entry in the *dbm* database after changing the mail forwarding alias file.

The **-v** option may be used to get a verbose description of the program's activities.

The **-d** option may be used to specify an alternate database. The default is given by the **tbldbm** configuration variable or by the MDBM **/usr/mmdf/mmdftailor** variable.

If no arguments are given to **dbmedit**, then the program goes into an interactive mode, and prompts the user for each command. Otherwise the arguments are taken as one command.

Commands in **dbmedit** refer to keys, tables, and values. Tables (see **tables**(F)) are hashed into the database using **dbmbuild**(ADM). (Tables that refer to domain name servers are not part of the database.) The keys appear on the left side of the tables and the values on the right side. In general, only the first occurence of a value for a given key/table pair is significant. For example, the table entries:

table1:
⟨key1⟩: ***val1***
⟨key2⟩: ***val2***

table2:
⟨key1⟩: ***val3***
⟨key1⟩: ***val4***

get hashed into the following database entries:

⟨key1⟩ *table1* ***val1***
⟨key1⟩ *table2* ***val3***
⟨key1⟩ *table2* ***val4***
⟨key2⟩ *table1* ***val2***

(In the current implementation, the database is keyed on only the key, and table/value pairs are encoded in the data portion. This is likely to change but will not affect this or any other program.)

The command lines in interactive mode are parsed using the standard MMDF string-to-argument routines so the same quoting and escape conventions are used. For example, if you want double-quotes or spaces in the value, they must be escaped with a backslash or the string must be quoted (for spaces).

The commands are:

print ⟨key⟩ [***table***]
: Print the value of the key/table pair. If the table is omitted, then print the value of any table entry with this key.

add ⟨key⟩ ***tablevalue***
: Add a key/table entry with the given value. In verbose mode, a warning message is printed if the given key/table pair already has a value in the database.

delete ⟨key⟩ [***table*** [***value***]]
: Delete the values for the specified key. If a table is specified, delete only the values for the specified key/table pair. If a value is also specified, delete only entries for the pair with that value. It is an error to try to delete something which does not appear in the database as specified.

change ⟨key⟩ ***table*** [***oldvalue***] ***newvalue***
: Change the value of the specified key/table pair to ***newvalue.*** If ***oldvalue*** is specified, change the entry matching that value. Otherwise, change the value of the first occurence or add a new key/table pair if none already exists.

help Give a brief summary of the commands

quit Exit the program.

All commands may be shortened to their first character only. If the wrong number of arguments is given to a command, a "Usage:" message is displayed. This program may be used while MMDF processes are running.

NOTE: All changes are made in real time; no temporary copy of the database is made while editing takes place.

File

$(tbldbm).{dir,pag} the MMDF database

See also

dbmbuild(ADM), **tables**(F)

Credit

This utility was written by Phil Cockcroft.

MMDF was developed at the University of Delaware and is used with permission.

dcopy

copy UNIX filesystems for optimal access time

Syntax

/etc/dcopy [**-s***X*] [**-a***n*] [**-d**] [**-v**] [**-f***fsize* [**:***isize*]] ***inputfs outputfs***

Description

The **dcopy** command copies filesystem ***inputfs*** to ***outputfs***. ***inputfs*** is the device file for the existing file system; ***outputfs*** is the device file to hold the reorganized result. This utility is for UNIX fileystems only. For the most effective optimization, ***inputfs*** should be the raw device and ***outputfs*** should be the block device. Both ***inputfs*** and ***outputfs*** should be unmounted file systems.

With no options, **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**(ADM).

-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 subdirectories to the beginning of directories).

-v currently 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, followed by an interrupt or quit.

dcopy also attempts to modify its command line arguments so its progress can be monitored with **ps**(C).

See also

fsck(ADM), **mkfs**(ADM), **ps**(C)

deliver

MMDF mail delivery process

Syntax

/usr/mmdf/bin/deliver [**-bdpsw**] [**-c***chan,chan*] [**-l***mins*] [**-t***hrs*]
[**-m***maxsort*] [**-L***logfile*] [**-T***secs*]
[**-V***loglevel*] [*message1 ... messageN*]

Description

The **deliver** program handles the management of all mail delivery under the MMDF mail system. **deliver** does not deliver mail directly, but instead calls on MMDF channels to handle actual delivery. **deliver's** actions are guided by the MMDF tailoring file, */usr/mmdf/mmdftailor*, and by the command line options. The program can run as either a daemon or a user-invoked program. The program may be called to process the entire mail queue or just handle some explicitly named messages. When possible, **deliver** will attempt to process messages in the order received. **deliver** also maintains a cache of host information on a per-channel basis which allows hosts which are unavailable for delivery to be skipped until available.

deliver first builds a list of channels to process, either from the command line or composed of all the non-passive channels in the system. Next, a list of messages to process is collected, either from the command line or by scanning the mail queue for each channel. If the the number of messages in the queue for a given channel is more than ***maxsort*** (set in the tailor file or on the command line), the queue directory for that channel will be processed in the order read, without sorting by submission time. If a list of messages is given on the command line, no sorting will take place and the messages will be delivered in the order specified. The sorting keys are (in order): **channel**, **submission time**, and finally **host**. This causes many accesses to the messages but minimizes the invocation of channel programs.

deliver is **setuid** to the super user to allow it to set its real and effective UID and GID to that of the MMDF user.

The following options may be used to alter **deliver's** behavior:

-b Background mode. Causes **deliver** to run as a background daemon making periodic sweeps over the mail queues looking for undelivered mail and attempting deliver. The invoker must be the MMDF user or the superuser to use this option. **deliver** attempts delivery for all eligible messages, then sleeps, and then repeats the process. The default sleep time is 10 minutes but it can be changed (see the **-T** option below).

-c*channel1,channel2,...*
Channel selection. A comma-separated list of channels to be processed.

-d Already in "*quedfldir*". This option will cause **deliver** to assume it is already in the mail queue and therefore it will not issue an explicit **chdir**. This is useful if you wish to have **deliver** operate on an alternate mail queue hierarchy, mainly for testing.

-l*minutes* Sets the "time-to-live" for entries in the dead-host cache. This time defaults to 2 hours. The dead host cache is used to prevent attempts to deliver to hosts that are known to be down. The "time-to-live" is given in minutes. If the number of minutes is negative, dead host caching is disabled.

-m*maxsort* Sets the sort threshold. If there are more than *maxsort* messages in a given channel's queue, then they are processed in directory order without first sorting by submission time. If **-m** is not specified, the value of *maxsort* is given in the tailor file by **MMAXSORT**.

-p Pickup only mode. Indicates that the invoker would like to pickup a passive mail channel.

-s Force linear search of the mail queue. Normally **deliver** will deliver messages in the order they were received which seldom matches the order in the directory. This option is useful if the queue gets so large that **deliver** can no longer deal with sorting the queue in a reasonable time.

-t*hrs* Time limiting. This option prevents **deliver** from attempting to deliver messages which have been in the queue for more than *hrs* hours. For efficiency reasons, this option only applies when the queue is being sorted. If an explicit list of messages was given on the command line, if the **-s** option is in effect, or there are more messages than the maxsort threshold (see the **-m** option), then time limiting does not occur.

-w Watch the delivery. Causes **deliver** to print informative messages on the standard output as it is attempting delivery. This option is passed onto the channel programs which also give informative messages.

-L*logfile* Sets the logfile for this **deliver** to the file specified. The default is to log into the file *msg.log* in the MMDF log directory. This option is only available to the superuser and MMDF.

-T*seconds* Sets the sleep time between background sweeps of the mail queue. This defaults to 10 seconds.

-V*loglevel* Sets the logging level for this **deliver** to the level specified. The *loglevel* should be a valid MMDF logging level string such as FTR. This option is only available to the superuser and MMDF.

See also

submit(ADM), **queue**(F), **mmdftailor**(F)

Value added

deliver is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

dial, uuchat

dial a modem

Syntax

/usr/lib/uucp/dial*X* ***ttyname telno speed***

/usr/lib/uucp/dial*X* **-h [-c]** ***ttyname speed***

/usr/lib/uucp/uuchat ***ttyname speed chat-script***

Description

/usr/lib/uucp/dial*X* dials a modem attached to ***ttyname***. (*X* is a dialer name, such as "HA1200".) The **-h** option is used to hang up the modem.

The **-c** option tells the dialer to wait for a connection and adjust the line rate to match before returning. This feature requires that a **SIGUSR2** be sent back to **uugetty**(ADM) after the modem has been initialized but before the connection has been received. Examine the sample dialers in */usr/lib/uucp* to see how this is done.

uucico(ADM), **ct**(C), and **cu**(C) use **/usr/lib/uucp/dial***X*.

Several dialer programs are provided:

Binary File	Modem
dialHA12	Hayes Smartmodem 1200 or compatible
dialHA24	Hayes Smartmodem 2400 or compatible
dialHA96V	Hayes Smartmodem 9600 or compatible
dialMUL	Multitech Multimodem 224 EH
dialVA3450	Racal Vadic 3451 modem
dialVA96	Racal Vadic 9600 modem
dialTBIT	Telebit Trailblazer Modem

Source for these is provided in their respective *.c* files.

uucico(ADM) invokes **dial** with a ***ttyname***, ***telno*** (telephone number), and ***speed***. **dial** attempts to dial the phone number on the specified line at the given speed. When using **dialHA12** or **dialHA24**, ***speed*** can be a range of baud rates. The range is specified with the form:

lowrate - highrate

where ***lowrate*** is the minimum acceptable connection baud rate and ***highrate*** is the maximum.

The **dial** program returns the status of the attempt through the following dial return codes:

bit 0x80 = 1 The connection attempt failed.

bits 0x0f = ***n*** If bit 0x80 is a 1, then these bits are the dialer error code ***n***:

0 general or unknown error code

1 line is being used

2 a signal has aborted the dialer

3 dialer arguments are invalid

4 the phone number is invalid

5 the baud rate is invalid or the dialer could not connect at the requested baud rate

6 can't open the line

7 ioctl error on the line

8 timeout waiting for connection

9 no dialtone was detected

10 unused

11 unused

12 unused

13 phone is busy

14 no carrier is detected

15 remote system did not answer

Error codes 12-15 are used to indicate that the problem is at the remote end.

If bit 0x80 is a 0, then these bits are used to indicate the actual connection baud rate. If 0, the baud rate is the same as the baud rate used to dial the phone number or the highest baud rate if a range was specified. Otherwise, these four bits are the **CBAUD** bits in the **`struct termio c_flag`** and the **`struct sgttyb sg_ispeed`** and **`sg_ospeed`** tty **ioctl** structures.

You can copy and modify one of the files */usr/lib/uucp/dialHA12.c* etc., to use a different modem. There is a makefile in */usr/lib/uucp* that you can copy and modify to use for compiling a new dialer program.

If you create a **dial** program for another modem, send us the source. User generated **dial** programs will be considered for inclusion in future releases.

The **dial** program to be used on a particular line is specified in the fifth field of the entry for that line in */usr/lib/uucp/Devices*. If there is no **dial** program of that name, then **uucico**, **ct**, and **cu** use a built-in dialer, together with the chat-script of that name in */usr/lib/uucp/Dialers*.

dial -h is executed by **getty** when it is respawned on a line shared between dial-in and dial-out. If there is no **dial** program, then **getty** uses */usr/lib/uucp/uuchat*, passing it the *&* chat-script from */usr/lib/uucp/Dialers*.

Files

/usr/lib/uucp/dial.c*	source files for the dialer programs
/usr/lib/uucp/makefile	makefile to compile new dialer
/usr/lib/uucp/uuchat	

See also

ct(C), **cu**(C), **dialers**(F), **getty**(M), **uucico**(ADM), **uugetty**(ADM)

Notes

You must have the Development System installed in order to compile and install a new **dial** program.

Value added

dial is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

diskusg

generate disk accounting data by user ID

Syntax

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

Description

diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. **diskusg** outputs lines on the standard output, one per user, in the following format: ***uid login #blocks***

where

uid the numerical user ID of the user.

login the login name of the user; and

#blocks 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 (see **labelit**(ADM)).

-p *file* use *file* as the name of the password file to generate login names. */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** (see **acct**(ADM)) which generates total accounting records that can be merged with other accounting records. **diskusg** is normally run in **dodisk** (see **acctsh**(ADM)).

Examples

The following will generate daily disk accounting information:

```
for i in /dev/dsk/0s1 /dev/dsk/0s3; do
        diskusg $i > dtmp.`basename $i` &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

File

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

See also

acct(ADM), **acct**(FP), **acctsh**(ADM)

Standards conformance

diskusg is conformant with:

AT&T SVID Issue 2.

displaypkg

display installed packages

Syntax

displaypkg

Description

The **displaypkg** command will list the names of all the AT&T-style UNIX packages that were installed using the **installpkg** command.

See also

installpkg(ADM), **removepkg**(ADM)

Note

This command does not work on packages installed with **custom**(ADM).

divvy

disk dividing utility

Syntax

divvy [-m | -i [-n] | -D # | -P [#] | -C #1 #2 #3] [*device*]

Description

divvy divides an **fdisk**(ADM) partition into a number of separate areas known as "divisions". A division is identified by unique major and minor device numbers and can be used for a filesystem, swap area, or for isolating bad spots on the device.

The default device is */dev/hd0a*. To access non-default disks, specify a device file on the command line.

With **divvy** you can:

- Divide a disk or **fdisk** partition into separate devices.
- Create new filesystems.
- Change the size of filesystems.
- Remove filesystems.

Options

Options to **divvy** are:

-i Installation only. Disk being divided will contain a *root* filesystem on division 0. Only to be done from a non-active partition, or the root floppy.

-m Disk being divided should be made into a number of mountable filesystems.

-n Non-interactive installation; automatic option. Disk being divided will contain the following:

root filesystem on division 0
swap on division 1
/u filesystem on division 2
scratch on division 5

-D # delete division number #.

-P # print start block number and end block number of division number **#** (or all divisions if **#** is missing).

-C *#1* #2 #3
Create division number ***#1*** starting at block number **#2** and ending at block number **#3**.

Usage

divvy can be used on any character or block disk device file that refers to a UNIX or XENIX partition. If no device is specified, **divvy** defaults to the active UNIX partition on the root hard disk.

The **-i** option is used during installation. It specifies the device being divided will contain a *root* filesystem. With this option, device nodes are created relative to the new *root*, generally a hard disk, instead of the current *root*, often an installation floppy. A *root* filesystem, swap area, and recover area are created. **divvy** prompts for the size of the swap area. If the disk is large enough, then **divvy** prompts for a separate */u* (user) filesystem. **divvy** also prompts for block-by-block control over the layout of the filesystem(s). If the *root* filesystem is large enough to require a scratch filesystem, (more than 40,000 blocks) then **divvy** will prompt for whether one should be created.

The **-m** option is used for initial installation on devices that will not be used as the *root*. It causes the user to be prompted for a number of filesystems.

When **divvy** is invoked from the command line, you see a main menu:

```
n[ame]       Name or rename a division.
c[reate]     Create a new file system on this division.
t[ype]       Select or change filesystem type on new filesystems.
p[revent]    Prevent a new file system from being created on this division.
s[tart]      Start a division on a different block.
e[nd]        End a division on a different block.
r[estore]    Restore the original division table.

Please enter your choice or 'q' to quit:
```

(After the first command has been chosen, an additional option will be displayed in the main menu:

```
u[ndo]       Undo the last change
```

This command may be selected at any time and will reverse the effect of the most recent previous change to the division table.) To choose a command, enter the first letter of the command, then press ⟨Return⟩.

The **divvy** division table might look something like this:

```
+----------+-------------+--------+---+-------------+------------+
| Name     | Type        | New FS | # | First Block | Last Block |
+----------+-------------+--------+---+-------------+------------+
| root     | EAFS        |  no    | 0 |           0|       47402|
| swap     | NON FS      |  no    | 1 |       47403|       50368|
| u        | EAFS        |  no    | 2 |       50369|       70368|
|          | NOT USED    |  no    | 3 |          -|          -|
|          | NOT USED    |  no    | 4 |          -|          -|
|          | NOT USED    |  no    | 5 |          -|          -|
| recover  | NON FS      |  no    | 6 |       70369|       70378|
| hd0a     | WHOLE DISK  |  no    | 7 |           0|       70676|
+----------+-------------+--------+---+-------------+------------+
70379 1K blocks for divisions, 298 1K blocks reserved for the system
```

divvy also displays information about block allocation for system tables and bad tracks.

You can change the name of the device with the **n** command. **divvy** prompts you for the division number (from the **divvy** table displayed above), then for a new name.

The **c** command causes a given division to become a new, empty filesystem when you exit from **divvy**. After using the **c** command, you will see a "yes" in the "New File System?" column. If you use command **p**, the "yes" in the "New File System?" column will change to a "no", and the contents of the division will not change. The **c** command must be used when changing the size of a filesystem.

With the **s** or "start" command, you can start a division on a different block number. With the **e** or "end" command, you can end a division on a different block number. Note that when you alter a division, that filesystem will be remade (**mkfs** is run) and the contents destroyed. If you are resizing your filesystems, make certain you have made backups first.

You can use these commands to change the size of a division. For example, if your disk is similar to the one in the sample **divvy** table above, and you want to make the */u* filesystem larger and the *swap* area smaller, do this:

- Reduce the size of the swap area with the **e** command.
- Increase the size of the */u* division with the **s** command.
- Recreate the */u* filesystem using the **c** command.

Note that if any of the divisions overlap, **divvy** will report an error when you try to exit and put you back in the menus to correct the situation.

The **r** or "restore" command restores the original partition table. This is useful if you make a serious mistake and want to return to where you started.

When you exit from **divvy**, you are prompted whether you want to save any changes you made, or exit without saving the changes. At this time, you can also go back to the **divvy** menu, and may also have the option to reinstall the original, default partition table. If you elect to save your changes, the new partition table will be written to the hard disk and any new filesystems (designated with the **c** command) will be created.

Examples

divvy	active partition on root disk
divvy /dev/hd0a	same
divvy /dev/hd12	second partition on second disk
divvy /dev/rhd12	same

See also

badtrk(ADM), **fdisk**(ADM), **fsck**(ADM), **fsname**(ADM), **hd**(HW), **mkdev**(ADM), **mkfs**(ADM), **mknod**(C)

Notes

divvy requires kernel level support from the device driver. If **divvy** lists the size of a disk as 0 blocks, or displays the following error messages, the device may not support dividing:

```
cannot read division table
```

or:

```
cannot get drive parameters
```

These errors may also occur if the prerequisite programs **dparam**, **fdisk** and **badtrk** are not run correctly.

If you change the size of filesystems (such as */u*) after you have installed a XENIX filesystem, you will have to use the **c** command to re-create the filesystem and reinstall the files that are kept there. This is because the free list for that filesystem has changed. Be sure to backup the files in any filesystem you intend to change, using **backup**(ADM), **tar**(C), or **cpio**(C), before you run **divvy**. To change the size of the *root* filesystem, the operating system must be reinstalled.

During installation, if the filesystem on division 0 (generally *root*) becomes or remains large enough to require a scratch area during **fsck**, and one does not already exist, **divvy** prompts for whether one should be created. (The resulting filesystem, */dev/scratch*, is used by **autoboot** if it runs **fsck**. */dev/scratch* should also be entered when **fsck** prompts for a scratch filename, provided that the filesystem being checked is not larger than the *root* filesystem.) If all disk divisions have been used up, **divvy** will not prompt for a scratch filesystem, even if the root filesystem is large enough to require one.

Should division 0 ever extend beyond the 1024th cylinder of the hard disk, **divvy** warns that the division may not be bootable and offers the user an opportunity to correct the situation. If this occurs during non-interactive installation, **divvy** tries to adjust the divison table automatically, but still warns the user of the fact before allowing manual interaction. **divvy** does not force division 0 to be located within the first 1024 cylinders if the user specifically requires an unusual location.

This utility uses 1K-byte blocks.

Value added

divvy is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

dlvr_audit

produce audit records for subsystem events

Syntax

/etc/auth/dlvr_audit [**-v**] *tstamp event record pid cmd code* [*args* ...]

Description

dlvr_audit is used by programs implementing protected subsystems as the means for sending audit records to the audit subsystem. Because those programs do not have the **writeaudit** privilege, they invoke **dlvr_audit** which sends the data over a message queue to the audit daemon, which appends the record to the audit trail. Because **dlvr_audit** is run as a child process of the process producing the record, it does not have the ability to write the audit device either. The message queue that it uses is only usable by the **audit** user, so **dlvr_audit** must be run **SUID** to the **audit** user. The group is inherited from the invoking process and is checked against those groups associated with protected subsystems. If the group cannot be identified with a protected subsystem, the record is ignored (so that general user programs cannot flood the audit subsystem with invalid messages).

The **-v** flag forces the program to report all of its actions. Normally, this flag is not used so that audit records can be made without the knowledge of the program user.

The required arguments apply to all audit records. The ***tstamp*** argument is the (ASCII number representation of the) time in seconds past Jan 1, 1970 that the audit record was produced. The ***event*** argument is the number of the event type as described in *sys/audit.h*. Similarly, the ***record*** argument is the audit record format type as described in *sys/audit.h*. The ***pid*** is the process ID of the event process. ***cmd*** is the name of the protected subsystem command. ***code*** is specific to the ***event*** type being generated.

There may be 0 or more optional arguments depending on the code. **dlvr_audit** uses the extra arguments to fill in specific fields required by the particular record format.

See also

audit(HW), **authaudit**(S)

"Using the audit subsystem", chapter of the *System Administrator's Guide*

Value added

dlvr_audit is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

dmesg

display the system messages on the console

Syntax

dmesg [–]

Description

The **dmesg** command displays all the system messages that have been generated since the last time the system was booted. If the option – (the dash key) is specified, it displays only those messages that have been generated since the last time the **dmesg** command was performed.

dmesg can be invoked periodically by placing instructions in the file */usr/lib/crontab*. It can also be invoked automatically by the */etc/rc2* scripts whenever the system is booted. See "Notes", below.

dmesg logs all error messages it prints in */usr/adm/messages*. If **dmesg** is invoked automatically, the *messages* file continues to grow and can become very large. The system administrator should occasionally erase its contents.

Files

/etc/dmesg
/usr/adm/messages
/usr/adm/msgbuf

Notes

dmesg is included in this release for backwards compatibility only. The device */dev/error* provides a more flexible means of logging error messages, and is recommended over **dmesg**. See **error**(M) for more information.

See also

cron(C), **error**(M),

Credit

dmesg was developed at the University of California, Berkeley, and is used with permission.

dparam, dkinit

display/change hard disk characteristics

Syntax

dparam [-w]

dparam /dev/rhd[0 | 1]0 [***characteristics*** **]**

Description

dkinit - front end to dparam.

The **dparam** command displays or changes the hard disk characteristics currently in effect. These changes go into effect immediately and are also written to the master boot block for subsequent boots. If a non-standard hard disk is used, this utility must be called before accessing the drive.

(**dkinit** provides a menu-driven front end to **dparam**. For full details on the use of **dkinit**, please refer to the *System Administrator's Guide*.)

-w Causes a copy of */etc/masterboot* to be copied to disk to ensure that non-standard hard disks are supported for the specified drive. This call must precede a call to write non-standard disk parameters for the desired parameters to be saved correctly in the masterboot block.

When called without options or disk characteristics, **dparam** prints the current disk characteristics (on the standard output) for the specified hard disk. These values are printed in the same order as the argument list.

When writing characteristics for the specified hard disk, **dparam** changes the current disk controller status and updates the masterboot block. The argument ordering is critical and must be entered as specified below. All characteristics must be entered when writing disk characteristics, otherwise an error is returned. Hard disk characteristics (in respective order) are:

number of cylinders	total number of cylinders on the hard disk
number of heads	number of heads
reduced write current cylinder	hardware specific, consult your hardware manual
write precompensation cylinder	hardware specific, consult your hardware manual
ecc	number of bits of error correction on I/O transfers, consult your hardware manual
control	hardware specific, consult your hardware manual

landing zone cylinder	where to park heads after shutting down the system
number of sectors per track	number of sectors per track on the hard disk

Examples

dparam -w

dparam /dev/rhd10

dparam /dev/rhd00 700 4 256 180 5 0 640 17

Notes

This utility changes the kernel's view of the hard disk parameters. It may be subject to restrictions imposed by the hardware configuration.

Value added

dparam is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

ecc, eccd

memory Error Correction Code (ECC) facility

Syntax

/etc/ecc

Description

ecc - add/delete entries from the bad page table

eccd - ECC daemon

The memory Error Correction Code (ECC) utilities periodically check RAM for single and double-bit errors to increase data integrity. This feature is specific to Corollary smp RAM used in Corollary and Corollary-compatible systems.

The ECC daemon: eccd

The ECC daemon, or background program, scans the smp RAM checking for single bit errors. Single bit errors themselves are harmless and are automatically corrected by hardware. However, if an additional bit is corrupted at the same location, a double bit error occurs and the system panics.

The ECC daemon helps avoid double bit errors by informing the system administrator of existing single bit errors. Errors are reported via the system console and */usr/adm/messages*. The system administrator should periodically check */usr/adm/messages* for any single bit error notifications and use the **ecc** utility to map the affected 4K page out of memory.

The script that controls this process, */etc/idrc.d/ecc*, is built manually using **mkdev eccd**. The */etc/idrc.d/ecc* script is thereafter invoked automatically when the system enters multiuser mode.

The ecc utility

The administrator should check periodically for memory ECC errors. The ECC errors are displayed on the console and stored in */usr/adm/messages*. The messages appear as follows:

```
found a single-bit error
board=n bad_addr=xxxx
```

where ***n*** is the board number and ***xxxx*** is the address of the error.

ECC errors are mapped and stored in a bad page table using the **ecc** utility. The main **ecc** menu appears as follows:

```
        1. Print Current Bad Memory Page Table
        2. Add Entries to Current Bad Memory Page Table
        3. Delete Entries from Current Bad Memory Page Table

Enter your choice or 'q' to quit:
```

Option 1 prints the current bad memory page table. Option 2 is used to add new entries to the table. Option 3 can be used to delete entires from the bad page table when memory boards or individual SIMMs have been replaced.

You must reboot the system after modifying the bad page table for it to take effect.

Note

This utility works only with Corollary smp RAM and compatibles.

Files

/dev/ecc	ECC device
/etc/idrc.d/ecc	ECC daemon startup script
/etc/idsd.d/ecc	ECC daemon shutdown script
/etc/eccdata	bad page table
/etc/eccdpid	deamon process ID

Value added

ecc is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

eisa

report on boards that are installed on the EISA bus

Syntax

/etc/eisa [*slots* | h | H]

Description

The **eisa** command provides information about the host adapters that are installed on the system. By default, **eisa** reports on the motherboard and 15 slots (slots 0-15). You can specify a number of ***slots*** for which you want a report. If you specify a higher number of slots than you have on your computer, the **eisa** report for those slots is invalid. If you do not specify slot information, use the **H** or **h** options to get usage information.

Here is a sample report:

```
Slot:   EISA ID:        Vendor: Prod#:  Rev#:   EISA Ver:
MB      22 f0 fd 09     HWP     0xfd    01      1
 4      04 90 00 00     ADP     0x000   00
 6      04 90 00 00     ADP     0x000   00
15      22 f0 08 01     HWP     0x080   01
```

These columns have the following meanings:

Slot — logical number of the slot on the EISA bus to which the board is attached. The motherboard is always configured as slot 0, so slot 0 is reported as "MB".

EISA ID — standard EISA ID. The first four digits represent the vendor; the next four digits represent the product number and the revision number for the product.

Vendor — abbreviation for the vendor of the board. The sample report shows that the motherboard and the board in slot 15 are sold by HP and the boards in slots 4 and 6 are sold by Adaptec. Note that this column merely interprets the vendor information given in the first four digits of the EISA ID column.

Prod# — product number of the individual board. These numbers are assigned by the vendor and should be explained in the documentation that accompanies the board.

Rev# — hardware revision number for the board

EISA Ver — EISA version number. Each motherboard is encoded with a version of the EISA specification to which it conforms, and that version number is given in this column.

Notes

The **eisa** command reports only EISA boards that are installed on an EISA system, not the 8- or 16-bit ISA (XT/AT) boards.

Only *root* can execute the **eisa** command.

Diagnostics

eisa returns a 0 value if successful. A return value of 1 indicates a command line error, a return value of 2 indicates that the motherboard was not located, and a return value greater than 2 indicates an unspecified error.

If the slot number that you specify is larger than the actual number of slots, the results are unpredictable; however, the return value is still 0.

Value added

eisa is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fdisk

maintain disk partitions

Syntax

/etc/fdisk [[**-p**] [**-ad** *partition*] [**-c** *partition start size*]
[**-t** *ostype*]] [**-f** *devicename*] [**-f** *devicename*]]

Description

fdisk displays information about disk partitions. It also creates and deletes disk partitions and changes the active partition. **fdisk** functionality is a superset of the MS-DOS command of the same name. **fdisk** is usually used interactively from a menu.

The hard disk has a maximum of four partitions. Only one partition is active at any given time. It is possible to assign a different operating system to each partition. Once a partition is made active, the operating system resident in that partition boots automatically once the current operating system is halted.

The **fdisk** utility reports disk sizes in tracks. The number of tracks available on a hard disk is equal to the number of heads times the number of cylinders. The **fdisk** utility does not allocate the first track or the last cylinder on the hard disk when the "Use Entire Disk for UNIX" option is used. The first track on the hard disk is reserved for **masterboot** and the last cylinder is generally used when running hard disk diagnostics. You should not allocate the last cylinder if you plan to run diagnostics on your hard disk.

For example, if a disk has 4 heads and 615 cylinders, it has 2460 tracks, which **fdisk** reports as tracks 0-2459. If you choose the "Use Entire Disk for UNIX" option, **fdisk** will create a UNIX partition on tracks 1-2455. Track 0 is reserved for **masterboot**, and the last cylinder (tracks 2455-2459) is not assigned with the "Use Entire Disk for UNIX" option.

Partitions are defined by a "partition table" at the end of the master boot block. The partition table provides the location and size of the partitions on the disk. The partition table also defines the active partition. Each partition can be assigned to the UNIX system, DOS, or some other operating system. Once a DOS partition is set up, DOS files and directories resident in the DOS partition may be accessed from the UNIX system partition by means of the **dos**(C) commands. DOS may be booted without the DOS partition being active by entering **dos** at the boot prompt. See **boot**(HW).

Arguments

-p, -a, -d,-c These flags are used to invoke **fdisk** non-interactively. The argument ***number***, below, refers to a valid partition number (1-4).

-p Prints out the disk partition table, one partition to a line. For each partition, **fdisk** displays the following information:

```
partition start stop size status type
```

-a *number* Activates partition ***number***.

-d *number* Deletes partition ***number***.

-c *number start size*
Creates a partition ***number*** that is ***size*** tracks long beginning at track ***start***. The **-c** option is used to use the entire disk for UNIX; the appending of a dash (-) to the end of the command line accomplishes this, as in the following example:

fdisk -c 1 1 -

This syntax is used only during installation. If there are any existing partitions on the disk, this command will fail.

-f *name* Open device ***name*** and read the partition table associated with that device's partition. The default is */dev/rhd00*.

-t *ostype* Specify the partition type of the partition being created, where ***ostype*** is one of the following: UNIX, XENIX, DOS, DOS_12, DOS_16, DOS_32, OS/2 or CCPM. DOS is the same as DOS_16. If no **-t** option is specified, the default partition type is UNIX.

Options

When invoked interactively (without the **-p**, **-a**, **-d**, or **-c** options), **fdisk** displays a prompt and a menu of five options. No changes are made to the partition table on the disk until you enter "q" from the main menu.

1. Display Partition Table
   This option displays a table of information about each partition on the hard disk. The "PARTITION" column gives the partition number. The "STATUS" column tells whether the partition is active (A) or inactive (I). "TYPE" tells whether the partition is a UNIX system partition, a DOS partition, or "other". The option also displays the starting track, ending track and total number of tracks in each partition.

2. Use Entire Disk for UNIX
   fdisk creates one partition that includes all the tracks on the disk, except the first track and the last cylinder. This partition is assigned to the UNIX system and is designated the active partition.

3. Use Rest of Disk for UNIX
   fdisk creates one partition that occupies the largest available contiguous area of the disk. This partition is assigned to the UNIX system and is designated the active partition.

4. Create UNIX Partition
   This option allows the creation of a partition by altering the partition table. **fdisk** reports the number of tracks available for each partition and the number of tracks in use. **fdisk** prompts for the partition to create, the starting track and size in tracks. The change is written to the hard disk when you enter "q" from the main menu.

5. Activate Partition
   This option activates the specified partition. Only one partition may be active at a time. The change is not effective until you exit. The operating system residing in the newly activated partition boots once the current operating system is halted.

6. Delete Partition
   This option requests which partition you wish to delete. **fdisk** reports the new available amount of disk space in tracks. The change is not effective until you exit.

Exit the **fdisk** program by typing a "q" at the main **fdisk** menu. Your changes are now written to the hard disk.

Notes

The minimum recommended size for a UNIX system partition is 40 megabytes.

Since **fdisk** is intended for use with DOS, it may not work with all operating system combinations.

OS/2 partitions are displayed as UNKNOWN.

See also

dos(C), **hd**(HW)

Value added

fdisk is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fdswap

swap default boot floppy drive

Syntax

fdswap [on | off]

Description

fdswap tells the CMOS to swap the default floppy drive used to read boot information at boot time. For example, if your computer defaults to read boot information on drive A, **fdswap on** changes the default drive to drive B.

fdswap with no arguments reports the current **fdswap** state, on or off. **fdswap off** switches the drive setting back to the default configuration. Changing the drives take effect on the next boot of the system.

Notes

Support for this functionality is only available on a small number of machines. The ROMs must recognize and interpret the CMOS flag that specifies that the floppy drives are swapped.

ff

list file names and statistics for a filesystem

Syntax

/etc/ff [*options*] *special*

Description

The **ff** command reads the i-list and directories of the ***special*** file, assuming it is a file system. Inode data is saved for files which match the selection criteria. Output consists of the path name for each saved inode, plus other file information requested using 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:

path-name i-number

With all options enabled, output fields would be:

path-name 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 path name.

-l Generate a supplementary list of all path names for multiple-linked files.

-p*prefix* The specified ***prefix*** will be added to each generated path name. The default is "." (dot).

-s Print the file size, in bytes, after each path name.

-u Print the owner's login name after each path name.

-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***.

See also

find(C), **ncheck**(ADM)

Notes

If the **-l** option is not specified, only a single path name out of all possible ones is generated for a multiple-linked inode. If **-l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

This command only works on UNIX filesystems.

fixmog, cps

make all system files consistent with the authentication database,
make specific system files consistent with the authentication database

Syntax

/tcb/bin/fixmog [**-i**] [**-v**]

/tcb/bin/cps [***absolute_pathnames***]

Description

fixmog attempts to correct inconsistencies found by **integrity**(ADM). **integrity** traverses the File Control database and compares each entry to the real file in the filesystem. Each file is checked to ensure it has the specified owner, group, access permissions and type. **fixmog** changes the owner, group and access permissions of files to those in the File Control database. If the **-i** (interactive) option is used, **fixmog** requests confirmation before making any changes. If the **-v** (verbose) option is in effect, **fixmog** displays a line detailing each change made. The **-i** option overrides the **-v** option. If a file is of the wrong type (for example, a regular file when it should be a directory), a message giving the expected and actual types is output and no changes are made to that file. If a change fails, an error message giving the change attempted is output.

Like **fixmog**, **cps** is used to correct problems in the TCB. However, **cps** checks specified files rather than all files in the File Control database. **cps** is used primarily by the crash recovery script to ensure files critical to the TCB exist and have the correct owner, group and access permissions specified in the File Control database.

cps accepts absolute pathnames of directories and files to be created. Absolute pathnames are complete pathnames (for example */tcb/bin/cps*) as opposed to relative pathnames (for example *../file*). An entry (containing a mode) for each component of each pathname must be present in the File Control database, otherwise a fatal error is returned.

Each missing element of each path is created as specified in the File Control database. Elements of each path that already exist, but have incorrect owner, group or access permissions, are changed so they agree with their File Control database entries.

If no parameters are supplied, the pathnames are read from the standard input, which should contain absolute pathnames separated by newlines.

Only the super user can use the **fixmog** and **cps** commands.

File

/etc/auth/system/files File Control database

See also

integrity(ADM), **fixmog**(ADM), **tcbck**(ADM)

Diagnostics

fixmog returns an exit status of 1 if the user attempting to run the program is not the superuser, invalid options were specified or the **integrity**(ADM) command could not be run: otherwise 0 is returned. Errors cause appropriate error messages to be displayed.

cps returns an exit status of 1 if a fatal error was detected; otherwise 0 is returned if no changes were required and 2 if any changes to the file system were made. Errors cause appropriate error messages to be displayed.

Notes

cps converts the pathnames supplied to canonical pathnames, i.e. ones that don't contain consecutive "/"s, and none of the directories are "." or "..". This enables pathnames to be looked up in the File Control database where pathnames should also be in this format.

Value added

fixmog and **cps** are an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fixperm

correct or initialize file permissions and ownership

Syntax

/etc/fixperm [-acCDfgilLnOpsSUvwX] [-d *pkg*] [-u *pkg*] *specfile*

Description

For each line in the specification file ***specfile***, **fixperm** makes the listed pathname conform to a specification. **fixperm** is typically used to configure a UNIX system upon installation. It can only be invoked by a super user, and it only works from the root directory. If it is invoked from any other directory, incorrect results will be returned.

The specification file has the following format: Each non-blank line consists of either a comment or an item specification. A comment is any text from a number sign (#) up to the end of the line. There is one item specification per line. User and group ID numbers must be specified at the top of the specification file for each user and group mentioned in the file. The syntax for the definition section is simple: the first field indicates the type of ID (either **uid** or **gid**), the second contains the name reference for the ID, and the third is the corresponding numeric ID. For example:

```
uid     root    0
```

An item specification consists of a package specifier, a permission specification, owner and group specifications, the number of links on the file, the file name, and an optional volume number.

The package specifier is an arbitrary string which is the name of a package within a distribution set. A package is a set of files.

After the package specifier is a permission specification. The permission specification consists of a file type, followed by a numeric permission specification. The file type is one of the following characters:

a Archive.

b Block device.

c Character device.

d Directory.

e Empty file (create if **-c** option given).

f Text file.

p Named pipe.

o OK. It indicates to **fixperm** that there should be no file type checking allowing any format or contents in what would normally be the header section of an executable file. For example, data files and encrypted files should be of type "o".

x Executable.

C Compress file. This is optional and can be used in addition to the previous file types described above. When used with the **-C** option, it enables the file to be compressed.

U Uncompress file. This is also optional and enables the file to be uncompressed when used with the **-U** option.

If the file type is used as an upper-case letter, then the file associated with it is optional, and **fixperm** will not return an error message if it does not exist.

The numeric permission conforms to the scheme described in **chmod**(C). The owner and group are in the third column separated by a slash: for example: "bin/bin". The fourth column indicates the number of links. If there are links to the file, the next line contains the linked filename with no other information. The fifth column is a pathname. The pathname must be relative, that is, not preceded by a slash " / ". The sixth column is only used for special files, giving the major and minor device numbers, or volume numbers.

Options

The following options are available from the command line, unless otherwise noted:

-a Ensures that all files specified in the list exist on the hard disk.

-c Create empty files and missing directories. Also creates (or modifies) device files.

-C Compress all types of files with the additional "C" permission specification.

-d ***pkg*** Process input lines beginning with given package specifier string (see above). For instance, **-d**BASE processes only items specified as belonging to the Basic utilities set. The default action is to process all lines.

-D List directories only on standard output. Does not modify target files.

-f List files only on standard output. Does not modify target files.

-g Instructs **fixperm** to list devices as specified in the permlist (similar to the **-f** flag, which lists files on standard output). No changes are made as a result of this flag.

-i (Available from a program or shell script only)

Check only if the selected packages are installed. Return values are:

0: package completely installed
3: not found
4: package not installed
5: package partially installed

-l List files and directories on standard output. Does not modify target files.

-L List files on standard output but any compressed files (that is, C file types) are printed with a ".Z" suffix.

-n Report errors only. Does not modify target files.

-O Omit link names from lists when used with the list options; **-D**, **-f**, **-l**, **-L** or **-w**.

-p Override default uid/gid found in */etc/passwd* and */etc/group* with the value found in the permlist. Because UNIX and XENIX have different values for certain uid and gids (for example, in UNIX bin=2, and XENIX bin=3) the default value is gleaned from the */etc/passwd* and */etc/group* files. This option forces the values to be taken from the perms list. It also generates a warning if the perms list doesn't include */etc/passwd* and */etc/group*.

-s Modify special device files in addition to the rest of the permlist.

-S Issues a complaint if files are not in *x.out* format.

-u ***pkg*** Like **-d**, but processes items that are not part of the given package.

-U Uncompress all types of files with the additional "C" permission specification.

-v Verbose, in particular, issues a complaint if executable files are word swapped, not fixed stack, not separate "I" and "D", or not stripped.

-w Lists where (what volume) the specified files or directories are located.

-X Print only files and/or directories not installed.

The following two lines make a distribution and invoke **tar**(C) to archive only the files in */etc/perms/inst* on */dev/sample*:

```
/etc/fixperm -f /etc/perms/inst > list
tar cfF /dev/sample list
```

This example reports BASE package errors:

```
/etc/fixperm -nd BASE /etc/perms/
```

or

```
/etc/fixperm -nd BASE /etc/perms/filename
```

Note

Usually **fixperm** is only run by a shell script at installation.

See also

custom(ADM)

Value added

fixperm is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fsave

interactive, error-checking filesystem backup

Syntax

fsave *filesystem* [*backupinfo*] [*mediainfo*] [*sitename*]

Description

fsave is used by **fsphoto**(ADM) to provide a semi-automated interface to **xbackup**(ADM) and **cpio**(C) for backing up filesystems. Operator intervention is required to mount and dismount tapes or floppies at the appropriate times, but is kept to a minimum to reduce the potential for error.

The operator is prompted each time some action is required, such as mounting or unmounting a tape or floppy. These prompts, and their possible selections, are described below.

For all prompts, an answer of **h**, **H**, or **?** will display a short summary of the possible answers.

Filesystem dump (backup)

The following prompt displays the defaults (gleaned from the *schedule* database file) and presents options to alter them:

```
Level dumplevel dump of filesystem filesystem , date
        media size: size feet [or Kb]
        media drive: drive
This media will be saved for howlong, and is howvital.

M)ounted volume, P)ostpone, C)heck or F)ormat volumes, R)etension or H)elp:
```

The values displayed dictate the following instructions: *filesystem* is to be backed up using *size*-foot long magtapes (or *size*-kilobyte big floppies) mounted on drive *drive*. The *media* will be saved for *howlong* ("1 year", "2 months", etc.), and being a level *dumplevel* dump, is *howvital* ("critical", "precautionary", etc.).

The menu options are:

m — A volume of the asked for *size* has been mounted (write-enabled), so begin the dump.

m*newsize* — Insufficient volumes of the originally asked for size are available, so a *newsize* big volume has been mounted instead. If the dump extends across more than one volume, each volume must be of the same size.

p — Postpone this backup until later (**fsphoto** will automatically retry this *filesystem* next time it is run).

c Recheck the volumes used to back up ***filesystem*** for errors. This answer is useful when a dump mysteriously fails and **fsave** is starting over from the beginning, but the operator doesn't believe there really is a problem (for example, the tape drive was accidentally left offline or the floppy door was left open), and wants to check the volumes again.

f Format the currently mounted volume (useful mainly for floppies).

r Retension cartridge tape using **tape**(C).

If multiple volumes are required, **backup** will pause for the next volume to be mounted. Be certain to keep track of the volume order.

Format check

The format of "critical" volumes is checked using **dumpdir**(ADM):

```
Check vital volumes for format errors
M)ounted first volume, S)kip format check, or H)elp:
```

The menu options are:

m The first volume has been (or still is) mounted, and **dumpdir** can now check the volume format.

s Skip checking the volume format, and continue on to the read error check (below).

The format is not always checked, but when it is, the first volume written must be mounted.

Read error check

All volumes created using **xbackup**(ADM) are read using **xrestore**(ADM), which checks for errors during reading. If an error occurs, the dump is declared unsuccessful and is retried from the beginning.

```
Check vital volumes for read errors
M)ounted which volume, E)rror on previous volume, D)one, S)kip checks, or H)elp:
```

The menu options are:

m The ***which*** ("first" or "next") volume has been mounted on the drive and is ready to be checked for read errors.

e An error occurred on the last volume checked, and the dump should be retried.

d All volumes have been checked and no errors occurred, so the filesystem has been successfully backed up.

s Don't bother (skip) checking the rest of the volumes for read errors.

Every volume should be checked for read errors; **xrestore** requires the volumes to be checked in first-to-last order. Volumes that produce read errors should be marked "suspect", discarded, and the dump run once again.

After the backup has been successfully performed, instructions are given on how to label the volumes.

Arguments

fsave is normally run by **fsphoto**, which passes all the proper arguments based on the **schedule**(ADM) database.

filesystem The filesystem to be backed up.

dumpinfo A set of blank-separated strings that give some optional information about this backup:

dumplevel size savetime importance marker

Each of these component strings may be quoted and can thus contain spaces.

- ***dumplevel*** The level of the dump to be performed. This is a single digit from **0** to **9** (passed to **xbackup**), or the letter **x** (which means no backup is to be done). The default is to perform a level 0 backup.
- ***size*** The size of the media volumes that should be used. This should be in feet for tapes and kilobytes for floppies. A ***size*** of **-** means to use the first size listed in ***mediainfo***. This is the default.
- ***savetime*** How long this backup is to be saved (for example, "3 months"). Default is "1 year".
- ***importance*** How important is this backup? (For example, "critical" or "precautionary.") Those which are "critical" have their format checked by **xdumpdir**. Default is "important".
- ***marker*** Either "none" (the default) or an additional label to place on each volume (for example, "a pink sticker").

A typical ***dumpinfo*** might look like:

```
9 1200 "2 weeks" useful "a blue X"
```

which specifies that a level 9 dump is to be performed on a 1200 foot tape (or 1200 kilobyte floppy) which will be saved for 2 weeks and is to be marked with a blue cross (in addition to a more descriptive label). This backup is merely considered "useful" and thus will not be checked by **xdumpdir**.

mediainfo A set of blank-separated strings that give some optional information about this the media to be used:

***drive* d *density sizes...* [*format*]**
***drive* k *sizes...* [*format*]**

drive The name of backup device to use. The default is */dev/rmt0*.

k *sizes...* If **k** is specified, ***drive*** is assumed to be a floppy, and the list of ***sizes*** which follow define the allowable capacities of the floppies that can be used (in kilobytes).

d *density sizes...*
Otherwise, **d** must be specified. In this case, ***drive*** is assumed to be a magtape at ***density*** BPI, in one of the possible ***sizes*** (in feet).

format The command used to format the tape or floppy so described.

A ***mediainfo*** describing 9-track magtape would be:

```
media /dev/rmt0 d 1600 2400 1200 600
media /dev/rmt2 d 800  1400 1200 600
```

which specifies that */dev/rmt0* is a 1600 BPI magtape capable of handling 2400, 1200, and 600 foot reels, and that */dev/rmt2* is the 800 BPI device.

A floppy might be described with:

```
media /dev/fd0 k 1024 format /dev/fd0
```

which describes device */dev/fd0* as a megabyte (1024 kilobytes) floppy formatted by the command:

format /dev/fd0

sitename Where this backup was made (for example, the name of the company or which building). Note that the **uucp**(C) nodename from */etc/systemid* is automatically placed on the volume labels.

Only the super user can execute the **fsave** command.

Files

/etc/systemid	Name of this machine.
/etc/ddate	**xbackup**-maintained record of last time each filesystem was backed-up.
/dev/tty	Always-existent character-special device.

See also

basename(C), **cpio**(C), **dumpdir**(ADM), **fsphoto**(ADM), **schedule**(ADM), **xbackup**(ADM), **xrestore**(ADM)

Diagnostics

A successful backup exits successfully (0), but errors generate a complaint and an exit status of 1. **fsave** complains about illegal or incorrect arguments, and exits with a status of 2.

If the backup of *filesystem* is postponed, **fsave** exits with a status of 3.

Value added

fsave is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fsck, dfsck

check and repair filesystems

Syntax

/etc/fsck [*options*] [*filesystem*] ...

/etc/dfsck [*options1*] *filesys1* ... -[*options2*] *filesys2* ...

Description

dfsck - check and repair filesystems

The **fsck** command audits and interactively repairs inconsistent conditions for all supported filesystems. If the filesystem is consistent, the number of files, number of blocks used, and number of blocks free are reported. If the filesystem 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 the loss may be determined from the diagnostic output. (An experienced operator can resolve discrepancies manually using **fsdb**(ADM), the filesystem debugger.) 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 the action of the **-n** option.

The following flags are interpreted by **fsck**:

-a (Autoboot.) When called with this option, fsck examines the FSCKFIX flag in the */etc/default/boot* file. If FSCKFIX is set to YES, **fsck** behaves as if it had been called with the **-y** flag.

-b (S51K and AFS filesystems only.) Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

-C[*clustersize*]
(S51K filesystems only.) Converts the named S51K filesystem into an AFS (Acer Fast Filesystem). The **-s** option must also be present. The ***clustersize*** argument must be a power of 2 and less than 16 (8 is the recommended value). The increase in speed that is possible with a fast filesystem will not be immediately apparent; it will take affect only with the new files added to the filesystem. There is little or no benefit in transforming a filesystem that is nearly full; if it is within a few blocks of being full, the conversion will not work. (This option can only be used to convert an S51K fileystem.)

-E Converts the named AFS filesystem to Extended Acer Fast Filesystem (EAFS), which includes support for long filenames and symbolic links. Can be combined with **-C** option to convert an S51K filesystem to EAFS.

-y Assumes a *yes* response to all questions asked by **fsck**.

-n Assumes a *no* response to all questions asked by **fsck**; do not open the filesystem for writing.

-s*b*:*c* Ignores the actual free list and (unconditionally) reconstructs a new one by rewriting the super-block of the filesystem. The filesystem *must* be unmounted while this is done.

The **-s***b*:*c* option allows for creating an optimal free-list organization. The following forms are supported:

-s
-s***Blocks-per-cylinder*:*Blocks-to-skip*** (filesystem interleave)
(for anything else)

If *b*:*c* is not given, then the values used when the filesystem was created are used. If these values were not specified, then a reasonable default value is used.

-S Conditionally reconstructs the free list. This option is like **-s***b*:*c* above except that the free list is rebuilt only if there are no discrepancies discovered in the filesystem. Using **-S** forces a *no* response to all questions asked by **fsck**. This option is useful for forcing free list reorganization on uncontaminated filesystems.

-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. Make certain you leave a space between the **-t** and the filename, or **fsck** will use the entire filesystem as a scratch file and erase the entire disk. If you created a scratch filesystem during installation then you can use */dev/scratch* as the filename, provided that the filesystem being checked is no larger than the *root* filesystem. Without the **-t** flag, **fsck** prompts the operator for the name of the scratch file. The file chosen should not be on the filesystem being checked, and if it is not a special file or did not already exist, it is removed when **fsck** completes. If the system has a large hard disk there may not be enough space on another filesystem for the scratch file. In such cases, if the system has a floppy drive, use a blank, formatted floppy in the floppy drive with (for example) */dev/fd0* specified as the scratch file.

-q Quiet **fsck**. Do not print size-check messages in Phase 1. Unreferenced FIFO files will selectively be removed. If **fsck** requires it, counts in the super-block will be 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 will be reconstructed (Phase 6) if it is necessary.

-rr (XENIX filesystems only.) Recovers the root filesystem. The required ***filesystem*** argument must refer to the root filesystem, and preferably to the block device (normally */dev/root*). This switch implies **-y** and overrides **-n**. If any modifications to the filesystem are required, the filesystem will be automatically mounted.

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

Inconsistencies checked are as follows:

- Blocks claimed by more than one inode or the free list
- Blocks claimed by an inode or the free list outside the range of the filesystem
- Incorrect link counts
- Size checks:
  Incorrect number of blocks
  Directory size not 16-byte aligned
- Bad inode format
- Blocks not accounted for anywhere
- Directory checks:
  File pointing to unallocated inode
  Inode number out of range
- Super-block checks:
  More than 65536 inodes
  More blocks for inodes than there are in the filesystem
- Bad free block list format
- Total free block 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. The name assigned is the inode number. The only restriction is that the directory *lost+found* must preexist in the root of the filesystem 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).

dfsck allows two filesystem checks on two different drives simultaneously. ***options1*** and ***options2*** are used to pass options to **fsck** for the two sets of filesystems. A "**-**" is the separator between filesystem groups.

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

Do not use **dfsck** to check the *root* filesystem.

Files

/etc/checklist	Contains default list of filesystems to check
/etc/default/boot	Automatic boot control

See also

autoboot(ADM), **checklist**(F), **filesystem**(FP), **fsdb**(ADM), **init**(M)

Notes

The directory */etc/fscmd.d/TYPE* contains programs for each filesystem type; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks.

fsck will not run on a mounted non-raw filesystem unless the filesystem is the root filesystem or unless the **-n** option is specified and no writing out of the filesystem will take place. If any such attempt is made, a warning is displayed and no further processing of the filesystem is done for the specified device.

Although checking a raw device is almost always faster, there is no way to tell if the filesystem is mounted. Cleaning a mounted filesystem will almost certainly result in an inconsistent super-block.

Warning

Filesystems created under UNIX-86 version 3.0 are not supported under UNIX System V/386 Release 3.2 because the word ordering in type long variables has changed. **fsck** is capable of auditing and repairing UNIX System V/386 version 3.0 filesystems if the word ordering is correct.

For the root filesystem, **fsck -b /dev/root** should be run. For all other filesystems, **fsck /dev/??** on the *unmounted* block device should be used.

Diagnostics

Initialization phase

Command syntax is checked. Before the filesystem check can be performed, **fsck** sets up certain tables and opens some files. The **fsck** terminates on initialization errors.

General errors

Three error messages may appear in any phase. While they seem to offer the option to continue, it is generally best to regard them as fatal, end the run, and investigate what may have caused the problem.

`CAN NOT SEEK: BLK B (CONTINUE?)`
The request to move to a specified block number *B* in the filesystem failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

`CAN NOT READ: BLK B (CONTINUE?)`
The request for reading a specified block number *B* in the filesystem failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

`CAN NOT WRITE: BLK B (CONTINUE?)`
The request for writing a specified block number *B* in the filesystem failed. The disk may be write-protected.

Meaning of yes/no responses

Prompt	*n*(no)	*y*(yes)
CONTINUE?	Terminates program. (This is the recommended response.)	Attempts to continue to run filesystem check. Often, however, the problem persists. The error condition does not allow a complete check of the filesystem. A second run of **fsck** should be made to recheck this filesystem.

Phase 1: check blocks and sizes

This phase checks the inode list.

Meaning of yes/No responses—Phase 1

Prompt	*n*(no)	*y*(yes)
CONTINUE?	Terminates the program. (Recommended response.)	Continues with the program. This error condition means that a complete check of the filesystem is not possible. A second run of **fsck** should be made to recheck this filesystem.
CLEAR?	Ignores the error condition. A **NO** response is only appropriate if the user intends to take other measures to fix the problem.	Deallocates i-node *I* by zeroing its contents. This may invoke the `UNALLOCATED` error condition in Phase 2 for each directory entry pointing to this i-node.

Phase 1 error messages

`EMPTY SYMLINK (CLEAR?)`
There is no pathname associated with a symbolic link.

`UNKNOWN FILE TYPE I=I (CLEAR?)`
The mode word of the i-node *I* suggests that the i-node is not a pipe, special character i-node, regular i-node, or directory i-node. This is also displayed when a non-EAFS version of **fsck** is run on a filesystem containing symbolic links.

`LINK COUNT TABLE OVERFLOW (CONTINUE?)`
An internal table for **fsck** containing allocated i-nodes with a link count of zero has no more room.

`B BAD I=I`
I-node *I* contains block number *B* with a number lower than the number of the first data block in the filesystem or greater than the number of the last block in the filesystem. This error condition may invoke the `EXCESSIVE BAD BLKS` error condition in Phase 1 if i-node *I* has too many block numbers outside the filesystem range. This error condition invokes the `BAD/DUP` error condition in Phase 2 and Phase 4.

`EXCESSIVE BAD BLOCKS I=I (CONTINUE?)`
There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the filesystem or greater than the number of the last block in the filesystem associated with i-node *I*.

`B DUP I=I`
I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition may invoke the `EXCESSIVE DUP BLKS` error condition in Phase 1 if i-node *I* has too many block numbers claimed by other i-nodes. This error condition invokes Phase 1B and the `BAD/DUP` error condition in Phase 2 and Phase 4.

`EXCESSIVE DUP BLKS I=I (CONTINUE?)`
There is more than a tolerable number (usually 10) of blocks claimed by other i-nodes.

`DUP TABLE OVERFLOW (CONTINUE?)`
An internal table in **fsck** containing duplicate block numbers has no more room.

`POSSIBLE FILE SIZE ERROR I=I`
The i-node *I* size does not match the actual number of blocks used by the i-node. This is only a warning. If the **-q** option is used, this message is not printed.

`DIRECTORY MISALIGNED I=I`
The size of a directory i-node is not a multiple of 16. This is only a warning. If the **-q** option is used, this message is not printed.

`PARTIALLY ALLOCATED INODE I=I (CLEAR?)`

I-node *I* is neither allocated nor unallocated.

Phase 1B: rescan for more DUPS

When a duplicate block is found in the filesystem, the filesystem is rescanned to find the i-node that previously claimed that block. When the duplicate block is found, the following information message is printed:

`B DUP I=I`

I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition invokes the `BAD/DUP` error condition in Phase 2. I-nodes with overlapping blocks may be determined by examining this error condition and the `DUP` error condition in Phase 1.

Phase 2: check path names

This phase removes directory entries pointing to bad i-nodes found in Phase 1 and Phase 1B.

Meaning of yes/no responses—Phase 2

Prompt	*n*(no)	*y*(yes)
FIX?	Terminates the program since **fsck** will be unable to continue.	In Phase 2, a y(yes) response to the `FIX?` prompt says: Change the root i-node type to "directory." If the root i-node data blocks are not directory blocks, a very large number of error conditions are produced.
CONTINUE?	Terminates the program.	Ignores `DUPS/BAD` error condition in root i-node and attempt to continue to run the filesystem check. If root i-node is not correct, then this may result in a large number of other error conditions.
REMOVE?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Removes duplicate or unallocated blocks.

Phase 2 error messages

`ROOT INODE UNALLOCATED. TERMINATING`
The root i-node (always i-node number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem. The program stops.

`ROOT INODE NOT DIRECTORY (FIX?)`
The root i-node (usually i-node number 2) is not directory i-node type.

`DUPS/BAD IN ROOT INODE (CONTINUE?)`
Phase 1 or Phase 1B found duplicate blocks or bad blocks in the root i-node (usually i-node number 2) for the filesystem.

`I OUT OF RANGE I=I NAME=F (REMOVE?)`
A directory entry *F* has an i-node number *I* that is greater than the end of the i-node list.

`UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE?)`
A directory entry *F* has an i-node *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed. If the filesystem is not mounted and the **-n** option was not specified, the entry is removed automatically if the i-node it points to is character size 0.

`DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE?)`
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory entry *F*, directory i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

`DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE?)`
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file entry *F*, i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed.

`BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T`
This message only occurs when the **-D** option is used. A bad block was found in *DIR* i-node *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".\&." entries, and embedded slashes in the name field. This error message means that the user should at a later time either remove the directory i-node if the entire block looks bad or change (or remove) those directory entries that look bad.

Phase 3: check connectivity

This phase is concerned with the directory connectivity seen in Phase 2.

Meaning of yes/no responses—Phase 3

Prompt	*n*(no)	*y*(yes)
RECONNECT?	Ignores the error condition. This invokes the UNREF error condition in Phase 4. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Reconnects directory i-node *I* to the filesystem in directory for lost files (usually *lost+found*). This may invoke a *lost+found* error condition if there are problems connecting directory i-node *I* to *lost+found*. This invokes CONNECTED information message if link was successful.

Phase 3 error messages

`UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT?)`

The directory i-node *I* was not connected to a directory entry when the filesystem was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed. The **fsck** program forces the reconnection of a nonempty directory.

`SORRY. NO lost+found DIRECTORY`

There is no *lost+found* directory in the root directory of the filesystem; **fsck** ignores the request to link a directory in *lost+found*. This invokes the UNREF error condition in Phase 4. Possible problem with access modes of *lost+found*.

`SORRY. NO SPACE IN lost+found DIRECTORY`

There is no space to add another entry to the *lost+found* directory in the root directory of the filesystem; **fsck** ignores the request to link a directory in *lost+found*. This invokes the `UNREF` error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger (see Procedure 5.2).

`DIR I=I1 CONNECTED. PARENT WAS I=I2`

This is an advisory message indicating a directory i-node ***I1*** was successfully connected to the *lost+found* directory. The parent i-node ***I2*** of the directory i-node ***I1*** is replaced by the i-node number of the *lost+found* directory.

Phase 4: check reference counts

This phase checks the link count information seen in Phases 2 and 3.

Meaning of yes/No responses—Phase 4

Prompt	*n*(no)	*y*(yes)
RECONNECT?	Ignores this error condition. This invokes a `CLEAR` error condition later in Phase 4.	Reconnect i-node ***I*** to filesystem in the directory for lost files (usually *lost+found*). This can cause a *lost+found* error condition in this phase if there are problems connecting i-node ***I*** to *lost+found*.
CLEAR?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Deallocates the i-node by zeroing its contents.
ADJUST?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Replaces link count of file i-node ***I*** with ***Y***.
FIX?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Replaces count in superblock by actual count.

Phase 4 error messages

`UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT?)`
I-node *I* was not connected to a directory entry when the filesystem was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **-n** option is omitted and the filesystem is not mounted, empty files are cleared automatically. Nonempty files are not cleared.

`SORRY. NO lost+found DIRECTORY`
There is no *lost+found* directory in the root directory of the filesystem; **fsck** ignores the request to link a file in *lost+found*. This invokes the `CLEAR` error condition later in Phase 4.

`SORRY. NO SPACE IN lost+found DIRECTORY`
There is no space to add another entry to the *lost+found* directory in the root directory of the filesystem; **fsck** ignores the request to link a file in *lost+found*. This invokes the `CLEAR` error condition later in Phase 4. Check size and contents of *lost+found*.

`(CLEAR)`
The i-node mentioned in the immediately previous `UNREF` error condition cannot be reconnected.

`LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULDBE Y (ADJUST?)`
The link count for i-node *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed.

`LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST?)`
The link count for i-node *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed.

`LINK COUNT F I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST?)`
The link count for *F* i-node *I* is *X* but should be *Y*. The filename *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

`UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)`
I-node *I*, which is a file, was not connected to a directory entry when the filesystem was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **-n** option is omitted and the filesystem is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

`UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)`
I-node *I*, which is a directory, was not connected to a directory entry when the filesystem was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed. If the **-n** option is omitted and the filesystem is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared.

`BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)`
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

`BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR?)`
Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

`FREE INODE COUNT WRONG IN SUPERBLK (FIX?)`
The actual count of the free i-nodes does not match the count in the super-block of the filesystem. If the **-q** option is specified, the count will be fixed automatically in the super-block.

Phase 5: check free list

This phase checks the free-block list.

Meaning of yes/no responses—Phase 5

Prompt	*n*(no)	*y*(yes)
CONTINUE?	Terminates the program.	Ignores rest of the free-block list and continue execution of **fsck**. This error condition will always invoke `BAD BLKS IN FREE LIST` error condition later in Phase 5.
FIX?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Replaces count in super-block by actual count.
SALVAGE?	Ignores the error condition. A NO response is only appropriate if the user intends to take other measures to fix the problem.	Replaces actual free-block list with a new free-block list. The new free-block list will be ordered according to the gap and cylinder specs of the **-s** or **-S** option to reduce time spent waiting for the disk to rotate into position.

Phase 5 error messages

`EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE?)`
: The free-block list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the filesystem or greater than the last block in the filesystem.

`EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE?)`
: The free-block list contains more than a tolerable number (usually 10) of blocks claimed by i-nodes or earlier parts of the free-block list.

`BAD FREEBLK COUNT`
: The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition will always invoke the `BAD FREE LIST` condition later in Phase 5.

`X BAD BLKS IN FREE LIST`
: *X* blocks in the free-block list have a block number lower than the first data block in the filesystem or greater than the last block in the filesystem. This error condition will always invoke the `BAD FREE LIST` condition later in Phase 5.

`X DUP BLKS IN FREE LIST`
: *X* blocks claimed by i-nodes or earlier parts of the free-block list were found in the free-block list. This error condition will always invoke the `BAD FREE LIST` condition later in Phase 5.

`X BLK(S) MISSING`
: *X* blocks unused by the filesystem were not found in the free-block list. This error condition will always invoke the `BAD FREE LIST` condition later in Phase 5.

`FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)`
: The actual count of free blocks does not match the count in the superblock of the filesystem.

`BAD FREE LIST (SALVAGE?)`
: This message is always preceded by one or more of the Phase 5 information messages. If the **-q** option is specified, the free-block list will be salvaged automatically.

Phase 6: salvage free list

This phase reconstructs the free-block list. It has one possible error condition that results from bad blocks-per-cylinder and gap values.

Phase 6 error messages

`DEFAULT FREE-BLOCK LIST SPACING ASSUMED`

This is an advisory message indicating the blocks-to-skip (gap) is greater than the blocks-per-cylinder, the blocks-to-skip is less than 1, the blocks-per-cylinder is less than 1, or the blocks-per-cylinder is greater than 500. The values of 7 blocks-to-skip and 400 blocks-per-cylinder are used.

Cleanup phase

Once a filesystem has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the filesystem and status of the filesystem.

Cleanup phase messages

`X files Y blocks Z free`

This is an advisory message indicating that the filesystem checked contained *X* files using *Y* blocks leaving *Z* blocks free in the filesystem.

`***** BOOT UNIX (NO SYNC!) *****`

This is an advisory message indicating that a mounted filesystem or the root filesystem has been modified by **fsck**. If the UNIX system is not rebooted immediately without **sync**, the work done by **fsck** may be undone by the in-core copies of tables the UNIX system keeps. If the **-b** option of the **fsck** command was specified and the filesystem is *root*, a reboot is automatically done.

`***** FILE SYSTEM WAS MODIFIED *****`

This is an advisory message indicating that the current filesystem was modified by **fsck**.

fsdb

filesystem debugger

Syntax

/etc/fsdb *special* [–]

Description

fsdb can be used to patch up a damaged filesystem after a crash. It has conversions to translate block and inumbers 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 filesystem tree.

fsdb should only be used on an unmounted filesystem.

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 filesystem 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 will therefore work 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 inumber 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

, 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 short words

e print as decimal short words

x print as hexadecimal short 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 new-line character will increment 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 filesystem. 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 inumber and the character representation of the entry name. I-nodes 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 inumber 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 filesystem.
d5i.fc	changes the current inode to that associated with the 6th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0x	prints the superblock of this filesystem in hexadecimal.

2i.a0b.p3d	prints the first 3 entries in the root directory. This example also shows how several operations can be combined on one command line.
2i.a0b.d7=3	changes the inumber for the seventh directory slot in the root directory to 3.
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.

Notes

The directory */etc/fscmd.d/TYPE* contains programs for each filesystem type; each of these programs applies some appropriate heuristic to determine whether the supplied ***special*** file is of the type for which it checks.

See also

dir(FP), **filesystem**(FP), **fsck**(ADM),

"Troubleshooting your system" in the *System Administrator's Guide.*

fsname

print or change the name of a file system

Syntax

/etc/fsname [-p] [-s *name*] */dev/device*

Description

The **fsname** utility is used to print or change the name of a XENIX filesystem. The options are:

-p Select the "pack" name field instead of the filesystem name field.

-s *name* Change the specified field in the superblock.

The default action is to print the name of the filesystem.

Note

This program only works on XENIX filesystems. For other filesystem types, use **labelit**(ADM).

See also

filesystem(FP), **labelit**(ADM), **mkfs**(ADM), **ustat**(S)

Value added

fsname is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

fsphoto

perform periodic semi-automated system backups

Syntax

fsphoto [**-i**] *schedule* [*drive*]

Description

fsphoto, in conjunction with **fsave**(ADM), provides a semi-automated interface to **xbackup**(ADM) and **cpio**(C) for backing-up filesystems (**xbackup** can only be used to back up XENIX fileystems). A human operator is required to mount and dismount tapes or floppies at the appropriate times, so some interaction is necessary, but all such interaction is kept to a minimum to reduce the potential for human error.

The selection and timing of backups for all filesystems is governed by the **schedule**(ADM) database. The system administrator must set up this file, and make arrangements to run **fsphoto** on the implicitly defined schedule (normally once per weekday). **fsphoto** can be invoked most easily from the **sysadmsh**(ADM). **fsphoto** interprets *schedule*, and for each filesystem that should be backed-up on that day, runs **fsave** to interact with the operator and backup the filesystem without error.

The optional argument *drive* specifies the magtape or floppy device to use; the default is specified in the *schedule* file.

Backups may be postponed (via **fsave**) or interrupted. The resulting "partial" backups are automatically resumed the next time **fsphoto** is run: any missed filesystems are backed-up as if the original backup had not been delayed. The **-i** flag ignores any pending partial backups.

If there is a pending partial backup, the normally scheduled backups are not done. This means that if a partial backup is resumed, and the normally scheduled backups are to be done, **fsphoto** must be run twice.

You must be the super user to use this program.

Files

/usr/lib/sysadmin/schedule	Database describing which filesystems are to be backed-up when, and at what dump level.
/dev/tty	Source of interactive input.
/usr/lib/sysadmin/past	Record of filesystems successfully backed-up in the pending partial backup.
/tmp/backup$$	Temporary file for recording successfully backed-up filesystems.

See also

basename(C), **fsave**(ADM), **schedule**(ADM), **xbackup**(ADM),

Diagnostics

fsphoto complains of syntax errors in *schedule*, and exits with a status of 1.

fsphoto complains about illegal or incorrect arguments, and exits with a status of 1.

An interrupt will cause an exit status of 2.

Notes

If a ***drive*** is explicitly given, the "raw" (*/dev/r**) form of the device should be used.

Value added

fsphoto is an extension to AT&T System V developed by The Santa Cruz Operation, Inc.

fsstat

report file system status

Syntax

/etc/fsstat ***special_file***

Description

The **fsstat** command reports on the status of the file system on ***special_file***. During startup, this command is used to determine if the file system needs checking before it is mounted. The **fsstat** command succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

See also

filesystem(FP)

Diagnostics

The command has the following exit codes:

0 the file system is not mounted and appears okay, (except for root where 0 means mounted and okay)

1 the file system is not mounted and needs to be checked

2 the file system is mounted

3 the command failed

This command does not work on DOS filesystems.

The directory */etc/fscmd.d/TYPE* contains programs for each file system type, **fsstat** invokes the appropriate binary.

fstyp

determine file system identifier

Syntax

/etc/fstyp *device*

Description

The **fstyp** command allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by **mount**(S) and sometimes by **mount**(ADM) to mount file systems of different types.

fstyp runs the programs in */etc/fscmd.d/TYPE* in alphabetical order, passing *device* as an argument; if any program succeeds, its filesystem type identifier is printed and **fstyp** exits immediately. If no program succeeds, **fstyp** prints

```
Unknown_fstyp
```

to indicate failure.

See also

mount(ADM), **mount**(S), **sysfs**(S)

fuser

identify processes using a file or filesystem

Syntax

/etc/fuser [-ku] ***files*** **|** ***filesystems*** **[-] [[-ku]** ***files*** **|** ***filesystems*** **]**

Description

The **fuser** command outputs the process IDs of the processes that are using the ***files*** or local ***filesystems*** specified as arguments. (**fuser** does not work on remote (NFS) filesystems.) Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as

1. its current directory, the code is **c**;
2. the parent of its current directory (only when the file is being used by the system), the code is **p**; or
3. its root directory, the code is **r**.

For block-special devices with mounted filesystems, all processes using any file on that device are listed. For all other types of files (text files, executables, directories, devices, etc.), only the processes using that file are reported.

The following options may be used with **fuser**:

-u the user login name, in parentheses, also follows the process ID.

-k the **SIGKILL** signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately (see **kill**(S)).

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash (-) cancels the options currently in force; then, the new set of options applies to the next group of files.

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.

You cannot list processes using a particular file from a remote filesystem mounted on your machine. You can only use the filesystem name as an argument.

Any user with permission to read */dev/kmem* and */dev/mem* can use **fuser**. Only the super user can terminate another user's process

Files

/unix	for system name list
/dev/kmem	for system image
/dev/mem	also for system image

See also

kill(S), **mount**(ADM), **ps**(C), **signal**(S)

fwtmp, wtmpfix

manipulate connect accounting records

Syntax

/usr/lib/acct/fwtmp [**-ic**]

/usr/lib/acct/wtmpfix [*files*]

Description

wtmpfix - corrects wtmp files

fwtmp

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

The argument **-ic** is used to denote that input is in ASCII form, and output is to be written 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 – (the ⟨dash⟩ key) can be used in place of ***files*** to indicate the standard input. If time/date corrections are not performed, **acctcon**(ADM) will fault 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** will check the validity of the "name" field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, **wtmpfix** reduces the chance that **acctcon**(ADM) will fail when processing connect accounting records.

File

/etc/wtmp

See also

acct(ADM), **acct**(FP), **acct**(S), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acctmerg**(ADM), **acctprc**(ADM), **acctsh**(ADM), **ed**(C), **runacct**(ADM), **utmp**(F)

Standards conformance

fwtmp and **wtmpfix** are conformant with:

AT&T SVID Issue 2.

goodpw

check a password for non-obviousness

Syntax

goodpw [-absm] [-d *file* **] [-r** *reason* **] [-MR** *expr* **]**

Description

goodpw reads a proposed password from the standard input and applies a variety of heuristic checks intended to spot poor password choices. These checks can include checking against user names, English words, and too short or too simple passwords. The checks applied depend on the settings in */etc/default/goodpw*, the file specified by the **-d** option, and the expressions specified by the **-M** and **-R** options.

The first line read from the standard input is taken to be the proposed password. A list of "canonical forms" is then generated; the canonical form is the password without any non-letters and with all letters converted to uppercase. The list always includes the canonical form of the password and may, depending on the settings in */etc/default/goodpw*, also contain left or right "rotations" of the canonical form. A rotation to the left is a shifting of the second through last character one position to the left, with the first character becoming the last; a rotation to the right is similar but in the opposite direction. The canonical list so generated is what most of the checks are applied against; if any (possibly rotated) canonical form in the list fails a check, the password is considered inadvisable and is rejected.

Any subsequent lines read from the standard input are taken to be a "stop list" of disallowed passwords. Each line in the stop list is reduced to its canonical form and checked against the canonical list; if there is a match, the password is rejected.

When a password is rejected, the reason is written to the standard error output and **goodpw** exits with a non-zero status. If a password passes all checks and hence is not rejected, no message is issued and **goodpw** exits with a zero status.

The **-s** and **-m** options modify this behavior: If **-s** is specified, no reason is issued. If **-m** is specified, then:

1. the stop list terminates with an empty line,
2. one line is written to the standard output indicating the acceptance or rejection of the password, and
3. the entire procedure is repeated using a new password and stop list read from the standard input.

This allows one **goodpw** process to check multiple passwords. The line written by **goodpw** to the standard output if **-m** is specified is one of:

g The password passed all checks and seems to be acceptable.

r*reason* The password was rejected for the indicated ***reason***.

e*error* The indicated system ***error*** occurred and it cannot be determined whether or not the password is acceptable.

If **-s** was specified, then no ***reason*** or ***error*** is written after a **r** or **e**, respectively.

The other options are:

-a Use American spelling (default).

-b Use British spelling.

-r*reason* Specify the message to be issued in case the proposed password matches one of those in the stop-list. The default ***reason*** is "same as previous password".

-d*file* Read the named ***file*** (which should be in the same format as */etc/default/goodpw*) and apply the various checks specified.

-M*expr* The password must match ***expr***, a boolean combination of regular expressions. If the first character of ***expr*** is a slash ("/") and a regular file by that name exists, the contents of that file are used as the expression. (If the file cannot be read, an error results.)

-R*expr* The password must not match ***expr***.

The boolean combination of regular expressions (***expr***) is built from the following operations:

expr1 **&** ***expr2*** True if, and only if, both expressions ***expr1*** and ***expr2*** are true. If ***expr1*** is not true, ***expr2*** is not evaluated.

expr1 **|** ***expr2*** True if either (or both) of ***expr1*** or ***expr2*** is true. If ***expr1*** is true, ***expr2*** is not evaluated.

expr1 **^** ***expr2*** True if exactly one of ***expr1*** and ***expr2*** are true. Both ***expr1*** and ***expr2*** are always evaluated.

! ***expr*** True if ***expr*** is not true; ***expr*** is always evaluated.

(***expr*** **)** True if, and only if, ***expr*** is true; ***expr*** is always evaluated.

/re/ True if, and only if, regular expression ***re*** matches the password. Any regular expression defined by **regcmp**(S) is understood; substrings defined by **(...)$***n* are placed in "accumulator" ***n***.

$*n* **~** ***/re/*** True if, and only if, accumulator ***n*** (**0-9**, or *****) matches regular expression ***re***; accumulator star (*****) is the entire password.

$*n* !~ /*re*/ True if, and only if, accumulator ***n*** is not matched by regular expression ***re***.

The possible **goodpw** checks, their control settings in */etc/default/goodpw*, and default values are:

MATCH=*/usr/lib/goodpw/match*
An expression (***expr***), or the name of file containing an expression, that the password must match. This expression also may be specified by the **-M** option.

REJECT=*/usr/lib/goodpw/reject*
An expression, or the name of a file containing an expression, that the password must not match. This expression may also be specified by the **-R** option.

LEFT_ROTATIONS=UNIQUE
How left rotations of the canonical form of the password should be treated: **NO** - ignored; **YES** - considered in other checks (that is, added to the canonical list) and may contain duplications; **UNIQUE** - considered in other checks but must not contain any duplications.

RIGHT_ROTATIONS=UNIQUE
Similarly for right rotations.

BOTH_ROTATIONS=UNIQUE
Similarly for rotations in both directions taken together.

AVOID_USERS=YES
Should the canonical list be checked against user login names and real names, obtained from */etc/passwd*?

AVOID_GROUPS=YES
Should the canonical list be checked against group names and group member lists, obtained from */etc/group*?

AVOID_MACHINES=YES
Should the canonical list be checked against machine names obtained from a number of files, including */etc/systemid* and */usr/lib/mail/top*?

AVOID_ALIASES=YES
Should the canonical list be checked against mail aliases obtained from */usr/lib/mail/aliases*?

AVOID_WORDS=YES
Should the canonical list be checked for properly spelled English words?

BRITISH=NO
Should **spell** use American or British spelling? Which spelling to use may be specified by the **-a** and **-b** options.

SITECHECKS=NO
The name of a program to run to provide additional checking. The program is run with no arguments. Passed to the program on its standard input, on separate lines, is first the actual proposed password and then the canonical list. If the program exits with a non-zero status, the password is rejected.

SITEREASON=Rejected by site-specific check(s)
The reason to give when the **SITECHECKS** program rejects the password. The values for the default settings can be adjusted to reflect the local system's security concerns. If */etc/default/goodpw* does not exist or cannot be read, the above default values are used (except for **MATCH** and **REJECT**). The default **MATCH** expression matches any password which:

1. Contains lower-case letters, upper-case letters, and digits, and whose length is four or more characters; *or*,

2. Contains no lower-case letters, no upper-case letters, and no digits, and whose length is four or more characters; *or*,

3. Contains both lower-case letters and digits, or both upper-case letters and digits, or both lower- and upper-case letters, and whose length is five or more characters; *or*,

4. Contains nothing but lower-case letters, and whose length is six or more characters; *or*,

5. Contains nothing but upper-case letters, and whose length is six or more characters.

The default **REJECT** expression is:

/[Ss][Cc][Oo]/ | /[Xx][Ee][Nn][Ii][Xx]/

which matches any password that contains either "SCO" or "XENIX" regardless of case.

Files

/usr/lib/goodpw/match	Expression that all passwords must match; by default, it contains the above-described **MATCH** expression.
/usr/lib/goodpw/reject	Expression that no passwords should match; by default, it contains the above-described **REJECT** expression.

See also

default(F), **group**(F), **passwd**(C), **passwd**(FP), **regex**(S), **systemid**(F), **spell**(CT)

Notes

Not all valid English words are known to **spell**, and hence some English words are considered acceptable as passwords.

The maximum length of a password is 100 characters, none of which may be an ASCII **NUL** or **LF** (newline).

Empty passwords are always rejected.

Value added

goodpw is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

graph

draw a graph

Syntax

graph [*options*]

Description

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

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

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

-a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **-x**).

-b Break (disconnect) the graph after each label in the input.

-c Character string given by next argument is default label for each point.

-g Next argument is grid style: 0 no grid, 1 frame with ticks; 2 full grid (default).

-l Next argument is label for graph.

-m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (for example, the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

-s Save screen, do not erase before plotting.

-x [l] If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.

-y [l] Similarly for y.

-h Next argument is fraction of space for height.

-w Similarly for width.

-r Next argument is fraction of space to move right before plotting.

-u Similarly to move up before plotting.

-t Transpose horizontal and vertical axes. (Option **-x** now applies to the vertical axis.) A legend indicating grid range is produced with a grid unless the *-s* option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

See also

spline(C), **tplot**(ADM)

Notes

The **graph** command stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

grpck

check group file

Syntax

grpck [*file*]

Description

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(F), **passwd**(FP), **pwck**(ADM)

Diagnostics

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

Value added

grpck is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

haltsys, reboot

close out filesystems and shut down the system

Syntax

/etc/haltsys [-d]

/etc/reboot

Description

haltsys - shuts down the system

reboot - shuts down the system and reboots

The **haltsys** utility performs a **uadmin**( ) system call (see **uadmin**(S)) to flush out pending disk I/O, mark the filesystems as clean, and halt the processor. **haltsys** takes effect immediately, so user processes should be killed beforehand. **shutdown**(ADM) is recommended for normal system shutdown, since it warns users, terminates processes, then calls **haltsys**. Use **haltsys** directly only if you cannot run **shutdown**; for example, because of some system problem.

haltsys displays a prompt indicating that the system has been shut down and can be rebooted or powered down. If the **-d** option is used, the system will remain down and you are not given the option to reboot.

The **reboot** command performs the same function as **haltsys**, except that the system is rebooted automatically afterwards.

Only the super user can execute **haltsys** or **reboot**.

Note

haltsys locks hard disk heads.

See also

shutdn(S), **uadmin**(S), **shutdown**(ADM)

Value added

haltsys is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

reboot was developed at the University of California, Berkeley, and is used with permission.

idaddld

add or remove line disciplines from kernel configuration files

Syntax

/etc/conf/bin/idaddld [-a *prefix routine1 ... routine8*] [-dc *prefix*]

Description

idaddld is used to add or remove line discipline declarations from kernel configuration files. If no arguments are given, **idaddld** enters an interactive mode. In this mode the user can add, delete or view the current configuration. If a change is specified then the user is prompted to relink the kernel. If arguments are given on the command line, **idaddld** enters a non-interactive mode, executing the specified command silently. It is the responsibility of the calling program to insure that the kernel is relinked to effect the desired changes.

Options

The following options are available from the command line.

-a *prefix routine1 ... routine8*
: Add a line discipline to configuration files. ***prefix*** is a tag used to identify the line discipline for future inquiries or removal. For example, the terminal line discipline uses the prefix *tty*. ***routine1*** through ***routine8*** define the list of line discipline routines. There must be eight routines with the keyword "nulldev" used as a placeholder. The order of the routines is critical. They must be ordered as follows:

  open close read write ioctl rxint txint modemint

-d*prefix*
: Remove the line discipline whose identifier matches ***prefix***.

-c*prefix*
: Scan the line discipline switch table for an entry which matches ***prefix***. The program will exit with a return status 0 if a match is found and 1 otherwise.

Notes

When a line discipline is added, it is appended to the current switch table configuration.

Value added

idaddld is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

idbuild, idmkenv, idmkunix, idconfig, idvidi, idscsi

build new UNIX system kernel

Syntax

/etc/conf/bin/idbuild

Description

idconfig - configure UNIX system kernel

The **idbuild** script builds a new UNIX system kernel using the current system configuration in */etc/conf*. The **idconfig** script builds the system kernel configuration files. Kernel reconfigurations are usually performed after a device driver is installed, or system tunable parameters are modified. The script uses the shell variable **$ROOT** from the user's environment as its starting path. Except for the special case of kernel development in a non-root source tree, the shell variable **$ROOT** should always be set to null or to " / ". **idbuild** exits with a return code of zero on success and non-zero on failure.

Building a new UNIX system image consists of generating new system configuration files, then link-editing the kernel and device driver object modules in the */etc/conf/pack.d* object tree. This is done by **idbuild** by calling the following commands:

/etc/conf/bin/idconfig To build kernel configuration files.

/etc/conf/bin/idmkunix To process the configuration files and link-edit a new UNIX system image.

The **-p** option to **idbuild** specifies that temporary files created during the build should not be deleted. This results in object modules and C source modules remaining on the system. These modules can be used as an aid to debugging. Temporary files created during earlier builds will, however, be removed.

The system configuration files are built by processing the Master and System files representing device driver and tunable parameter specifications. The files */etc/conf/cf.d/mdevice*, and */etc/conf/cf.d/mtune* represent the Master information. The file */etc/conf/cf.d/stune*, and the files specified in */etc/conf/sdevice.d/** represent the System information. The kernel also has file system type information defined in the files specified by */etc/conf/sfsys.d/** and */etc/conf/mfsys.d/**.

idvidi and **idscsi** read the video driver and SCSI driver configurations, respectively.

idconfig reads the system configuration files and reports any conflicts and errors.

idmkunix links the necessary modules to create the new kernel.

Once a new UNIX system kernel has been configured and linked, **idmkenv** is invoked to back up the current */unix* and replace it with the new kernel, and rebuild the kernel environment.

Diagnostics

Since **idbuild** calls other system commands to accomplish system reconfiguration and link editing, it will report all errors encountered by those commands, then clean up intermediate files created in the process. In general, the exit value 1 indicates an error was encountered by **idbuild.**

The errors encountered fall into the following categories:

Master file error messages.
System file error messages.
Tunable file error messages.
Compiler and Link-editor error messages.

All error messages are designed to be self-explanatory.

See also

configure(ADM), **idinstall**(ADM), **idtune**(ADM), **mdevice**(F), **mfsys**(FP), **mtune**(F), **sdevice**(F), **sfsys**(FP), **stune**(F)

idcheck

return selected information about system configuration

Syntax

/etc/conf/bin/idcheck

Description

This command returns selected information about the system configuration. It is useful in add-on device Driver Software Package (DSP) installation scripts to determine if a particular device driver has already been installed, or to verify that a particular interrupt vector, I/O address or other selectable parameter is in fact available for use.

To check whether a vector is in use, use **vectorsinuse**(ADM) rather than **idcheck**. **idcheck -v** *vector* may be used to determine whether a vector is shareable. (This option returns the type field (as defined in the **sdevice**(F) reference page) corresponding to the given *vector*.)

The various forms are:

idcheck -p *device-name* **[-i** *dir***] [-r]**

idcheck -v *vector* **[-i** *dir***] [-r]**

idcheck -d *dma-channel* **[-i** *dir***] [-r]**

idcheck -a -l *lower_address* **-u** *upper_address* **[-i** *dir***] [-r]**

idcheck -c -l *lower_address* **-u** *upper_address* **[-i** *dir***] [-r]**

This command scans the System and Master modules and exits with the following status:

100 if an error occurs.

0 if no conflict exists.

a positive number greater than 0 and less than 100 if a conflict exists.

The command line options are:

-r Reports device name of any conflicting device on *stdout*.

-p *device-name* This option checks for the existence of four different components of the DSP. The exit code is the addition of the return codes from the four checks.

Add 1 to the exit code if the DSP directory under */etc/conf/pack.d* exists.

Add 2 if the Master module has been installed.

Add 4 if the System module has been installed.

Add 8 if the Kernel was built with the System module.

Add 16 if a **Driver.o** is part of the DSP (vs. a *stubs.c* file).

-v ***vector*** This option returns the value of the "type" field in the *mdevice* file for the device that is already using the vector. Do not use this option to check whether a vector is in use; some devices use interrupt 0, and this will cause **idcheck** to return the same result as a free vector. To check whether a vector is available, use **vectorsinuse**(ADM) instead.

-d ***dma-channel*** Returns 1 if the dma channel specified is being used.

-a This option checks whether the IOA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **-l** and **-u** options). The exit code is based on the first conflicting device found.

The exit code is 1 if the IOA region overlaps with another device.

The exit code is 2 if the IOA region overlaps with another device and that device has the 'O' option specified in the "type" field of the Master module. The 'O' option permits a driver to overlap the IOA region of another driver.

-c Returns 1 if the CMA region bounded by "lower" and "upper" conflict with another DSP ("lower" and "upper" are specified with the **-l** and **-u** options).

-l ***address*** Lower bound of address range specified in hex. The leading **0x** should not be included.

-u ***address*** Upper bound of address range specified in hex. The leading **0x** should not be included.

-i ***dir*** Specifies the directory in which the ID files *sdevice* and *mdevice* reside. The default directory is */etc/conf/cf.d*.

Diagnostics

There are no error messages or checks for valid arguments to options. **idcheck** interprets these arguments using the rules of **scanf**(S) and queries the *sdevice* and *mdevice* files. For example, if a letter is used in the place of a digit, **scanf**(S) will translate the letter to 0. **idcheck** will then use this value in its query.

See also

idinstall(ADM), **mdevice**(F), **sdevice**(F)

idinstall

add, delete, update, or get device driver configuration data

Syntax

/etc/conf/bin/idinstall -[adu] [-e] [-k] [-msnirhclopt] ***dev_name***

/etc/conf/bin/idinstall -g -[snirhclpt] ***dev_name***

Description

The **idinstall** command is called by a Driver Software Package (DSP) Install script or Remove script to Add (**-a**), Delete (**-d**), Update (**-u**), or Get (**-g**) device driver configuration data. **idinstall** expects to find driver component files in the current directory. When components are installed or updated, they are moved or appended to files in the */etc/conf* directory and then deleted from the current directory unless the **-k** flag is used. The options for the command are as follows:

Action Specifiers:

- **-a** Add the DSP components
- **-d** Delete the DSP components
- **-u** Update the DSP components
- **-g** Get the DSP components (print to *stdout*, except Master)

Component Specifiers: (*)

- **-m** Master component
- **-s** System component
- **-o** Driver.o component
- **-p** Space.c component
- **-t** Stubs.c component
- **-n** Node (special file) component
- **-i** Inittab component
- **-r** Device Initialization (rc) component
- **-h** Device shutdown (sd) component
- **-c** Mfsys component: file system type config (Master) data
- **-l** Sfsys component: file system type local (System) data

(*) If no component is specified, the default is all except for the **-g** option where a single component must be specified explicitly.

Miscellaneous:

-e Disable free disk space check

-k Keep files (do not remove from current directory) on add or update.

In the simplest case of installing a new DSP, the command syntax used by the DSP's installation script should be **idinstall -a *dev_name***. In this case the command will require and install a Driver.o, Master and System entry, and optionally install the Space.c, Stubs.c, Node, Init, Rc, Shutdown, Mfsys, and Sfsys components if those modules are present in the current directory.

The *Driver.o*, *Space.c*, and *Stubs.c* files are moved to a directory in */etc/conf/pack.d*, and end up in */etc/conf/pack.d/dev_name*. The ***dev_name*** is passed as an argument, which is used as the directory name. The remaining components are stored in the corresponding directories under */etc/conf* in a file whose name is ***dev_name***. For example, the Node file would be moved to */etc/conf/node.d/dev_name*.

The **idinstall -m** usage provides an interface to the **idmaster** command which will add, delete, and update **mdevice** file entries using a Master file from the local directory. An interface is provided here so that driver writers have a consistent interface to install any DSP component.

As stated above, driver writers will generally use only the **idinstall -a *dev_name*** form of the command. Other options of **idinstall** are provided to allow an Update DSP (that is, one that replaces an existing device driver component) to be installed, and to support installation of multiple controller boards of the same type.

If the call to **idinstall** uses the **-u** (update) option, it will:

overlay the files of the old DSP with the files of the new DSP.

invoke the **idmaster** command with the 'update' option if a Master module is part of the new DSP.

idinstall also does a verification that enough free disk space is available to start the reconfiguration process. This is done by calling the **idspace** command. **idinstall** will fail if insufficient space exists, and exit with a non-zero return code. The **-e** option bypasses this check.

idinstall makes a record of the last device installed in a file (*/etc/.last_dev_add*), and saves all removed files from the last delete operation in a directory (*/etc/.last_dev_del*). These files are recovered by **/etc/conf/bin/idmkenv** whenever it is determined that a system reconfiguration was aborted due to a power failure or unexpected system reboot.

Diagnostics

An exit value of zero indicates success. If an error was encountered, **idinstall** will exit with a non-zero value, and report an error message. All error messages are designed to be self-explanatory. Typical error messages that can be generated by **idinstall** are as follows:

```
Device package already exists.Cannot make the driver package directory.
Cannot remove driver package directory.
Local directory does not contain a Driver object (Driver.o) file.
Local directory does not contain a Master file.
Local directory does not contain a System file.
Cannot remove driver entry.
```

See also

idspace(ADM), **idcheck**(ADM), **mdevice**(F), **sdevice**(F)

idleout

log out idle users

Syntax

idleout [***minutes*** | ***hours:minutes***]

Description

The **idleout** command monitors line activity and logs out users whose terminal remains idle longer than a specified period of time. Minutes are assumed; if a colon appears in the number, hours are assumed.

The utility uses a default file, */etc/default/idleout*, to indicate the interval a user's terminal may remain idle before being logged out. This file has one entry:

IDLETIME=*time*

The time format is identical to that used on the command line. The time specified in the default file is overridden by **idletime** if **idletime** is specified on the command line. Note that, if **idletime** is zero, no monitoring takes place and idle users are not logged out. You can either **run idleout** from the command line, or, to have continuous coverage, you must add the program name in */etc/rc2.d/S88USRDEFINE* to see to it that the program is **run** each time the system is rebooted.

Files

/etc/default/idleout
/etc/utmp
/etc/wtmp

See also

getut(S), **kill**(S), **who**(C)

Value added

idleout is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

idmkinit

read files containing inittab specifications

Syntax

/etc/conf/bin/idmkinit

Description

This command reads the files containing specifications of */etc/inittab* entries from */etc/conf/init.d* and constructs a new *inittab* file in */etc/conf/cf.d*. It returns 0 on success and a positive number on error.

The files in */etc/conf/init.d* are copies of the Init modules in device Driver Software Packages (DSP). There is at most one Init file per DSP. Each file contains one line for each *inittab* entry to be installed. There may be multiple lines (that is, multiple *inittab* entries) per file. An *inittab* entry has the form (the "id" field is often called the *tag*):

id:rstate:action:process

The Init module entry must have one of the following forms:

action:process

rstate:action:process

id:rstate:action:process

When **idmkinit** encounters an entry of the first type, a valid "id" field will be generated, and an "rstate" field of 2 (indicating run on init state 2) will be generated. When an entry of the second type is encountered, only the "id" field is prefixed. An entry of the third type is incorporated into the new *inittab* unchanged.

Since add-on *inittab* entries specify init state 2 for their "rstate" field most often, an entry of the first type should almost always be used. An entry of the second type may be specified if you need to specify other than state 2. DSPs should avoid specifying the "id" field as in the third entry since other add-on applications or DSPs may have already used the "id" value you have chosen. The **/etc/init** program will encounter serious errors if one or more *inittab* entries contain the same "id" field.

idmkinit determines which of the three forms above is being used for the entry by requiring each entry to have a valid ***action*** keyword. Valid ***action*** values are as follows:

off
respawn
ondemand
once
wait
boot
bootwait
powerfail
powerwait
initdefault
sysinit

See *inittab*(F) for a description of the action keywords.

The **idmkinit** command is called automatically upon entering init state 2 on the next system reboot after a kernel reconfiguration to establish the correct */etc/inittab* for the running kernel. **idmkinit** can be called as a user level command to test modification of *inittab* before a DSP is actually built. It is also useful in installation scripts that do not reconfigure the kernel but need to create *inittab* entries. In this case, the *inittab* generated by **idmkinit** must be copied to */etc/inittab*, and a **telinit**(M) command must be run to make the new entry take effect.

The command line options are

-o *directory* *inittab* will be created in the directory specified rather than */etc/conf/cf.d*.

-i *directory* The ID file *init.base*, which normally resides in */etc/conf/cf.d*, can be found in the directory specified.

-e *directory* The Init modules that are usually in */etc/conf/init.d* can be found in the directory specified.

Diagnostics

An exit value of zero indicates success. If an error was encountered, **idmkinit** will exit with a non-zero value and report an error message. All error messages are designed to be self-explanatory.

See also

idbuild(ADM), **idinstall**(ADM), **idmknod**(ADM), **init**(M), **inittab**(F)

idmknod

remove nodes and read specifications of nodes

Syntax

/etc/conf/bin/idmknod -o*directory* **-i** *directory* **-e***directory* **-s**

Description

This command performs the following functions:

- Removes the nodes for non-required devices (those that do not have an "r" in field 3 of the device's **mdevice** entry) from */dev*. Ordinary files will not be removed. If the */dev* directory contains subdirectories, those subdirectories will be traversed and nodes found for non-required devices will be removed as well. If empty subdirectories result due to the removal of nodes, the subdirectories are then removed.
- Reads the specifications of nodes given in the files contained in */etc/conf/node.d* and installs these nodes in */dev*. If the node specification defines a path containing subdirectories, the subdirectories will be made automatically.
- Returns 0 on success and a positive number on error.

idmknod is run automatically when **idbuild**(ADM) installs a newly built kernel as */unix*. **idmknod** can be called as a user level command to test modification of the */dev* directory before a DSP is actually built. It is also useful in installation scripts that do not reconfigure the kernel, but need to create */dev* entries.

The files in */etc/conf/node.d* are copies of the *Node* modules installed by device Driver Software Packages (DSP). There is at most one file per DSP. Each file contains one line for each node that is to be installed. The format of each line is:

```
name  type  minor  [ owner  group  mode ]
```

Name of device entry (field 1) in the *mdevice* file (The mdevice entry will be the line installed by the DSP from its *Master* module). This field must be from 1 to 8 characters in length. The first character must be a letter. The others may be letters, digits, or underscores.

Name of node to be inserted in */dev*. The first character must be a letter. The others may be letters, digits, or underscores. This field can be a path relative to */dev*, and **idmknod** will create subdirectories as needed.

The character "b" or "c". A "b" indicates that the node is a 'block' type device and "c" indicates 'character' type device.

Minor device number. This value must be between 0 and 255, except for drivers that have duplicate entries with the 'M' characteristic in *mdevice*(F). In such a case, the upper limit is determined by adding 255 to

the maximum **OFFSET** value defined in *mdevice*(F) for the particular driver; that is, upper limit = 255 + maximum **OFFSET**. If this field is a non-numeric, it is assumed to be a request for a streams clone device node, and **idmknod** will set the minor number to the value of the major number of the device specified.

"owner". (Optional field). Contains the name of the owner of the device node.

"group". (Optional field). Contains the name of the group to which the device node belongs.

"mode". (Optional field). Contains the mode of the device node, as an octal number (see **chmod**(C) for details).

Some example node file entries are as follows:

```
asy  tty00  c  1
```

makes */dev/tty00* for device 'asy' using minor device 1.

```
qt  rmt/c0s0  c  4
```

makes */dev/rmt/c0s0* for device 'qt' using minor device 4.

```
clone  net/nau/clone  c  nau
```

makes */dev/net/nau/clone* for device 'clone'. The minor device number is set to the major device number of device 'nau'.

The command line options are:

-o *directory* Nodes will be installed in the directory specified rather than */dev*.

-i *directory* The file *mdevice* which normally resides in */etc/conf/cf.d* can be found in the directory specified.

-e *directory* The Node modules that normally reside in */etc/conf/node.d* can be found in the directory specified.

-s Suppress removing nodes (just add new nodes).

Diagnostics

An exit value of zero indicates success. If an error was encountered due to a syntax or format error in a nodes entry, an advisory message will be printed to *stdout* and the command will continue. If a serious error is encountered (i.e., a required file cannot be found), **idmknod** will exit with a non-zero value and report an error message. All error messages are designed to be self-explanatory.

See also

idinstall(ADM), **idmkinit**(ADM), **mdevice**(F), **sdevice**(F)

idspace

investigate free space

Syntax

/etc/conf/bin/idspace [-i *inodes*] [-r *blocks*] [-u *blocks*] [-t *blocks*]

Description

This command investigates free space in /, */usr*, and */tmp* filesystems to determine whether sufficient disk blocks and inodes exist in each of potentially 3 filesystems. The default tests that **idspace** performs are as follows:

- Verify that the root filesystem (/) has 400 blocks more than the size of the current */unix*. This verifies that a device driver being added to the current */unix* can be built and placed in the root directory. A check is also made to insure that 100 inodes exist in the root directory.
- Determine whether a */usr* filesystem exists. If it does exist, a test is made that 400 free blocks and 100 inodes are available in that filesystem. If the filesystem does not exist an error is returned, but since files used by the reconfiguration process will be created in the root file system, space requirements are covered by the test above.
- Determine whether a */tmp* filesystem exists. If it does exist, a test is made that 400 free blocks and 100 inodes are available in that filesystem. If the filesystem does not exist an error is returned, but since files used by the reconfiguration process will be created in the root file system, space requirements are covered by the test above.

The command line options are:

-i *inodes* This option overrides the default test for 100 inode in all of the **idspace** checks.

-r *blocks* This option overrides the default test for */unix* size + 400 blocks when checking the root (/) filesystem. When the **-r** option is used, the */usr* and */tmp* filesystems are not tested unless explicitly specified.

-u *blocks* This option overrides the default test for 400 blocks when checking the */usr* filesystem. When the **-u** option is used, the root (/) and */tmp* filesystems are not tested unless explicitly specified. If */usr* is not a separate filesystem, an error is reported.

-t *blocks* This option overrides the default test for 400 blocks when checking the */tmp* filesystem. When the **-t** option is used, the root (/) and */usr* filesystems are not tested unless explicitly specified. If */tmp* is not a separate filesystem, an error is reported.

Diagnostics

An exit value of zero indicates success. If insufficient space exists in a filesystem or an error was encountered due to a syntax or format error, **idspace** will report a message. All error messages are designed to be self-explanatory. The specific exit values are as follows:

0 success.

1 command syntax error, or needed file does not exist.

2 filesystem has insufficient space or inodes.

3 requested filesystem does not exist (**-u** and **-t** options only).

See also

idbuild(ADM), **idinstall**(ADM)

idtune

attempt to set value of a tunable parameter

Syntax

/etc/conf/bin/idtune [-f | -m] ***name value***

Description

This script attempts to set the value of a tunable parameter. The tunable parameter to be changed is indicated by ***name***. The desired value for the tunable parameter is ***value***.

If there is already a value for this parameter (in the *stune* file), the user will normally be asked to confirm the change with the following message:

```
Tunable Parameter name is currently set to old_value.
Is it OK to change it to value? (y/n)
```

If the user answers **y**, the change will be made. Otherwise, the tunable parameter will not be changed, and the following message will be displayed:

```
name left at old_value.
```

However, if the **-f** (force) option is used, the change will always be made and no messages will ever be given.

If the **-m** (minimum) option is used and there is an existing value which is greater than the desired value, no change will be made and no message will be given.

If system tunable parameters are being modified as part of a device driver or application add-on package, it may not be desirable to prompt the user with the above question. The add-on package Install script may chose to override the existing value using the **-f** or **-m** options. However, care must be taken not to invalidate a tunable parameter modified earlier by the user or another add-on package or to set a value outside the minimum and maximum values allowed by the *mtune* file.

In order for the change in parameter to become effective, the UNIX system kernel must be rebuilt and the system rebooted.

Diagnostics

The exit status will be non-zero if errors are encountered.

See also

idbuild(ADM), **mtune**(F), **stune**(F)

infocmp

compare or print out terminfo descriptions

Syntax

infocmp [-d] [-c] [-n] [-I] [-L] [-C] [-r] [-u] [-s d | i | l | c] [-v] [-V] [-1] [-w *width*] [-A *directory*] [-B *directory*] [*termname* ...]

Description

The **infocmp** command can be used to compare a binary *terminfo*(F) entry with other terminfo entries, rewrite a *terminfo* description to take advantage of the **use=** terminfo field, or print out a *terminfo* description from the binary file (*term*(F)) in a variety of formats. In all cases, the Boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default options

If no options are specified and zero or one ***termnames*** are specified, the **-I** option will be assumed. If more than one ***termname*** is specified, the **-d** option will be assumed.

Comparison options [-d] [-c] [-n]

The **infocmp** command compares the *terminfo* description of the first terminal ***termname*** with each of the descriptions given by the entries for the other terminal's ***termnames***. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **-1** for integer variables, and **NULL** for string variables.

-d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using **infocmp** will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

-c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **-u** option is worth using.

-n produce a list of each capability that is in neither entry. If no ***termnames*** are given, the environment variable **TERM** will be used for both of the ***termnames***. This can be used as a quick check to see if anything was left out of the description.

Source listing options [-I] [-L] [-C] [-r]

The **-I**, **-L**, and **-C** options will produce a source listing for each terminal named.

-I use the *terminfo* names

-L use the long C variable name listed in <*term.h*>

-C use the *termcap* names

-r when using **-C**, put out all capabilities in *termcap* form

If no ***termnames*** are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the **-C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. **infocmp** will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*, but which are derivable from other *terminfo* variables, will be output. Not all *terminfo* capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **-r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible; it is not always possible to convert a *terminfo* string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo* format will not necessarily reproduce the original *terminfo* source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	

Use= option [-u]

-u produce a *terminfo* source description of the first terminal ***termname*** which is relative to the sum of the descriptions given by the entries for the other terminals' ***termnames***. It does this by analyzing the differences between the first ***termname*** and the other ***termnames*** and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using **infocmp** will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first **termname**, but one of the other **termname** entries contains a value for it. A capability's value gets printed if the value in the first **termname** is not found in any of the other **termname** entries, or if the first of the other **termname** entries that has this capability gives a different value for the capability than that in the first **termname**.

The order of the other **termname** entries is significant. Since the terminfo compiler **tic**(C) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given. **infocmp** will flag any such inconsistencies between the other **termname** entries as they are found.

Alternatively, specifying a capability **after** a **use=** entry that contains that capability will cause the second specification to be ignored. Using **infocmp** to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. **infocmp** will flag any other **termname use=** fields that were not needed.

Other options [-s d | i | l | c] [-v] [-V] [-1] [-w width]

-s sort the fields within each type according to the argument below:

d leave fields in the order that they are stored in the *terminfo* database.

i sort by *terminfo* name.

l sort by the long C variable name.

c sort by the *termcap* name.

If no -s option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **-C** or the **-L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

-v print out tracing information on standard error as the program runs.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.

-w change the output to ***width*** characters.

Changing data bases [-A directory] [-B directory]

The location of the compiled *terminfo* database is taken from the environment variable **TERMINFO**. If the variable is not defined or the terminal is not found in that location, the system *terminfo* database, usually in */usr/lib/terminfo*, will be used. The options **-A** and **-B** may be used to override this location. The **-A** option will set **TERMINFO** for the first ***termname*** and the **-B** option will set **TERMINFO** for the other ***termnames***. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo* database for a comparison to be made.

Files

*/usr/lib/terminfo/?/**	compiled terminal description database

Diagnostics

`malloc is out of space!`
: There was not enough memory available to process all the terminal descriptions requested. Run **infocmp** several times, each time including a subset of the desired ***termnames***.

`use= order dependency found:`
: A value specified in one relative terminal specification was different from that in another relative terminal specification.

`'use=term' did not add anything to the description.`
: A relative terminal name did not contribute anything to the final description.

`must have at least two terminal names for a comparison to be done.`
: The **-u**, **-d**, and **-c** options require at least two terminal names.

See also

captoinfo(ADM), **curses**(S), **term**(F), **terminfo**(F), **tic**(C)

initcond

special security actions for init and getty

Syntax

/tcb/lib/initcond [init | getty] [*args* **...]**

Description

To save space in the **init**(M) and **getty**(M) programs, which are memory resident, the space intensive security actions are done in **initcond** as a subprocess of these programs.

If the argument is **init**, one of two actions may occur. First, no argument means that **sulogin**(ADM) should prompt for and verify a single user password if required by the System Default database. This is used for password checking before a single user shell. Second, if two other arguments are supplied, they are the terminal device name and the user name respectively of the session that just terminated. This information is reflected in both the Protected Password and Terminal Control databases.

If the argument is **getty**, and one additional argument is provided, it is the terminal to be invalidated before a login. **initcond** invalidates a terminal by setting a restricted set of permissions on the terminal device and by using **stopio**(S) to invalidate all open file descriptors that reference the terminal. These include synonym devices for the same physical device as listed in the device assignment database.

Files

/tcb/files/initcondlog	Log file for **init** and **getty** events
/etc/auth/system/ttys	Terminal Control database
/etc/auth/system/devassign	Device Assignment database

See also

getdvagent(S), **getprtcent**(S), **stopio**(S)

"Maintaining system security", chapter of the *System Administrator's Guide*

Value added

initcond is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

initscript

script that executes inittab commands

Syntax

/bin/sh /etc/initscript ***id rstate action process***

Description

/etc/initscript is the shell script used by **init**(M) to execute commands in */etc/inittab*.

Your system's **initscript** will look similiar to the following:

```
:
#       @(#) initscrp.dfl 22.2 90/02/23
#
#       Copyright (C) The Santa Cruz Operation, 1989, 1990.
#       This Module contains Proprietary Information of
#       The Santa Cruz Operation, and should be treated as Confidential.
#
# set up the default environment for command started by init.
#
# Usage: /etc/initscript id level action cmd
#

PATH=/bin:/usr/bin
export PATH

HZ=100
export HZ

[ -x /etc/TIMEZONE ] && . /etc/TIMEZONE

umask 027

eval exec "$4"
```

initscript sets the **PATH** variable, sets the **HZ** variable, checks for an **/etc/TIMEZONE** file and executes it, sets the **umask**, and runs the fourth argument (***process***), which is the process field from */etc/inittab*.

HZ is the hertz value, as described in **environ**(M).

/etc/TIMEZONE sets and exports the **TZ** variable, as described in **environ**(M).

Files

/etc/initscript
/etc/inittab

See also

environ(M), **init**(M), **inittab**(F), **sh**(C)

install

install commands

Syntax

/etc/install [**-c** *dira*] [**-f** *dirb*] [**-i**] [**-n** *dirc*] [**-m** *mode*] [**-u** *user*] [**-g** *group*] [**-o**] [**-s**] *file* [*dirx* ...]

Description

The **install** command is most commonly used in "makefiles" (see **make**(CP)) 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** will search 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 one or more directories (*dirx* ...) are specified after *file*, those directories will be 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 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 will be set to **755** and *bin*, respectively. If the file already exists, the mode and owner will be 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 any other options except **-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 will be set to **755** and *bin*, respectively. May be used alone or with any other options except **-c** and **-f**.

-m ***mode*** The mode of the new file is set to ***mode***. Only available to the super user.

-u ***user*** The owner of the new file is set to ***user***. Only available to the super user.

-g ***group*** The group id of the new file is set to ***group***. Only available to the super user.

-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 frequently used file such as **/bin/sh** or **/etc/getty**, where the existing file cannot be removed. May be used alone or with any other options except **-c**.

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

See also

make(CP)

installf

add a file to the software installation database

Syntax

installf [**-c** *class*] *pkginst pathname* [*ftype* [[*major minor*] [*mode owner group*]]

installf [**-c** *class*] *pkginst* **-**

installf -f [**-c** *class*] *pkginst*

Description

installf informs the system that a pathname not listed in the *pkgmap* file is being created or modified. It should be invoked before any file modifications have occurred.

When the second synopsis is used, the pathname descriptions will be read from standard input. These descriptions are the same as would be given in the first synopsis but the information is given in the form of a list. The descriptions should be in the form:

pathname ftype [[*major minor*] [*mode owner group*]]

After all files have been appropriately created and/or modified, **installf** should be invoked with the **-f** synopsis to indicate that installation is final. Links will be created at this time and, if attribute information for a pathname was not specified during the original invocation of **installf** or was not already stored on the system, the current attribute values for the pathname will be stored. Otherwise, **installf** verifies that attribute values match those given on the command line, making corrections as necessary. In all cases, the current content information is calculated and stored appropriately.

-c *class* Class to which installed objects should be associated. Default class is **none**.

pkginst Name of package instance with which the pathname should be associated.

pathname Pathname that is being created or modified.

ftype A one-character field that indicates the file type. Possible file types include:

- **f** a standard executable or data file
- **e** a file to be edited upon installation or removal
- **v** volatile file (one whose contents are expected to change)

d directory

x an exclusive directory

l linked file

p named pipe

c character special device

b block special device

s symbolic link

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (?) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked or symbolically linked files.

owner The owner of the file (for example, *bin* or *root*). The field is limited to 14 characters in length. A question mark (?) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked or symbolically linked files.

group The group to which the file belongs (for example, *bin* or *sys*). The field is limited to 14 characters in length. A question mark (?) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked or symbolically linked files.

-f Indicates that installation is complete. This option is used with the final invocation of **installf** (for all files of a given class).

Notes

When *ftype* is specified, all applicable fields, as shown below, must be defined:

ftype	Required fields
p, **x**, **d**, **f**, **v** or **e**	***mode***, ***owner*** and ***group***
c or **b**	***major***, ***minor***, ***mode***, ***owner*** and ***group***

The **installf** command will create directories, named pipes and special devices on the original invocation. Links are created when **installf** is invoked with the **-f** option to indicate installation is complete.

Links should be specified as ***path1=path2***. ***path1*** indicates the destination and ***path2*** indicates the source file.

For symbolically linked files, ***path2*** can be a relative pathname, such as ./ or ../. For example, if you enter a line such as

s /foo/bar/etc/mount=../etc/mount

path2 (*/foo/bar/etc/mount*) will be a symbolic link to *../etc/mount*.

Files installed with **installf** will be placed in the class **none**, unless a class is defined with the command. Subsequently, they will be removed when the associated package is deleted. If this file should not be deleted at the same time as the package, be certain to assign it to a class which is ignored at removal time. If special action is required for the file before removal, a class must be defined with the command and an appropriate class action script delivered with the package.

When classes are used, **installf** must be used as follows:

installf -c class1 ...
installf -f -c class1 ...
installf -c class2 ...
installf -f -c class2 ...

Example

The following example shows the use of **installf** invoked from an optional preinstall or postinstall script:

```
#create /dev/xt directory
#(needs to be done before drvinstall)
installf $PKGINST /dev/xt d 755 root sys ||
        exit 2
majno=`/usr/sbin/drvinstall -m /etc/master.d/xt
     -d $BASEDIR/data/xt.o -v1.0` ||
        exit 2
i=00
while [ $i -lt $limit ]
do
    for j in 0 1 2 3 4 5 6 7
    do
        echo /dev/xt$i$j c $majno `expr $i ? 8 + $j`
             644 root sys |
        echo /dev/xt$i$j=/dev/xt/$i$j
    done
    i=`expr $i + 1`
    [ $i -le 9 ] && i="0$i" #add leading zero
done | installf $PKGINST - || exit 2
# finalized installation, create links
installf -f $PKGINST || exit 2
```

See also

pkgadd(ADM), **pkgask**(ADM), **pkgchk**(ADM), **pkginfo**(ADM), **pkgmk**(ADM), **pkgparam**(ADM), **pkgproto**(ADM), **pkgtrans**(ADM), **pkgrm**(ADM), **removef**(ADM)

installpkg

install package

Syntax

installpkg

Description

The **installpkg** command is used to install an AT&T-style UNIX system software package.

You will have to be *root* to install certain packages successfully.

You will be prompted to insert the floppy disk that the installation package resides on. Everything else is automatic.

Notes

You must invoke **installpkg** on the console.

This command does not work on packages installed with **custom**(ADM).

See also

displaypkg(ADM), **removepkg**(ADM)

integrity

examine system files against the authentication database

Syntax

/tcb/bin/integrity [-v] [-e] [-m]

Description

integrity traverses the File Control database and compares each entry in turn to the real file in the file system. If the owner, group or permissions are different, an error message is output.

Wildcard entries in the File Control database are handled as follows. For file names, those file names that have /* as the last entry are treated as wild cards. Any file in the directory matches that entry, unless the specific file under consideration has its own (non-wildcard) entry in the database appearing before the wildcard entry. In this case, the file is ignored in the check because it would have been located previously. For owners (groups), if the File Control entry does not explicitly list an owner (group), all owners (groups) match correctly.

The **-v** option lists all files under consideration, even those that match. The **-e** option explains why discretionary checks fail and exactly what the discrepancy is.

Normally, (non-wildcard type) files in the File Control database that are missing from the file system are not reported. The **-m** option will override that default and report such missing files.

Notes

Only *root* can run this utility. **fixmog**(ADM) can be used to correct problems found by **integrity**.

Files

/etc/auth/system/files	File Control database
/etc/auth/system/default	System Defaults database

See also

authck(ADM), **fixmog**(ADM), **getprfient**(S), **stat**(S)

"Maintaining System Security," chapter of the *System Administrator's Guide.*

Diagnostics

integrity returns a zero exit status if there are no discrepancies, and 1 if discrepancies are found.

Value added

integrity is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

ipcrm

remove a message queue, semaphore set or shared memory ID

Syntax

ipcrm [*options*]

Description

ipcrm removes one or more specified messages, a semaphore or shared memory identifiers. The identifiers are specified by the following ***options***:

-q ***msqid*** removes the message queue identifier ***msqid*** from the system and destroys the message queue and data structure associated with it.

-m ***shmid*** removes the shared memory identifier ***shmid*** from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-s ***semid*** removes the semaphore identifier ***semid*** from the system and destroys the set of semaphores and data structure associated with it.

-Q ***msgkey*** removes the message queue identifier, created with key ***msgkey***, from the system and destroys the message queue and data structure associated with it.

-M ***shmkey*** removes the shared memory identifier, created with key ***shmkey***, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-S ***semkey*** removes the semaphore identifier, created with key ***semkey***, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in **msgctl**(S), **shmctl**(S), and **semctl**(S). The identifiers and keys may be found by using **ipcs**(ADM).

See also

ipcs(ADM), **msgctl**(S), **msgget**(S), **msgop**(S), **semctl**(S), **semget**(S), **semop**(S), **shmctl**(S), **shmget**(S), **shmop**(S)

Note

ipcrm cannot be used to remove semaphores created using **creatsem**(S) or to remove shared memory created using **sdget**(S).

ipcs

report the status of inter-process communication facilities

Syntax

ipcs [*options*]

Description

ipcs prints certain information about active inter-process communication facilities. Without ***options***, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following ***options***:

-q Print information about active message queues.

-m Print information about active shared memory segments.

-s Print information about active semaphores.

If any of the options **-q**, **-m**, or **-s** are specified, information about only those indicated are displayed. If none of the three options are specified, information about all three are displayed.

-b Print biggest allowable size information (maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores). See below, for the meaning of columns in a listing.

-c Print creator's login name and group name. See below.

-o Display information on outstanding usage (number of messages on queue, total number of bytes in messages on queue, and the number of processes attached to shared memory segments).

-p Display process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues. It displays the process ID of the creating process and the process ID of the last process to attach or detach on shared memory segments.) See below.

-t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last **msgsnd** and last **msgrcv** on message queues, last **shmat** and last **shmdt** on shared memory, and last **semop**(S) on semaphores.) See below.

-a Use all print ***options***. (This is a shorthand notation for **-b**, **-c**, **-o**, **-p**, and **-t**.)

-C ***corefile***
Use the file ***corefile*** in place of */dev/kmem.*

-N ***namelist***
The argument will be taken as the name of an alternate ***namelist*** (*/unix* is the default).

-X Print information about XENIX interprocess communication, in addition to the standard interprocess communication status. The XENIX process information describes a second set of semaphores and shared memory. Note that the **-p** option does not print process number information for XENIX shared memory, and the **-t** option does not print time information about XENIX semaphores and shared memory.

The column headings and the meaning of the columns in an **ipcs** listing are given below; the letters in parentheses indicate the ***options*** that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these ***options*** only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

- **q** message queue;
- **m** shared memory segment;
- **s** semaphore.

ID (all) The identifier for the facility entry. Note that **ID** is **"X"** for facilities created using **creatsem**(S) or **sdget**(S).

KEY (all) The key used as an argument to **msgget, semget,** or **shmget** to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** from when the segment has been removed until all processes attached to the segment detach it.)

MODE (all) The facility access modes and flags: The mode consists of 11 characters that are interpreted as follows:

The first two characters are:

- **R** if a process is waiting on a **msgrcv;**
- **S** if a process is waiting on a **msgsnd;**
- **D** if the associated shared memory segment has been removed. It will disappear when the last process attached to the segment detaches it;
- **C** if the associated shared memory segment is to be cleared when the first attach is executed;
- **–** if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

r if read permission is granted;

w if write permission is granted;

a if alter permission is granted;

– if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the associated queue.

LRPID (a,p) The process ID of the last process to receive a message from the associated queue.

STIME (a,t) The time the last message was sent to the associated queue.

RTIME (a,t) The time the last message was received from the associated queue.

CTIME (a,t) The time when the associated entry was created or changed.

NATTCH (a,o) The number of processes attached to the associated shared memory segment.

SEGSZ (a,b) The size of the associated shared memory segment.

CPID (a,p) The process ID of the creator of the shared memory entry.

LPID (a,p) The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t) The time the last attach was completed to the associated shared memory segment.

DTIME (a,t) The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b) The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t) The time the last semaphore operation was completed on the set associated with the semaphore entry.

Files

/unix	system namelist
/dev/kmem	memory
/etc/passwd	user names
/etc/group	group names

See also

msgop(S), **semop**(S), **shmop**(S)

Warning

If the user specifies either the **-C** or **-N** flag, the real and effective UID/GID will be set to the real UID/GID of the user invoking **ipcs**.

Note

Things can change while **ipcs** is running; the picture it gives is only a close approximation.

Authorization

The behavior of this utility is affected by assignment of the *mem* authorization. If you do not have this authorization, the output will be restricted to data pertaining to your activities only. Refer to the "Using a trusted system" chapter of the *User's Guide* for more details.

kbmode

set keyboard mode or test keyboard support

Syntax

/etc/kbmode *command* [*file*]

Description

This command can be used to determine if your system keyboard supports AT mode. If it does, this utility can change the keyboard mode between AT mode and PC/XT compatibility mode.

If the *file* argument is specified, it should be a tty device of one of the multiscreens of the keyboard's group.

Valid commands are:

test determine if keyboard supports AT mode

at set keyboard to AT mode

xt set keyboard to PC/XT compatibility mode

Notes

Some keyboards look like an AT keyboard but do not support AT mode. Setting such a keyboard to AT mode will render it useless, unless it can be set to XT mode from another (serial) terminal.

See also

keyboard(HW)

Value added

kbmode is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

killall

kill all active processes

Syntax

/etc/killall [***signal***]

Description

The **killall** command is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

The **killall** command terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

The **killall** command sends ***signal*** (see **kill**(C)) to all processes not belonging to the above group of exclusions. If no ***signal*** is specified, a default of 9 is used.

File

/etc/shutdown

See also

kill(C), **ps**(C), **shutdown**(ADM), **signal**(S)

Note

The **killall** command can only be run by the super user.

Standards conformance

killall is conformant with:

AT&T SVID Issue 2.

labelit

provide labels for filesystems

Syntax

/etc/labelit ***special*** **[** ***fsname volume*** **[-n]]**

Description

The **labelit** command can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **-n** option provides for initial labeling only. (This destroys previous contents.)

With the optional arguments omitted, **labelit** prints current label values.

The ***special*** name should be the physical disk section (e.g., */dev/dsk/0s3*). The device may not be on a remote machine.

The ***fsname*** argument represents the mounted name (e.g., *root*, *u1*, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette, or tape.

For file systems on disk, ***fsname*** and ***volume*** are recorded in the super block.

See also

filesystem(FP), **fsname**(ADM), **sh**(C)

Standards conformance

labelit is conformant with:

AT&T SVID Issue 2.

link, unlink

link and unlink files and directories

Syntax

/etc/link *file1 file2*

/etc/unlink *file*

Description

The **link** command is used to create a file name that points to another file. Linked files and directories can be removed by the **unlink** command; however, it is strongly recommended that the **rm**(C) and **rmdir**(C) commands be used instead of the **unlink** command.

The only difference between **ln**(C) and **link/unlink** is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the **link**(S) and **unlink**(S) system calls.

See also

link(S), **rm**(C), **unlink**(S)

Note

These commands can be run only by the super user.

Standards conformance

link and **unlink** are conformant with:

AT&T SVID Issue 2;
X/Open Portability Guide, Issue 3, 1989;
IEEE POSIX Std 1003.1-1990 System Application Program Interface (API) [C Language] (ISO/IEC 9945-1);
and NIST FIPS 151-1.

link_unix

build a new UNIX system kernel

Syntax

/etc/conf/cf.d/link_unix

Description

After installing a device driver, use **link_unix** to build a new UNIX system kernel. This script builds */etc/conf/cf.d/unix* using the current system configuration in */etc/conf*.

See also

configure(ADM), **idbuild**(ADM)

Value added

link_unix is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

list

list processor channel for MMDF

Syntax

list

Description

list is an MMDF channel program for handling mailing lists. The channel functions as a feed-through between **deliver** and **submit**. The list channel has its own host table and domain table with one entry for the pseudo host "list-processor" or something similar. This program is called by the program **deliver** and is not meant to be invoked by users directly.

The **list** channel performs two basic services. First, it postpones the verification of the list addresses and performs the (possibly lengthy) verification in the background when the **list** channel resubmits the message to the mail system. This prevents tying up a network connection or a user's terminal when verifying a long mailing list. Second, the **list** channel will, under special circumstances, change the return address for the message to a generic maintainer's address. The return address is determined by first taking the destination address (for example, "largelist") and seeing if there is an address in the alias file called "largelist-request". If there is, then "largelist-request" is used as the return address. If that was not found, the list channel checks to see if the destination address has a trailing "-outbound". If so, this is stripped and a "-request" is added and the lookup in the alias file is made a second time. If the "-request" address is found, then that address is used as the return address. If no "-request" address is found, then the original return address is used (normally the address of the sender).

To use the **list** channel to process a list, it is generally necessary to make three entries in the alias file(s). Let us say that we wish to set up a list called "largelist" and we want this list to be processed by the **list** channel. We would need the following entries in the alias file:

```
largelist:              largelist-outbound@list-processor
largelist-outbound:     :include: /usr/mmdf/lists/largelist-file
largelist-request:      maintainer
```

The first line causes mail sent to "largelist" to be sent through the list processor, readdressed to "largelist-outbound". The second line is what actually references the mailing list file for "largelist". The third line is optional, and is used to set up the (informal) standard maintenance address. This -request address, if present, will also be used by the **list** channel as the return address for mail submitted to the list.

See also

deliver(ADM), **submit**(ADM)

File

mmdf-table-directory/aliases to find list-request addresses

Credit

MMDF was developed at the University of Delaware and is used with permission.

lpadmin

configure the print service

Syntax

/usr/lib/lpadmin -p *printer options*

/usr/lib/lpadmin -x *dest*

/usr/lib/lpadmin -d [*dest*]

/usr/lib/lpadmin -S *print-wheel* **-A** *alert-type* [**-W** $integer_1$] [**-Q** $integer_1$]

Description

lpadmin configures the **lp** print service to describe printers and devices. It is used to add and change printers, to remove printers from the service, to set or change the system default destination, to define alerts for print wheels and to define printers for remote printing services.

Adding or changing a printer

The first form of the **lpadmin** command (**lpadmin -p** *printer options*) is used to configure a new printer or to change the configuration of an existing printer. The following options are used and may appear in any order. For ease of discussion, the printer will be referred to as "P" below.

-F *fault-recovery*
Restores the **lp** print service after a printer fault according to the value of *fault-recovery*:

continue Continues printing on the top of the page where printing stopped. This requires a filter (see **lpfilter**(ADM)) to wait for the fault to clear before automatically continuing.

beginning Starts printing the request again from the beginning.

wait Disables printing on the printer and waits for the administrator or a user to enable printing again.

During the wait, the administrator or the user who submitted the stopped print request can issue a change request that specifies where printing should resume. If no change request is made before printing is enabled, printing will resume at the top of the page where stopped if the filter allows; otherwise, the request will be printed from the beginning.

This option specifies the recovery to be used for any print request that is stopped because of a printer fault.

-c ***class***
Inserts printer "P" into the specified ***class***. ***class*** will be created if it does not already exist.

-D ***comment***
Saves ***comment*** for display whenever a user asks for a full description of the printer "P" (see **lpstat**(C)). The **lp** print service does not interpret this comment.

-e ***printer***
Copies an existing ***printer***'s interface program to be the new interface program for printer "P".

-f allow:*form-list*
-f deny:*form-list*
Allows (**-f allow**) or denies (**-f deny**) the forms in ***form-list*** to be printed on printer "**P**".

For each printer, the **lp** print service keeps two lists of forms: an "allow-list" of forms that can be used with the printer and a "deny-list" of forms that shouldn't be used with the printer. With the **-f allow** option, the forms listed are added to the allow-list and removed from the deny-list. With the **-f deny** option, the forms listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the forms in the list can be used with the printer and all others cannot, regardless of the content of the deny-list. If the allow-list is empty but the deny-list is not, the forms in the deny-list cannot be used with the printer. All forms can be excluded from a printer by having an empty allow-list and putting the word **any** in the deny-list. All forms can be used on a printer by having an empty deny-list and specifying **any** for the allow-list, provided the printer can handle all the characteristics of the forms.

The **lp** print service uses this information as a set of guidelines for determining where a form can be mounted. Administrators, however, are not restricted from mounting a form on any printer. If mounting a form on a particular printer is in disagreement with the information in the allow-list or deny-list, the administrator is warned, but the mount is accepted. Nonetheless, if a user attempts to issue a print or change request for a form-and-printer combination that is in disagreement with the information, the request is accepted only if the form is currently mounted on the printer. If the form is later unmounted before the request can print, the request is canceled, and the user is notified by mail.

If an administrator tries to name a form as acceptable for use on a printer that doesn't have the capabilities needed by the form, the command is rejected.

Note the other use of **-f** below.

-h Indicates that the device associated with printer "P" is hardwired. This option is assumed when adding a new printer unless the **-l** option is supplied.

-i ***interface***
Establishes a new interface program for printer "P". ***interface*** is the pathname of the new program.

-I ***content-type-list***
Assigns printer "P" to handle print requests with content of a type listed in ***content-type-list***.

The type **simple** is recognized as the default content-type of files on the system. Such a data stream contains only printable ASCII characters and the following control characters:

Control Character	Octal Value	Meaning
backspace	010	move back to previous column, except at beginning of line
tab	011	move to next tab stop
linefeed (newline)	012	move to beginning of next line
form feed	014	move to beginning of next page
carriage return	015	move to beginning of current line

To force the print service to not consider **simple** as a valid type for the printer, give an explicit value (for example, the printer type) in the ***content-type-list***. If you do want **simple** included along with other types, you must include **simple** in the ***content-type-list***.

Each printer automatically has its printer type included in the list of content types it will accept.

Except for **simple**, each ***content-type*** name is freely determined by the administrator. If names given as content types are also printer types, the names are accepted without comment because the **lp** print service recognizes all printer types as potential content types as well.

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

-M -f ***form-name*** **[-a [-o filebreak]]**
Mounts the form ***form-name*** on "P". Print requests to be printed with the pre-printed form ***form-name*** will be printed on "P". If more than one printer has the form mounted and the user has specified **any** (with the **-d** option of the **lp** command) as the printer destination, then each print request will be printed on the printer that meets the other needs of the request.

The page length and width and character and line pitches needed by the form are compared with those allowed for the printer by checking the capabilities in the *terminfo*(F) database for the type of printer. If the form

requires attributes that are not available with the printer, the administrator is warned, but the mount is accepted. If the form lists a particular print wheel as mandatory but the print wheel mounted on the printer is different, the administrator is also warned but the mount is accepted.

If the **-a** option is given, an alignment pattern is printed, preceded by the same initialization of the physical printer that precedes a normal print request with one exception: no banner page is printed. Printing is assumed to start at the top of the first page of the form. After the pattern is printed, the administrator can adjust the mounted form in the printer, press ⟨Return⟩ for another alignment pattern (no initialization this time), and continue printing as many alignment patterns as desired. The administrator can quit printing alignment patterns by typing "q".

If the **-o filebreak** option is given, a formfeed is inserted between each copy of the alignment pattern. By default, the alignment pattern is assumed to correctly fill a form, so no formfeed is added.

A form is unmounted by mounting a new form in its place using the **-f** option. The **-f none** option can be used to specify no form. By default, a new printer has no form mounted.

Note the other use of **-f** above.

-M -S ***print-wheel***

Mounts the print wheel ***print-wheel*** on printer "P". Print requests to be printed with ***print-wheel*** will be printed on that printer. If more than one printer has the ***print-wheel*** mounted and the user has specified **any** (with the **-d** option of the **lp** command) as the printer destination, then each print request will be printed on the one that meets the other needs of the request.

If the ***print-wheel*** is not listed as acceptable for the printer, the administrator is warned, but the mount is accepted. If the printer does not take print wheels, the command is rejected.

A print wheel is unmounted by mounting a new print wheel in its place or by using the **-S none** option.

By default, a new printer has no special print wheel mounted. Until this is changed, a print request that asks for a specific print wheel will not be printed on a new printer with no special print wheel mounted.

Note the other uses of the **-S** option described below.

-m ***model***

Selects a model interface program provided with the **lp** print service for a given printer.

-o *printing-option*

Each **-o** option in the list below is the default given to an interface program if the option is not taken from a preprinted form description or is not explicitly given by the user submitting a request (see **lp**(C)). The only **-o** options that can have defaults defined are listed below:

length = *scaled-decimal-number*
width = *scaled-decimal-number*
cpi = *scaled-decimal-number*
lpi = *scaled-decimal-number*
stty = *stty-option-list*

The term *scaled-decimal-number* refers to a non-negative number used to indicate a unit of size. (The type of unit is shown by a trailing letter attached to the number.) Three types of scaled decimal numbers are discussed for the **lp** print service: numbers that show sizes in centimeters (marked with a trailing "c"), numbers that show sizes in inches (marked with a trailing "i"), and numbers that show sizes in units appropriate to use (without a trailing letter), that is, lines, columns, characters per inch (cpi) or lines per inch (lpi).

The first four default option values should agree with the capabilities of the type of physical printer as defined in the *terminfo*(F) database for the printer type. If they do not, the command is rejected.

The *stty-option-list* is not checked for allowed values but is passed directly to the **stty**(C) program by the standard interface program. Any error messages produced by **stty** when a request is processed (by the standard interface program) are mailed to the user submitting the request.

For each printing option not specified, the defaults for the following attributes are defined in the *terminfo* entry for the specified printer type:

length
width
cpi
lpi

The default for **stty** is

stty = 9600 cs8 -cstopb -parenb -paroff ixon
-ixany opost -olcuc -onlcr -ocrnl -onocr
-onlret -ofill nl0 cr0 tab0 bs0 vt0 ff0

You can set any of the **-o** options to the default values (which vary for different types of printers) by typing them without assigned values as follows:

length=
width=
cpi=
lpi=
stty=

-o nobanner
Allows users to submit a print request that asks that no banner page be printed.

-o banner
Forces a banner page to be printed with every print request, even when a user asks for no banner page. This is the default; you must specify **-o nobanner** if you want to allow users to specify **-o nobanner** with the **lp** command.

-R ***machine-list***
Sets up remote machines in ***machine-list*** to share print services. The **lp** print service arranges for the advertising and mounting of all necessary resources and for automatic recovery of shared print services when the machine is brought to a state where RFS is run.

The **lp** spooler keeps the parts of the print service owned by each machine separate, so that the administrator on one machine can change only the service provided by his or her machine. The **lp** spooler provides for no centrally managed print service using RFS.

-r ***class***
Removes a given printer from the specified ***class***. If the printer is the last member of the ***class***, then the ***class*** will be removed.

-S ***list***
Allows the aliases for character sets or print wheels named in ***list*** to be used with a given printer.

If the printer is a type that takes print wheels, then ***list*** is a list of print wheel names separated by commas or spaces. These will be the only print wheels considered mountable on the printer. (You can always force a different print wheel to be mounted, however.) Until the option is used to specify a ***list***, no print wheels will be considered mountable on the printer, and print requests that ask for a particular print wheel with this printer will be rejected.

If the printer is a type that has selectable character sets, then ***list*** is a list of character set name "mappings" or aliases separated by commas or spaces. Each "mapping" is of the form:

known-name = synonym

known-name is a character set number preceded by "cs", such as "cs3" for character set three, or a character set name from the *terminfo* database "csnm" entry. If this option is not used to specify a list, only the names already known from the *terminfo* database or numbers with a prefix of "cs" will be acceptable for the printer.

If ***list*** is the word **none**, the previous print wheel list or character set aliases will be removed.

Note the other uses of the **-S** option.

-T ***printer-type***

Assigns the given ***printer-type***, a representation of a physical printer of type ***printer-type***. ***printer-type*** is used to extract data from *terminfo*(F); this data is used to initialize the printer before printing each user's request. Some filters may also use ***printer-type*** to convert content for the printer. If this option is not used, the default ***printer-type*** will be **unknown**; no useful information will be extracted from *terminfo*(F), so each user request will be printed without first initializing the printer. Also, this option must be used if the following are to work: **-o cpi=**, **-o lpi=**, **-o width=**, and **-o length=** options of the **lpadmin** and **lp** commands, and the **-S** and **-f** options of the **lpadmin** command.

-u allow:*user-list*
-u deny:*user-list*

Allows (**-u allow**) or denies (**-u deny**) the users in ***user-list*** access to a given printer.

For normal access to each printer, the **lp** print service keeps two lists of users: an ***allow-list*** of people allowed to use the printer and a ***deny-list*** of people denied access to the printer. With the **-u allow** option, the users listed are added to the allow-list and removed from the deny-list. With the **-u deny** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the printer and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users except the **lp** print service administrator by putting **any** in the deny-list. To allow everyone access to a given printer and effectively empty both lists, put **any** in the allow-list.

-U ***dial-info***

Assigns the dialing information ***dial-info*** to the printer. ***dial-info*** is used with the **dial**(S) routine to call the printer. Any network connection supported by the Basic Networking Utilities will work. ***dial-info*** can be either a phone number for a modem connection or a system name for other kinds of connections. Or if **-U direct** is given, no dialing will take place because the name **direct** is reserved for a printer that is directly connected. If a system name is given, it is used to search for connection details from the file */usr/lib/uucp/Systems* or related files. The Basic Networking Utilities are required to support this option. By default, **-U direct** is assumed.

-v ***device***

Associates a new ***device*** with a given printer. ***device*** is the pathname of a file that is writable by **lp**. Note that the same ***device*** can be associated with more than one printer.

-A *alert-type* [**-W** *integer*]
The **-A** option is used to send the alert *alert-type* to the administrator when a printer fault is first detected and periodically thereafter until the printer fault is cleared by the administrator. The *alert-types* are:

mail Sends the alert message via **mail** (see **mail**(C)) to the administrator who issues this command.

write Writes the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Does not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the fault has been cleared and printing resumes, messages will again be sent when another fault occurs with the printer.

none Does not send messages until this command is given again with a different *alert-type*; removes any existing alert definition. No alert will be sent when the printer faults until a different *alert-type* is used (except quiet).

shell-command
shell-command is run each time the alert needs to be sent. *shell-command* should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail**(C) and **write**(C) values for this option are equivalent to the values **mail** *user-name* and **write** *user-name*, respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the **su** command to change to another user ID. If the **su** command has been used to change the user ID, then the *user-name* for the new ID is used.

list The type of the alert for the printer fault is displayed on the standard output. No change is made to the alert.

The message sent appears as follows:

```
The print wheel print-wheel needs to be mounted
on the printer(s):
printer-list
number-of-requests print requests await this print-wheel.
```

The printer *printer-name* has stopped printing for the reason given below. Fix the problem and bring the printer back on line. Printing has stopped but will be restarted in a few minutes; issue an enable command if you want to restart sooner.

Unless someone issues a change request

lp -i *request-id* **-P** ...

to change the page-list to print, the current request will be repeated from the beginning.

The reason(s) it stopped (multiple reasons indicate reprinted attempts):

reason

The **lp** print service can detect printer faults only through an adequate fast filter and (see **lpfilter**(ADM)) only when the standard interface program or a suitable customized interface program is used. Furthermore, the level of recovery after a fault depends on the capabilities of the filter.

If the *printer-name* is **all**, the alerting defined in this command applies to all existing printers.

If the **-W** option is not given or *integer* is zero (which represents **once** and is also the default), only one message will be sent per fault. If this command is not used to arrange fault alerting for a printer, the default procedure is to mail one message per fault to the administrator of the printer.

Restrictions

When creating a new printer, either the **-v** or the **-U** option must be supplied. In addition, only one of the following may be supplied: **-e**, **-i**, or **-m**; if none of these three options are supplied, the model standard is used. 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).

Removing a printer destination

The second form of the **lpadmin** command, **lpadmin -x***dest*, removes the destination *dest* from the **lp** print service. If *dest* is a printer and is the only member of a class, then the class will be deleted too. If *dest* is **all**, all printers and classes are removed. No other options are allowed with **-x**.

Changing the system default destination

The third form of the **lpadmin** command, **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. No other options are allowed with **-d**.

Setting an alert for a print wheel

The fourth form of the **lpadmin** command, **lpadmin -S** ***print-wheel*** **-A** ***alert-type*** [**-W** $integer_1$] [**-Q** $integer_2$], sends the alert ***alert-type*** to the administrator as soon as the ***print-wheel*** needs to be mounted and periodically thereafter. The ***alert-types*** are

mail Sends the alert message via mail (see **mail**(C)) to the administrator who issues this command.

write Writes the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Does not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the ***print-wheel*** has been mounted and subsequently unmounted, messages will again be sent when the number of print requests again exceeds the threshold.

none Does not send messages until this command is given again with a different ***alert-type*** (other than **quiet**).

shell-command
The ***shell-command*** is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** ***user-name*** and **write** ***user-name***, respectively, where ***user-name*** is the current name for the administrator. This will be the login name of the person submitting this command unless he or she has used the **su** command to change to another user ID. If the **su** command has been used to change the user ID, then the ***user-name*** for the new ID is used.

list The type of the alert for the print wheel is displayed on the standard output. No change is made to the alert.

The printers listed are those that the administrator had earlier specified were candidates for this print wheel. The number ($integer_3$) listed next to each printer is the number of requests eligible for the printer. The number ($integer_4$) shown after the printer list is the total number of requests awaiting the print wheel. It will be less than the sum of the other numbers if some requests can be handled by more than one printer.

If the ***print-wheel*** is **all**, the alerting defined in this command applies to all print wheels already defined to have an alert.

Only one administrator per print wheel can be alerted. If this command is run by more than one administrator for the same print wheel, the last command run applies.

If the **-W** option is not given or ***integer***$_1$ is 0 (which is interpreted as **once** and is also the default), only one message will be sent per need to mount a print wheel. If this command is not used to arrange alerting for a print wheel, no alerts will be sent for the print wheel.

If the **-Q** option is also given, the alert will be made when ***integer***$_2$ print requests that need the print wheel are waiting. If the **-Q** option is not given or ***integer***$_2$ is 1 or the word **any**, a message is sent as soon as anyone submits a print request for the print wheel when it is not mounted.

The **-S** option has a different meaning when used with the **-p** option.

Defining remote printers for remote printing services

The fifth form of the **lpadmin** command is used to define the remote printer, ***printer-name***$_2$, and its machine, ***machine-name***, that will handle remote print requests from the local machine. The remote printer will be referred to as ***printer-name***$_1$ on the local machine.

Files

*/usr/spool/lp/**

See also

accept(ADM), **enable**(C), **lp**(C), **lpfilter**(ADM), **lpsched**(ADM), **lpstat**(C), **stty**(C), **terminfo**(F)

Authorization

Permission to use this utility requires the *lp* authorization.

lpfilter

administer filters used with the print service

Syntax

/usr/lib/lpfilter -f ***filtername*** **-C** ***pathname***

/usr/lib/lpfilter -f ***filtername*** **-**

/usr/lib/lpfilter -f ***filtername*** **-i**

/usr/lib/lpfilter -f ***filtername*** **-x**

/usr/lib/lpfilter -f ***filtername*** **-l**

Description

The **lpfilter** command is used to add, change, delete, and list filters used with the LP print service. (The functions of **lpfilter** are also accessible through the **sysadmsh**(ADM) Printers ⇨ Auxillary ⇨ Filter selection.) These filters are used to convert the content type of a file to a content type acceptable to a given printer. One of the following options must be used with the **lpfilter** command: **-C** ***pathname*** (or **-** for standard input) to add or change a filter, **-i** to reset an original LP print service filter to its factory setting, **-x** to delete a filter, or **-l** to list a filter description.

The argument **all** can be used instead of a ***filtername*** with any of these options. When **all** is specified with the **-C** or **-** option, the requested change is made to all filters. Using **all** with the **-i** option has the effect of restoring to their original settings all filters for which predefined settings were initially available. Using the **all** argument with the **-l** option produces a list of all filters, and using **all** with the **-x** option results in all filters being deleted.

Filters can be set up to do the following:

- Convert a user's file into a data stream that prints properly on a given printer.
- Handle the various modes of printing that people may request with the **-y** option to the **lp** command, such as two-sided printing, landscape printing, draft or letter-quality printing, and so on.
- Detect printer faults and inform the print service, which in turn can alert the system administrator.

In general these three functions are very printer-specific, and a single filter capable of doing everything would be very complex. Therefore, in order to provice a flexible print service, the roles are separated out in a modular fashion. Consequently, the system administrator can substitute a different filter that performs a specific function without changing the rest of the print service.

A default filter is provided with the print service to provide simple printer fault detection; it does not convert files or handle any of the special modes. This may be adequate for your needs.

Converting files

The print service allows you to type each printer you add to the system and allows a user to type each file he or she submits for printing. This information is used to match a file with the printer that can best reproduce that file. Because many applications can generate data for various printers, this is often sufficient. However, not all of the applications you use may be able to generate output that works on your printers.

By defining and creating a filter that converts such output into a type that your printers can handle, you can begin to support more applications in the print service. A small set of simple filters are provided that convert output from applications like **nroff** to data streams that print properly on some printers.

Each filter that is added to the system is classified with the input type it can accept and the output type it can produce. Now the print service can be more sophisticated in its attempt to match a user's file with a printer. If it cannot find a direct match, it consults the table of filters to find one that converts the file's type into the printer's type.

Handling special modes

Another important role that filters can provide is the handling of the various printing modes that may be encountered. Each filter you add to the filter table can be registered as handling several aspects of printing. These are listed here:

Input type
Output type
Printer type
Character pitch
Line pitch
Page length
Page width
Pages to print
Character set
Form name
Number of copies
Modes

A filter is not required to handle most of these, only the modes. The print service provides a default handling for the rest. However, it may be more efficient to have a filter handle these, or it may be that a filter has to know several of these aspects if it is to fulfill its other roles properly. A filter may need to know, for example, the page size and the print spacing if it is going to break up the pages in a file to fit on the printed pages. As another example, some printers can handle multiple copies more efficiently than the print service can, so a filter that is controlling the printer can use the number of copies information to skip the print service's default handling of this.

Later we see how you can register the printing modes and other aspects of printing with each filter.

Detecting printer faults

Just as converting a file and handling special printing modes is a printer-specific role, so is the detection of printer faults. The print service attempts to do this in general, and for most printers it properly detects a fault. However, it is limited to checking for "hang-ups" (loss of carrier or the signal that indicates the printer is on-line) and excessive delays in printing (that is, receipt of an XOFF flow-control character to shut off the data flow with no matching XON to turn the flow back on). It also cannot determine the cause of the fault, so it cannot tell you what to look for.

A properly designed filter can provide better fault coverage. Some printers can send a message to the host describing the reason for a fault. Others indicate a fault by dropping carrier or shutting off data flow. A filter can serve you by giving more information about a fault and detecting more of them.

Another benefit a filter can give is to wait for a printer fault to clear and to resume printing. This allows for more efficient printing when a fault occurs because the print request that was interrupted does not have to be reprinted in its entirety. Only a real filter, which understands the control sequences used by a printer, know where a file breaks into pages; thus, only the filter knows how far back to go in the file to restart properly.

The print service has a simple interface that lets the filter get the fault information to you and restart if it can. The alerting mechanism is handled by the print service; the interface program that manages the filter takes all error messages from the filter and places them into an alert message that can be sent to you. Thus, you see any fault descriptions that the filter puts out. If you set the printer configuration so that printing should automatically resume after a fault is cleared, the interface program keeps the filter active so that it can pick right up where it left off.

Adding or changing a filter

Use **lpfilter -f** *filtername* to add or change a filter.

The filter named in the **-f** option and described in the input is added to the filter table. If the filter already exists, its description is changed to reflect the new information in the input. Once added, a filter is available for use.

The filter description is taken from the pathname if the **-C** option is given or from the standard input if the **-** option is given. One of the two must be given to define or change a filter. If the filter named is one originally delivered with the LP print service, the **-i** option will restore the original filter description.

Filters are used to convert the content of a request into a data stream acceptable to a printer. For a given print request, the LP print service will know the following:

- the type of content in the request
- the name of the printer
- the type of the printer
- the types of content acceptable to the printer
- the modes of printing asked for by the originator of the request

It will use this information to find a filter that will convert the content into a type acceptable to the printer.

Below is a list of items that provide input to this command and descriptions of each item. All lists are separated by commas or spaces.

Input types: ***content-type-list***
Output types: ***content-type-list***
Printer types: ***printer-type-list***
Printers: ***printer-list***
Filter type: ***filter-type***
Command: ***shell-command***
Options: ***template-list***

Input types This is the list of file types that the filter can process. Most filters can take only one input type, but the print service does not restrict them to one. Several file types may be similar enough for the filter that it can deal with them. You can use whatever names you like here. Because the print service uses these names to match a filter with a file type, you should be consistent in your naming convention. For example, if more than one filter can accept the same input type, use the same name.

These names should be advertised to your users so they know how to name a file's type when they submit the file for printing.

Output types This is the list of file types that the filter can produce as output. For each file, the filter produces a single output type, but it may be able to vary that type on demand.

These names should either match the types of printers you have on your system or should match the input types handled by other filters. The print service gangs filters together in a shell pipeline to produce a new filter if it finds that several passes by different filters are needed to convert a file. It is unlikely that you need this level of sophistication, but the print service allows it. Try to find a set of filters that take as input types all the different files your users may want printed and that convert those files directly into types your printers can handle.

Printer types This is a list of printer types into which the filter can convert files. While for most filters this list is identical to the output types, it can be different.

For example, you may have a printer that is given a single type for purposes of initialization but which can recognize several different types of files. In essence, these printers have an internal filter that converts the various types into one with which they can deal. Thus, a filter may produce one of several output types that match the "file types" that the printer can handle. The filter should be marked as working with that printer type.

As another example, you may have two different models of printers that are listed as both accepting the same types of files. However, due to slight differences in manufacture, one printer deviates in the results it produces. You label the printers as being of different printer types, say A and B, where B is the one that deviates. You create a filter that adjusts files to account for the deviation produced by printers of type B. Because this filter is only needed for those printer types, you would list it as working only on type B printers.

For most printers and filters, you can leave this part of the filter definition blank.

Printers You may have some printers that, although they are of the correct type for a filter, are in other ways not adequate for the output that the filter produces. For instance, you may want to dedicate one printer for fast turn-around; only files that the printer can handle without filtering are sent to that printer. Other printers, of identical type, you allow for files that may need extensive filtering before they can be printed. You will label the filter as working with only the latter printers.

In most cases, the filter should be able to work with all printers that accept the output that the filter produces, so you can leave this part of the filter definition blank.

Filter type The print service recognizes fast filters and slow filters. Fast filters are labeled fast either because they incur little overhead in preparing a file for printing or because they must have access to the printer when they run. A filter that is to detect printer faults has to be a fast filter. Slow filters are the opposite. Filters that incur a lot of overhead in preparing a file and that do not have to have access to the printer should be labeled slow. The print service runs slow filters in the background without tying up a printer. This allows files that need fast filtering (or no filtering) to move ahead; printers are not left idle while a slow filter works on a file if other files can be printed.

Command This is the full pathname of the program to run; this is the filter. If there are any fixed options that the program always needs, you can include them here.

Options This is a list of templates separated by commas used by the LP print service to construct options to the filter from the characteristics of each print request listed in the table later. In general, each template is of the following form:

keyword pattern = replacement

The ***keyword*** names the characteristic that the template attempts to map into a filter-specific option; each valid ***keyword*** is listed in the table below. A ***pattern*** is either a literal pattern of one of the forms listed in the table or a single asterisk, (*); if the ***pattern*** matches the value of the characteristic, the template fits and is used to generate a filter-specific option. A ***pattern*** of "*" matches any value. The ***replacement*** is a string used as a filter-specific option with an embedded asterisk (*) replaced with the value of the characteristic.

lp option	Characteristic	keyword	Possible
-T	Content type (input)	INPUT	content-type
N/A	Content type (output)	OUTPUT	content-type
N/A	Printer type	TERM	printer-type
-f, -o cpi=	Character pitch	CPI	integer
-f, -o lpi=	Line pitch	LPI	integer
-f, -o length=	Page length	LENGTH	integer
-f, -o width=	Page width	WIDTH	integer
-P	Pages to print	PAGES	page-list
-S	Character set/ print wheel	CHARSET	character-set-name/ print-wheel-name
-f	Form name	FORM	form-name
-y	Modes	MODES	mode
-n	Number of copies	COPIES	integer

If a pattern or replacement must include a comma or equals sign (=), escape its special meaning by preceding it with a backslash. A backslash in front of these two characters is removed when the pattern or replacement is used. (All other backslashes are left alone.)

The source of the values for these templates are as follows:

- The values of the **INPUT** and **OUTPUT** templates come from the file type that needs to be converted by the filter and the output type that has to be produced, respectively. They will each a type registered with the filter.
- The value for the **TERM** template is the printer type.
- The values for the **CPI**, **LPI**, **LENGTH**, and **WIDTH** templates come from the user's request, the form being used, or the defaults for the printer.
- The value for the **PAGES** template is a list of pages that should be printed. Typically it is a list of page ranges, either a pair of numbers or a single number, each range separated by a comma (for example, 1-5,6,8,10 for pages 1 through 5, 6, 8, and 10). However, whatever value was given in the **-P** option to a print request is passed unchanged.
- The value for the **CHARSET** template is the name of the character set to be used.
- The value for the **FORM** template is the name of the form being printed on, if any.
- The value of the **COPIES** template is the number of copies of the file that should be made. If the filter uses this template, the print service reduces the number of copies of the filtered file it prints to 1, because this "single copy" is really the multiple copies produced by the filter.
- The value of the **MODES** template comes from the **-y** option of the **lp** command, the command a person uses to submit a print request. Because a user can give several **-y** options, there may be several values for the **MODES** template. The values are applied in the left-to-right order given by the user.

For example, the template

MODES landscape = -l

would show that if a print request includes the **-y landscape** option, the filter should be given the option **-l**. As another example, the template

TERM * = -T *

would show that the filter should be given the option **-T** ***printer-type*** for whichever ***printer-type*** is associated with a print request using the filter.

When an existing filter is changed with this command, items that are not specified in the new information are left as they were. When a new filter is added with this command, unspecified items are given default values.

Note that a filter name and a command must be given. A filter with no input type value is assumed to work with any input type; this is also true for the output type, printer type, and printer values.

After you gather this information about the filter, use it as input to the **lpfilter**(ADM) command (or the **sysadmsh** equivalent). You may want to first record this information in your own file to make it easier to edit the information as you enter it. You can then give the file as input instead. However you enter it, you should present the information in the following way:

```
Input types: input-type-list
Output types: output-type-list
Printer types: printer-type-list
Printers: printer-list
Filter type: fast or slow
Command: simple-command
Options: template-list
```

The information can appear in any order. Not all the information has to be given. The table below contains the defaults used for any missing information.

Item	Default
Input types	any
Output types	any
Printer types	any
Printers	any
Filter type	slow
Command	(no default)
Options	(none)

As you can see, the defaults define a very flexible filter, so you probably have to supply at least the input and output type(s). When you enter a list, separate the items in the list with blanks or commas.

Once you have a filter definition complete, use one the following commands to add it to the system:

/usr/lib/lpfilter -f *filtername* **-F** *filename*
/usr/lib/lpfilter -f *filtername* **-**

Δ **sysadmsh** users select: Printers ⇨ Auxiliary ⇨ Filter ⇨ Change

The first command gets the filter definition from a file, and the second command gets the filter definition from you through the standard input. The *filtername* can be anything you choose as long as it contains 14 or less letters, digits, and underscores.

If you need to change a filter, just re-enter one of the same commands. You need only give the changed information; information you leave out stays the same.

Removing a filter

The print service has no fixed limit to the number of filters you can define. However, it is a good idea to remove filters no longer applicable to avoid extra processing by the print service, which must examine all filters to find one that works in a given situation.

Use the following command to remove a filter:

/usr/lib/lpfilter -f ***filtername*** **-x**

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ Filter ➪ Remove

Listing a filter description

You can examine a filter definition once you add it to the print service. The **lpfilter** command displays the definition of the filter in a form suitable as input again so that you can save the output in a file for future reference.

You can use one the following commands to examine a defined filter:

/usr/lib/lpfilter -f ***filtername*** **-l**
/usr/lib/lpfilter -f ***filtername*** **-l >***filename*

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ Filter ➪ List

The first command presents the definition of the filter on your screen; the second command captures this definition in a file, which can later be used to redefine the filter if you inadvertently remove the filter from the print service.

The **-l** option is used to list the description of the filter named in ***filter-name***. If the command is successful, the same listing of filter attributes is displayed that is shown earlier. If the command fails, an error message is sent to standard error.

See also

lp(C), **lpadmin**(ADM)

Authorization

Permission to use this utility requires the lp authorization.

Examples

This section contains several examples of filters and their uses.

Example 1

The filter program is called */usr/bin/npf*. The program takes two input types, nroff37 and X, produces an output type called TX, and works with any printer of type TX. The program accepts three options:

-Xb	Only for the input type X,
-l *integer*	For the length of the output page.
-w *integer*	For the width of the output page.

The filter definition looks like this:

```
Input types: X,nroff37
Output types: TX
Printer types: TX
Command: /usr/bin/npf
Options: INPUT X = -Xb, LENGTH * = -l*,
WIDTH * = -w*
```

A user submits a file of type nroff37 and asks that it be printed by a printer named *lp1* which is of type TX, and requests a page length of 72:

lp -T nroff37 -d lp1 -o length=72

This filter is called upon by the print service to convert the file. The filter is invoked as:

/usr/bin/npf -l72

Example 2

Another user submits a file of type X that is to be printed on the same printer, with default length and width. The filter is invoked as:

/usr/bin/npf -Xb

Example 3

The filter program is called */usr/bin/x9700*. It takes one input type, troff, produces an output type called 9700, and will work with any printer of type 9700. The program has one fixed option, **-ib**, and accepts three other options:

-l *integer*	For the length of the output page.
-s *name*	For the character set.
-o portrait **-o landscape**	For portrait or landscape orientation of the paper.

You decide that your users need to give just the abbreviations **port** and **land** when they ask for the paper orientation. Because these are not options intrinsic to the print service, users specify them using the **-y** option to the **lp** command.

The filter definition looks like this:

```
Input types: troff
Output types: 9700
Printer types: 9700
Command: /usr/bin/x9700 -ib
Options: LENGTH * = -l *, CHARSET * = -s *,
         MODES port = -o portrait, MODES land
         = -o landscape
```

(The last line is split into three lines for readability in this example. It must be entered as a single line.)

A user submits a file of type troff for printing on a printer of type 9700 and requests landscape orientation using the gothic character set:

lp -T troff -S gothic -y land

This filter is invoked by the print service to convert the file as follows:

/usr/bin/x9700 -ib -s gothic -o landscape

Notes

It is tempting to use a program like **troff**, **nroff**, or a similar word-processing program as a filter. However, the **troff** and **nroff** programs have a feature that allows people to reference additional files in the source document; these are called "include files". The spooler does not know about these files and does not queue them with the source document. The **troff** or **nroff** program may fail because it cannot access these additional files. Other programs may have similar features that limit their use as filters.

Here are guidelines that can help you choose a good filter:

- Examine the kinds of files people submit for printing that have to be processed by the filter. If they stand alone, that is, if they do not reference other files that the filter needs, the filter is probably okay. Check also to see if the filter expects any other files except those submitted by a user for printing.
- If there can be referenced files inside the files submitted for printing or if the filter needs files other than those submitted by a user, then the filter is likely to fail because it does not access the additional files. We suggest you do not use the program as a filter, but have each user run the program before submitting the files for printing.

Referenced files that are always given with full pathnames may be okay, but only if the filter is used for local print requests. When used on requests submitted from a remote machine for printing on your machine, the filter may still fail if the referenced files are only on the remote machine.

While a filter can be as simple or as complex as needed, there are only a few external requirements:

- The filter should get the content of a user's file from its standard input and send the converted file to the standard output.
- A slow filter can send messages about errors in the file to standard error. A fast filter should not send messages, as described below. Error messages from a slow filter will be collected and sent to the user who submitted the file for printing.
- If a slow filter dies because of receiving a signal, the print request is finished and the user who submitted the request is notified. Likewise, if a slow filter exits with a non-zero exit code, the print request is finished and the user is notified. The exit codes from fast filters are treated differently, as described later.
- A filter should not depend on other files that are not normally accessible to a regular user; if the filter fails when the user ran it directly, it will fail when the print service runs it.

There are a few more requirements if the filter is also to detect printer faults:

- If it can, it should wait for a fault to clear before exiting. Additionally, it should continue printing at the top of the page where printing stopped after the fault clears. If this is not the administrator's intention, the print service should stop the filter before alerting the administrator.
- The filter should send printer fault messages to its standard error as soon as the fault is recognized. It does not have to exit but can wait as described above.
- It should not send messages about errors in the file to standard error. Any messages on the standard error eventually generate a pointer fault. These should be included in the standard output stream, where they can be read by the user.
- It should exit with a zero exit code if the user's file is finished (even if errors in the file prevented it from printing correctly).
- It should exit with a non-zero exit code only if a printer fault kept it from finishing a file.
- When added to the filter table, it must be added as a fast filter.

Warnings

Adding, changing, or deleting filters can cause print requests still queued to be canceled. This is because the print service evaluates each print request still queued to see which are affected by the filter change. Requests that are no longer printable, because a filter has been removed or changed, are canceled (with notifications sent to the people who submitted them). There can also be a delay in the response to new or changed print requests when filters are changed, due to the many characteristics that must be evaluated for each print request still queued. This delay can become noticeable if there are a large number of requests needing filtering.

Because of this possible impact, you may want to make changes to filters during periods when the print service is not being used much.

lpforms

administer forms used with the print service

Syntax

/usr/lib/lpforms -f *formnameoption*

/usr/lib/lpforms -f *formname* **-A** *alert-type* [**-Q** $integer_1$] [**-W** $integer_2$]

/usr/lib/lpforms -f *formname* **-A list**

/usr/lib/lpforms -f *formname* **-A quiet**

/usr/lib/lpforms -f *formname* **-A none**

Description

The **lpforms** command is used to administer forms. (The functions of **lpforms** are also accessible through the **sysadmsh**(ADM) Printers ⇨ Auxiliary ⇨ Forms selection.) A preprinted form is a paper image of a blank form that you can load into your printer. An application typically generates a file that, when printed on the blank form, fills out the form. Common examples of forms are:

- blank checks,
- vouchers,
- receipts,
- labels,
- company letterhead, and
- special paper stock.

Typically, several copies of the blank form are loaded into the printer either as a tray of single sheets or as a box of fan-folded paper.

The print service helps you manage the use of preprinted forms but does not provide your application any help in filling out a form. This is solely your application's responsibility. The print service, however, keeps track of which print requests need special forms mounted and which forms are currently mounted, and it can alert you to the need to mount a new form. Using this command you can:

- define a new form,
- change an old form,
- remove a form,
- examine a form,
- restrict user access to a form,

- arrange alerting to the need to mount a form, and
- mount a form.

Options

The following ***options*** are available with the first form of the command, **/usr/lib/lpforms -f *formname* option**:

-F *pathname* to add or change a form as specified by the information in ***pathname***.

- to add or change a form, and supply information from standard input.

-x to delete a form. This option must be used separately; it cannot be used with any other option.

-l to list the attributes of a form. This option must be used separately; it cannot be used with any other option.

-u allow:*user-list* to allow users to request a form. This option can be used with the **-F** or **-** option.

-u deny:*user-list* to deny users access to a form. This option can be used with the **-F** or **-** option.

-A *alert-type* to define the type of alerting method to be used. The values are **list**, **quiet**, **none**, **mail**, **write**, and **'command'**.

-Q *number* defines the threshold in number of requests waiting, that is, used to restart the alert. Must be used with **-A**.

-W *minutes* defines the number of minutes between alerts. Must be used with **-A**.

Each of these options is explained in the sections that follow. **sysadmsh**(ADM) selections are referenced where possible to simplify the task of administering forms.

Adding or changing a form

The **-F *pathname*** option is used to add a new form to the **lp** print service, or to change the attributes of an existing form. The form description is taken from ***pathname*** if the **-F** option is given, or from the standard input if the **-** option is given. One of the two options must be given to define or change a form. ***pathname*** is the pathname of a file that contains all or any subset of the following information about the form.

Page length: *scaled-decimal-number*$_1$
Page width: *scaled-decimal-number*$_2$
Number of pages: *integer*
Line pitch: *scaled-decimal-number*$_3$
Character pitch: *scaled-decimal-number*$_4$
Character set choice: *character-set/print-wheel* **[mandatory]**
Ribbon color: *ribbon-color*
Comment:
comment
Alignment pattern: [*content-type*]
content

Except for the last two lines, the above lines can appear in any order. The **Comment:** and *comment* items must appear in consecutive order but can appear before the other items, and the **Alignment pattern:** and the *content* items must appear in consecutive order at the end of the file. Also, the *comment* item cannot contain a line that begins with any of the key phrases above, unless the key phrase is preceded with a ">" sign. Any leading ">" sign found in the *comment* will be removed when the comment is displayed. Case distinctions in the key phrases are ignored.

The print service does not try to mask sensitive information in an alignment pattern. If you do not want sensitive information printed on sample forms — probably the case when you align checks, for instance — then you should mask the appropriate data. The print service keeps the alignment pattern stored in a safe place, where only you (that is, the user *lp* and the super user *root*) can read it.

Upon issuing this command, the form named in *formname* is added to the list of forms. If the form already exists, its description is changed to reflect the new information in the input. Once added, a form is available for use in a print request, except where access to the form has been restricted, as described under the **-u allow:** option. A form may also be allowed to be used on certain printers only.

A description of each form attribute is given below:

Page length and **Page Width**

Before printing the content of a print request needing this form, the generic interface program provided with the **lp** print service will initialize the physical printer to handle pages **scaled-decimal-number**$_1$ long, and **scaled-decimal-number**$_2$ wide using the printer type as a key into the *terminfo*(F) database. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter "i", for inches, or the letter "c", for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with "i").

For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

The page length and page width will also be passed, if possible, to each filter used in a request needing this form.

Number of pages
Each time the alignment pattern is printed, the print service will attempt to truncate the ***content*** to a single form by, if possible, passing to each filter the page subset of 1 - ***integer***.

Line pitch and **Character pitch**
Before printing the content of a print request needing this form, the interface programs provided with the **lp** print service will initialize the physical printer to handle these pitches, using the printer type as a key into the *terminfo*(F) database. Also, the pitches will be passed, if possible, to each filter used in a request needing this form. ***Scaled-decimal-number***$_3$ is in lines per centimeter if a 'c' is appended, and lines per inch otherwise; similarly, ***scaled-decimal-number***$_4$ is in columns per centimeter if a 'c' is appended, and columns per inch otherwise. The character pitch can also be given as *elite* (12 characters per inch), *pica* (10 characters per inch), or *compressed* (as many characters per inch as possible).

Character set choice
When the **lp** print service alerts an administrator to mount this form, it will also mention that the print wheel ***print-wheel*** should be used on those printers that take print wheels. If printing with this form is to be done on a printer that has selectable or loadable character sets instead of print wheels, the interface programs provided with the **lp** print service will automatically select or load the correct character set. If **mandatory** is appended, a user is not allowed to select a different character set for use with the form; otherwise, the character set or print wheel named is a suggestion and a default only.

Ribbon color
When the **lp** print service alerts an administrator to mount this form, it will also mention that the color of the ribbon should be ***ribbon-color***.

Comment The **lp** print service will display the ***comment*** unaltered when a user asks about this form (see **lpstat**(C)).

Alignment pattern
When mounting this form, an administrator can ask that the ***content*** be repeatedly printed as an aid in correctly positioning the preprinted form. The optional ***content-type*** defines the type of printer for which ***content*** had been generated. If ***content-type*** is not given, **simple** is assumed. Note that the ***content*** is stored as given, and will be readable only by the user *lp*.

When an existing form is changed with this command, items missing in the new information are left as they were. When a new form is added with this command, missing items will get the following defaults:

Page Length: **66**
Page Width: **80**
Number of Pages: **1**
Line Pitch: **6**
Character Pitch: **10**
Character Set Choice: **any**
Ribbon Color: **any**
Comment: (no default)
Alignment Pattern: (no default)

Use one of the following commands to define the form:

/usr/lib/lpforms -f ***formname*** **-F** ***filename***
/usr/lib/lpforms -f ***formname*** **-**

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ PPforms ➪ Configure

Provide the pathname for the form as directed. The first command gets the form definition from a file; the second command gets the form definition from you through the standard input. The ***formname*** can be anything you choose.

If you need to change a form, just re-enter one of the same commands. You need only give the changed information; information you leave out stays the same.

Deleting a form

The **-x** option is used to delete the form specified in ***formname*** from the **lp** print service. Use the following command to remove a form:

/usr/lib/lpforms -f ***formname*** **-x**

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ PPforms ➪ Remove

Listing form attributes

The **-l** option is used to list the attributes of the existing form specified by ***formname***. The attributes listed are those described under "Adding or changing a form". Because of the potentially sensitive nature of the alignment pattern, only the administrator can examine the form with this command. Other people can use the **lpstat**(C) command to examine the non-sensitive part of the form description.

Use one of the following commands to examine a defined form:

/usr/lib/lpforms -f ***formname*** **-l**
/usr/lib/lpforms -f ***formname*** **-l >***filename*
lpstat -f ***formname***
lpstat -f ***formname*** **-l**

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ PPforms ➪ List

The first two commands present the definition of the form; the second command captures this definition in a file, which can later be used to redefine the form if you inadvertently remove the form from the print service. The last two commands present the status of the form, with the second of the two giving a long form of output similar to the output of **lpforms -l**.

Allowing and denying access to a form

The **lp** print service keeps two lists of users for each form, an *allow-list* and a *deny-list* of people denied access to the form. With the **-u allow:** option, the users listed are added to the allow-list and removed from the deny-list. With the **-u deny:** option, the users listed are removed from the allow-list and added to the deny-list.

The rules are as follows:

- An allow list contains those users allowed to use the form. A deny list contains those users denied access to the form.
- If the allow list is not empty, the deny list is ignored. If the allow list is empty, the deny list is used. If both lists are empty, there are no restrictions on who can use the form.
- Putting **any** or **all** into the allow list allows everybody to use the form; putting **any** or **all** into the deny list denies everybody use of the form, except the user *lp* and the super user *root*.

You can define who can use the form using the following commands:

/usr/lib/lpforms -f *formname* **-u allow:***user-list*
/usr/lib/lpforms -f *formname* **-u deny:***user-list*

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ PPforms ➪ Users

The ***user-list*** is a list of names of users separated by a comma or space. If you use spaces to separate the names, enclose the entire list (including the **allow:** or **deny:** but not the **-u**) in quotes. The first command adds the names to the allow list and removes them from the deny list. The second command adds the names to the deny list and removes them from the allow list. Using **allow:all** allows everybody; using **deny:all** denies everybody. If you do not add user names to the allow or deny lists, the print service assumes that everybody can use the form.

Alerting to mount forms

The second variation of the **lpforms** command is used to arrange for the alerting to mount forms on a printer.

When $integer_1$ print requests needing the preprinted form ***formname*** become queued because no printer satisfying all the needs of the requests has the form mounted, and for as long as this condition remains, an alert is sent to the administrator every $integer_2$ minutes until the form is mounted on a qualifying printer. If the ***formname*** is all, the alerting defined in this command applies to all existing forms. No alerting is done for a backlog of print requests needing a form if the administrator does not use this option.

You can choose one of several ways to receive an alert:

- You can receive an alert via electronic mail (see **mail**(C)).
- You can receive an alert written to whatever terminal on which you are logged in (see **write**(C)).
- You can receive an alert through a program of your choice.
- You can receive no alerts.

The method for sending the alert depends on the value of the **-A** option.

write The message is sent via **write**(C) to the terminal on which the administrator is logged in when the alert arises. If the administrator is logged in on several terminals, one is chosen arbitrarily.

mail The message is sent via mail to the administrator who issues this command.

The message sent appears as follows:

```
The form form-name needs to be mounted on the printer(s).
printer-list (integer r3 requests)
integer4 print request awaits this form.
Use the ribbon-color ribbon.
Use the print-wheel print wheel, if appropriate.
```

The printers listed are those that the administrator had earlier specified were candidates for this form. The number (*integer*$_3$) listed next to each printer is the number of requests eligible for the printer. The number (*integer*$_4$) shown after the printer list is the total number of requests awaiting the form. It will be less than the sum of the other numbers if some requests can be handled by more than one printer. The *ribbon-color* and *print-wheel* are those given in the form description. The last line in the message is given even if none of the printers listed use print wheels, because the administrator may choose to mount the form on a printer that does use a print wheel.

Where any color ribbon or any print wheel can be used, the statements above will read:

```
Use any ribbon.
Use any print-wheel.
```

shell-command

The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. Note that the **mail** and **write** values for the **-A** command are equivalent to the values **mail** *username* and **write** *username*, respectively, where *username* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the **su**(C) command to change to another user ID. If the **su** command has been used to change the user ID, then the **username** for the new ID is used.

If you elect to receive no alerts, you are responsible for checking to see if any print requests have not printed because the proper form is not mounted.

In addition to the method of alerting, you can also set the number of requests that must be queued before you are alerted, and you can arrange for repeated alerts every few minutes until the form is mounted. You can choose the rate of repeated alerts, or you can choose to receive only one alert per form.

To arrange for alerting to the need to mount a form, enter one of the following commands:

/usr/lib/lpforms -f ***formname*** **-A mail -Q** ***integer*** **-W** ***minutes***
/usr/lib/lpforms -f ***formname*** **-A write -Q** ***integer*** **-W** ***minutes***
/usr/lib/lpforms -f ***formname*** **-A** ***'command'*** **-Q** ***integer*** **-W** ***minutes***
/usr/lib/lpforms -f ***formname*** **-A none**

Δ **sysadmsh** users select: Printers ➪ Auxiliary ➪ PPforms ➪ Alerts ➪ Specify

The first two commands direct the print service to send you a mail message or to write the message directly to your terminal, respectively, for each alert. The third command directs the print service to run ***command*** for each alert. The shell environment currently in effect when you enter the third command is saved and restored for the execution of ***command***; this includes the environment variables, user and group IDs, and current directory.

The fourth command directs the print service not to send you an alert when the form needs to be mounted. ***integer*** is the number of requests that need to be waiting for the form, and ***minutes*** is the number of minutes between repeated alerts.

If you want mail sent or a message written to another person when a printer fault occurs, you must use the third command listed. Use the **-A 'mail** ***username'*** or **-A 'write** ***username'*** option.

If ***formname*** is **all** in any of the previous commands, the alerting condition applies to all forms.

If you do not define an alert method for a form, you do not receive an alert for it. If you do define a method but do not give the **-W** option, you are alerted once for each occasion.

Listing the current alert

The following **lpforms** syntax is used to list the type of the alert for the specified form:

/usr/lib/lpforms -f *formname* -A *list*

No change is made to the alert. If ***formname*** is recognized by the **lp** print service, one of the following lines is sent to the standard output, depending on the type of alert for the form.

```
When integer are queued:
alert with shell-command every integer minutes

When integer are queued:
write to username every integer minutes

When integer are queued:
mail to username every integer minutes

No alert
```

The phrase "every ***integer*** minutes" is replaced with "once" if $integer_2$ (the **-W** $integer_2$) is 0.

Terminating an active alert

The **quiet** option is used to stop messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form has been mounted and then unmounted, messages will again be sent when the queue size reaches $integer_1$ pending requests.

Once you start receiving repeated alerts, you can direct the print service to stop sending you alerts for the current case only by giving the following command:

/usr/lib/lpforms -f *formname* -A quiet

Δ **sysadmsh** users select: Printers ⇨ Admin ⇨ PPforms ⇨ Alerts ⇨ Terminate

Once the form is mounted and unmounted again, alerts start again if too many requests are waiting. Alerts also restart if the number of requests waiting falls below the **-Q** threshold and then rises up to the **-Q** threshold again, as when waiting requests are canceled or if the type of alerting is changed.

Removing an alert definition

No messages will be sent until the **none** option is given again with a different ***alert-type***. This can be used to permanently stop further messages from being sent.

See also

lp(C), **lpadmin**(ADM), **terminfo**(F)

Authorization

Permission to use this utility requires the *lp* authorization.

lpsched, lpshut, lpmove

start/stop the print service and move requests

Syntax

/usr/lib/lpsched

/usr/lib/lpsched -q ***integer***

/usr/lib/lpsched -a ***integer***

/usr/lib/lpsched -p ***integer***

/usr/lib/lpsched -s ***integer***

/usr/lib/lpshut

/usr/lib/lpmove ***requests dest***

/usr/lib/lpmove ***dest1 dest2***

Description

lpsched - start the print service

lpshut - stop the print service

lpmove - move print requests

lpsched starts the LP print service; this can be done only by *root*.

lpshut shuts down the print service. All printers that are printing at the time **lpshut** is invoked will stop printing. When **lpsched** is started again, requests that were printing at the time a printer was shut down will be reprinted from the beginning.

lpmove moves requests that were queued by **lp** between LP destinations. 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***; **lp** will then reject any new requests for ***dest1***.

Note that when moving requests, **lpmove** never checks the acceptance status (see **accept**(ADM)) of the new destination. Also, the request ID of the moved request is not changed so that users can still find their requests. The **lpmove** command will not move requests that have options (content type, form required, and so on) that cannot be handled by the new destination.

-q ***integer*** Specify the number of request structures you want to allocate.

-a ***integer*** Specify the number of alert structures you want to allocate. By default, 40 empty alert structures are allocated in addition to one for each printer or form on the system. Structures will always be allocated for existing printers and forms. You can choose, however, to have more or fewer than the 40 extra, by using the **-a** option. For example, if you want only as many alert structures as you have printers and forms on your system, enter the following command: **lpsched -a 0**.

-p ***integer*** Specify the number of print status structures you want to allocate. By default, 25 empty printer status structures are allocated in addition to one for each printer on the system. Structures will always be allocated for existing printers. You can choose, however, to have more or fewer than the 40 extra, by using the **-p** option.

-s ***integer*** Specify the number of slow filters per printer that can be run simultaneously.

Notes

By default, the directory */usr/spool/lp* is used to hold all the files used by the LP print service. This can be changed by setting the **SPOOLDIR** environment variable to another directory before running **lpsched**. If you do this, you should populate the directory with the same files and directories found under */usr/spool/lp*; the LP print service will not automatically create them. Also, the **SPOOLDIR** variable must then be set before any of the other LP print service commands are run.

Files

*/usr/spool/lp/**

See also

accept(ADM), **enable**(C), **lp**(C), **lpstat**(C), **lpadmin**(ADM)

lpsh

menu driven lp print service administration utility

Syntax

/usr/lib/sysadm/lpsh

Description

lpsh is the screen interface invoked by the **sysadmsh**(ADM) Printers selection to administer the print service. The interface performs all of the required **lp** print service functions that require system administrator authorization, **lp**.

The program allows the administrator to perform any of the following tasks:

- configure the **lp** print service to describe printers and devices
- administer filters to be used with the **lp** print service
- administer forms to be used with the **lp** print service
- start the **lp** print service
- shut down the **lp** print service
- move print requests between printer destination
- cancel print requests
- allow destinations to accept or reject print requests
- set the printing queue priorities that can be assigned to jobs submitted by users of the **lp** print service
- enable or disable printers

See also

accept(ADM), **atcronsh**(ADM), **auditsh**(ADM), **authsh**(ADM), **backupsh**(ADM), **enable**(C) **lp**(C), **lpadmin**(ADM), **lpfilter**(ADM), **lpforms**(ADM), **lpsched**(ADM), **lpusers**(ADM), **sysadmsh**(ADM)

Notes

Invoking the **lpsh** directly is not recommended; use the **sysadmsh** Printers selection.

Value added

lpsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

lpusers

set printing queue priorities

Syntax

/usr/lib/lpusers -d ***priority-level***

/usr/lib/lpusers -q ***priority-level*** **-u** ***user-list***

/usr/lib/lpusers -u ***user-list***

/usr/lib/lpusers -q ***priority-level***

/usr/lib/lpusers -l

Description

The **lpusers** command is used to set limits to the queue priority level that can be assigned to jobs submitted by users of the LP print service.

The first form of the command (with **-d**) sets the system-wide priority default to ***priority-level***, where ***priority-level*** is a value of 0 to 39, with 0 being the highest priority. If a user does not specify a priority level with a print request (see **lp**(C)), the default priority is used. Initially, the default priority level is 20.

The second form of the command (with **-q** and **-u**) sets the default highest ***priority-level*** (0-39) that the users in ***user-list*** can request when submitting a print request. Users that have been given a limit cannot submit a print request with a higher priority level than the one assigned, nor can they change a request already submitted to have a higher priority. Any print requests with priority levels higher than allowed will be given the highest priority allowed.

The third form of the command (with **-u**) removes the users from any explicit priority level and returns them to the default priority level.

The fourth form of the command (with **-q**) sets the default highest priority level for all users not explicitly covered by the use of the second form of this command.

The last form of the command (with **-l**) lists the default priority level and the priority limits assigned to users.

See also

lp(C)

majorsinuse

display the list of major device numbers currently specified in the mdevice file

Syntax

/etc/conf/cf.d/majorsinuse

Description

This script searches the *mdevice* file and displays a list of the major device numbers already in use.

When installing a device driver with the Link Kit, you can use **majorsinuse** to find an available major device number for the driver. When you invoke the **configure**(ADM) program to modify the system configuration files with the new driver information, use the **-m** option to indicate the major device number of the driver.

The **-j** option to **configure** performs a function similar to that of **majorsinuse**. If you give the **-j** option with the next major keyword, **configure** tells you the next available major device number.

File

/etc/conf/cf.d/mdevice

See also

configure(ADM), **mdevice**(F)

Value added

majorsinuse is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

makekey

generate an encryption key

Syntax

/usr/lib/makekey

Description

makekey improves the usefulness of encryption schemes by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way that is intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first 8 input bytes (the "input key") can be arbitrary ASCII characters. The last 2 input bytes (the "salt") are best chosen from the set of digits, dot ".", slash "\", and uppercase and lowercase letters. The salt characters are repeated as the first 2 characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the "output key".

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as the key, a constant string is fed into the machine and recirculated. The 64 bits that come out are distributed into the 66 output key bits in the result.

See also

passwd(FP)

menumerge

merge sysadmmenu(F) files

Syntax

menumerge ***menulist addlist***

Description

This program is intended for developers who wish to customise the extensible menus found in **sysadmsh**(ADM). These new menu entries would perhaps allow the customer to run configuration programs particular to the product being supplied. The **menumerge** utility would usually be run once from an installation script, merging in the extra menu entries from the file ***addlist***, present on the installation media, into the ***menulist*** file chosen from the following list;

menulist file	Extensible menu area
/usr/lib/sysadm/.menu-execute	System ➪ Execute
/usr/lib/sysadm/.menu-hardware	System ➪ Hardware
/usr/lib/sysadm/.menu-kernel	System ➪ Configure ➪ Kernel
/usr/lib/sysadm/.menu-network	System ➪ Configure ➪ Network
/usr/lib/sysadm/.menu-other	System ➪ Configure ➪ Other

The ***addlist*** file must be in the same *sysadmmenu*(F) format as the ***menulist*** file. The suggested method for producing and testing such an ***addlist*** file is as follows:

1. From the shell, rename the file **$HOME**/*.sysadmmenu*, if it exists.
2. Start **sysadmsh**, and select Systemu ➪ Configure ➪ Menus .
3. Select the User menu entry for configuration.
4. Edit the User menu entry, such that it contains all the menu entries that are to be added to the above ***menulist*** argument.
5. When you have finished editing the menu area, check that the menu works by selecting the User entry from the top level of **sysadmsh**. Repeat from step 2 where necessary.
6. When the menu area is correct, quit from **sysadmsh**.
7. Move the file **$HOME**/*.sysadmmenu* into the development source tree, or onto the installation media, and add the appropriate **menumerge** command to the installation script.

8. If need be, replace the original **$HOME/***.sysadmmenu*.

9. Run the installation script, and test that the appropriate extensible menu areas in **sysadmsh** contain the new merged entries.

See also

sysadmmenu(F), **sysadmsh**(ADM)

System Administrator's Guide

Value added

menumerge is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

mkdev

call scripts to add peripheral devices

Syntax

/etc/mkdev aio
/etc/mkdev bitpad
/etc/mkdev cdrom
/etc/mkdev dos
/etc/mkdev dda
/etc/mkdev fd
/etc/mkdev fs [*device file*]
/etc/mkdev graphics
/etc/mkdev hd [*drivenum*] [*ctrlnum*] [[*lun*] [*hatype*]]
/etc/mkdev high-sierra
/etc/mkdev layers
/etc/mkdev lp
/etc/mkdev mmdf
/etc/mkdev mouse
/etc/mkdev parallel
/etc/mkdev ptty
/etc/mkdev serial
/etc/mkdev shl
/etc/mkdev streams
/etc/mkdev tape
/etc/mkdev vpixld

Description

mkdev creates the device file(s) associated with a peripheral device. Based on the argument supplied, the **mkdev** command calls a script found in the directory */usr/lib/mkdev*. (There may be scripts found in this directory that are specific to a given application or software package that are not documented here.) If no arguments are listed, **mkdev** prints a usage message.

mkdev aio adds support for asynchronous disk I/O to the kernel.

mkdev bitpad configures supported bitpad devices.

mkdev cdrom adds CD-ROM support to the kernel.

mkdev dos initializes necessary devices and configures the system to support mounted DOS filesystems.

mkdev dda adds direct device access support for SCO VP/ix to the kernel.

mkdev fd creates bootable, root and filesystem floppy disks.

Several floppies can be created during a single **mkdev fd** session, but **mkdev** does not display a prompt to remove the first floppy and insert the next one. Insert the next floppy when **mkdev** prompts "Would you like to format the floppy first? (y/n)."

mkdev fs performs the system maintenance tasks required to add a new filesystem to the system after the device is configured using **mkdev hd**. **mkdev fs** creates the *mountpoint* and *lost+found* directory, reserves slots in the *lost+found* directory, (if either already exist, they are used unmodified) and modifies */etc/checklist*, */etc/default/filesys* and */etc/default* to check using **fsck**(ADM) and mount the filesystem using **mount**(ADM) or **mnt**(C) as appropriate.

mkdev graphics configures graphics adapters for use with applications that can take advantage of them.

mkdev hd creates device files for use with a peripheral hard disk. The device files for an internal hard disk already exist. It invokes the following utilities (as appropriate): **dparam**(ADM), **badtrk**(ADM), **fdisk**(ADM), and **divvy**(ADM). **mkdev hd** includes an extended syntax for use on multiple controllers. These syntax extensions use numbers to refer to the disk and controller numbers.

In addition, the codes ST506-, and IDA- SCSI- can be used to refer to the controller/adapter number, as shown in the examples below.

ST506 disks will install with one of the following commands:

mkdev hd 0 0 (or ST506-0)
first disk on first controller
mkdev hd 1 0 (or ST506-0)
second disk on first controller
mkdev hd 0 1 (or ST506-1)
first disk on second controller
mkdev hd 1 1 (or ST506-1)
second disk on second controller

ESDI disks will install with one of the following commands:

mkdev hd 0 0
first disk on controller (root disk)
mkdev hd 1 0
second disk on controller

The SCSI syntax is as follows:

mkdev hd ***id ha lun hatype***

where:

id	is a number from 0-7
ha	is SCSI-0 to SCSI-7
lun	is a number from 0-7
hatype	is the type of host adapter (listed in **/etc/default/scsihas**)

For example, the following command configures the second disk on the first Adaptec SCSI adapter:

mkdev hd 1 SCSI-0 0 ad

mkdev hd must be invoked twice to install a SCSI disk. The first time, the kernel will be reconfigured to support the new disk. The second time, the disk will be initialized. Use the same **mkdev hd** arguments both times.

Disks attached to Compaq IDA/Intelligent Array Expansion controllers are added with using the same syntax as standard disk support. The controller can be specified by number or IDA-*n*. Up to six IDA controllers are supported (0-5 or IDA-0 through IDA-5).

mkdev high-sierra
configures a mountable filesystem found on a CD-ROM drive.

mkdev layers adds support for serial terminals with AT&T windowing capabilities to the kernel.

mkdev lp adds or modifies a printer configuration.

mkdev mmdf interactively alters the MMDF configuration.

mkdev mouse initializes necessary devices and configures the system to use any supported mouse.

mkdev parallel allows the configuration of multiple parallel ports.

mkdev ptty adds pseudo-ttys to the system.

mkdev serial creates device files for use with serial cards. The device files for the first and second ports already exist. Additional device files must be created for the ports added when expansion cards are added to the system.

mkdev shl initializes necessary devices and configures kernel parameters associated with the number of shell layers sessions available on the system.

mkdev streams configures the kernel for streams support.

mkdev tape configures the tape driver in preparation for linking a new kernel that includes tape support. It adds a standard quarter-inch cartridge tape driver, a mini-cartridge tape driver, a QIC-40/80 tape driver or a SCSI tape driver.

When configuring a cartridge tape drive, the current driver configurations can be displayed, and changed if necessary. A zero in any of the fields means the driver automatically detects the type of tape device installed and uses the built-in values for that device. If the autoconfiguration values are not correct for your drive, refer to your hardware manual for the correct values, reconfigure the driver and relink the new kernel.

mkdev tape can also be used to remove a tape driver from the existing kernel.

Once the driver is configured, you are prompted for re-linking the kernel. The appropriate devices in */dev* are created.

The various scripts prompt for the information necessary to create the devices.

mkdev vpixld adds the line discipline for SCO VP/ix to the system.

Files

*/usr/lib/mkdev/**	location of scripts invoked by mkdev
/etc/default/scsihas	list of supported SCSI host adapters

The following are additional support scripts invoked indirectly by **mkdev**:

/usr/lib/mkdev/.scsi	SCSI support configuration routines
/usr/lib/mkdev/ida	Compaq-specific scripts
/usr/lib/mkdev/.hdfuncs	
/usr/lib/mkdev/.cpqs.tape	

See also

badtrk(ADM), **divvy**(ADM), **dparam**(ADM), **fd**(HW), **fdisk**(ADM), **filesys**(F), **format**(C), **hd**(HW), **lp**(HW), **mkfs**(ADM), **mknod**(C), **mount**(ADM), **serial**(HW), **usemouse**(C), **tape**(HW)

The *System Administrator's Guide* has chapters devoted to the installation of most peripheral devices.

Value added

mkdev is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

mkfs

construct a filesystem

Syntax

/etc/mkfs [**-y** | **-n**] [**-f** *fstype*] *filsys blocks* [: *inodes*] [*gap blocks/cylinder*] [*filesystem-specific options*]

/etc/mkfs [**-y** | **-n**] *filsys proto* [*gap blocks/cylinder*] [*filesystem-specific options*]

XENIX filesystem options: [**-s** *blocks* [: *inodes*]]

UNIX filesystem options: [**-b** *blocksize*]

AFS filesystem options: [**-E**] [**-C** *clustersize*]

Description

mkfs constructs a filesystem by writing on the special file ***filsys***, according to the directions found in the remainder of the command line. **mkfs** is actually a front-end that invokes the appropriate version of **mkfs** according to the filesystem type. The **-f** option specifies the filesystem type, which can be one of the following:

AFS (Acer Fast Filesystem)
S51K (UNIX)
XENIX
DOS

In addition, you can create an EAFS (Extended Acer Fast Filesystem, which supports symbolic links and long filenames) by specifying the **-E** option. Note that the EAFS type cannot be specified using the **-f** option.

Standard options

If it appears that the special file contains a filesystem, operator confirmation is requested before overwriting the data. The **-y** "yes" option overides this, and writes over any existing data without question. The **-n** option causes **mkfs** to terminate without question if the target contains an existing filesystem. The check used is to read block one from the target device (block one is the super-block) and see whether the bytes are the same. If they are not, this is taken to be meaningful data and confirmation is requested.

If the second argument to **mkfs** is a string of digits, the size of the filesystem is the value of ***blocks*** interpreted as a decimal number. This is the number of ***physical*** (512-byte) disk blocks the filesystem will occupy. If the number of inodes is not given, the default is approximately the number of ***logical*** blocks divided by 4. **mkfs** builds a filesystem with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, **mkfs** assumes it to be a prototype file ***proto***, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1       /stand/diskboot
2       4872 110
3       d--777 3 1
4       usr     d--777 3 1
5               sh      ---755 3 1 /bin/sh
6               ken     d--755 6 1
7                       $
8               b0      b--644 3 1 0 0
9               c0      c--644 3 1 0 0
10              $
11      $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512-byte) blocks the filesystem is to occupy and the number of inodes in the filesystem.

Lines 3-9 tell **mkfs** about files and directories to be included in this filesystem.

Line 3 specifies the *root* directory.

Lines 4-6 and 8-9 specify other directories and files.

The "$" on line 7 tells **mkfs** to end the branch of the filesystem it is on, and continue from the next higher directory. The "$" on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **-bcd** to specify regular, block special, character special and directory files, respectively. The second character of the mode 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 3-digit octal number giving the owner, group, and other read, write, execute permissions (see **chmod**(C)).

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

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

The ***gap blocks/cylinder*** argument in both forms of the command specifies the rotational gap and the number of blocks/cylinder. This number must be greater than zero and less than 1500, or else the default of 400 blocks/cylinder is used.

XENIX filesystem options

The **-s** option is a command-line override of the size and number of inodes in the ***proto*** file.

UNIX filesystem options

The **-b** ***blocksize*** option specifies the logical block size for the filesystem. The logical block size is the number of bytes read or written by the operating system in a single I/O operation. Valid values for ***blocksize*** are 512, and 1024. The default is 1024. If the **-b** option is used, it must appear last on the command line.

AFS filesystem options

The **-E** option creates an Extended Acer Fast Filesystem, which supports symbolic links and long file names.

The **-C** ***clustersize*** option specifies the cluster size for the filesystem. This only applies to AFS; if this is included on the command line, the filesystem created will be AFS regardless of the other options used. The **-C** option must be the last option on the command line.

File

*/etc/vtoc/**

See also

chmod(C), **dir**(FP), **filesystem**(FP)

Notes

With a prototype file, the maximum size of a file that can be copied in is dependent on the type of filesystem being created.

The directory */etc/fscmd.d/**type*** contains programs for each filesystem type; **mkfs** invokes the appropriate binary.

mmdf

route mail locally and over any supported network

Description

The operating system uses MMDF (the Multi-channel Memorandum Distribution Facility) to route mail locally and over Micnet, UUCP, or other networks that provide MMDF support. The **custom**(ADM) utility installs MMDF and configures a basic system for sending mail on a local machine.

MMDF is a very versatile and configurable mail routing system. MMDF configuration begins with the */usr/mmdf/mmdftailor* file, which defines the machine and domain names, the various tables (alias, domain, channel), and other configuration information. To change the configuration of MMDF on your system, you can log in as *mmdf* and edit the configuration files. Whenever you change MMDF alias or routing information in any way, you must rebuild the hashed database.

Files

/usr/mmdf/mmdftailor
/usr/mmdf/table/alias.list
/usr/mmdf/table/alias.user
/usr/mmdf/table/.chn*
/usr/mmdf/table/.dom*
*/usr/spool/mail/**
/usr/spool/mmdf/...

See also

dbmbuild(ADM), **deliver**(ADM), **mmdfalias**(ADM), **mmdftailor**(F), **mnlist**(ADM), **submit**(ADM), **tables**(F), **uulist**(ADM)

"Setting up electronic mail" in the *System Administrator's Guide*

Value added

mmdf is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

mmdfalias

convert XENIX-style aliases file to MMDF format

Syntax

/usr/mmdf/table/tools/mmdfalias

Description

mmdfalias is a conversion utility to produce MMDF-compatible alias files from the XENIX-format aliases file. **mmdfalias** also splits the converted contents of */usr/lib/mail/aliases* into two MMDF files containing list-type aliases and aliases that map users to machines.

After installing MMDF with **custom**, restore */usr/lib/mail/aliases* from backup tape. Place the following line in the file to indicate where the list aliases end and the mapping aliases begin:

```
# user-to-machine mapping
```

Log in as *mmdf* and run the **/usr/mmdf/table/tools/mmdfalias** conversion script from the */usr/mmdf/table* directory. You now have two MMDF files, *alias.list* and *alias.user*, in the current directory.

After creating these files in */usr/mmdf/table*, you must rebuild the MMDF hashed database. While logged in as *mmdf*, run **dbmbuild** from */usr/mmdf/table*.

Files

/usr/lib/mail/aliases
/usr/mmdf/table/alias.list
/usr/mmdf/table/alias.user

See also

dbmbuild(ADM), **tables**(F)

"Setting Up Electronic Mail" in the *System Administrator's Guide*

Value added

mmdfalias is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

mnlist

convert a XENIX-style Micnet routing file to MMDF format

Syntax

/usr/mmdf/table/tools/mnlist

Description

mnlist is a conversion utility to produce MMDF-compatible Micnet routing files from the XENIX-format Micnet routing file.

After installing MMDF with **custom**(ADM), restore */usr/lib/mail/top* from backup media. Log in as *mmdf* and run the conversion script **/usr/mmdf/table/tools/mnlist** from the */usr/mmdf/table* directory. You now have a Micnet channel file, *micnet.chn*, in the current directory.

After creating these files in */usr/mmdf/table*, you must rebuild the MMDF hashed database. While logged in as *mmdf*, run **dbmbuild**(ADM) from */usr/mmdf/table*.

Files

/usr/lib/mail/top
/usr/mmdf/table/micnet.chn

See also

dbmbuild(ADM), **tables**(F)

"Setting up electronic mail" in the *System Administrator's Guide*

Value added

mnlist is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

mount

mount and unmount a file structure

Syntax

/etc/mount [-v] [-r] [-f *fstyp*] *special-device directory*

/etc/umount *special-device*

Description

mount - mount a file structure

umount - unmount a file structure

mount announces to the system that a removable file structure is present on ***special-device.*** The file structure is mounted on ***directory.*** The ***directory*** must already exist; it becomes the name of the root of the newly mounted file structure. ***directory*** should be empty. If ***directory*** contains files, they will appear to have been removed while the ***special-device*** is mounted and reappear when the ***special-device*** is unmounted.

The **mount** and **umount** commands maintain a table of mounted devices. If **mount** is invoked without any arguments, it displays the name of each mounted device, and the directory on which it is mounted, whether the file structure is read-only, and the date it was mounted.

The **-f *fstyp*** option indicates that ***fstyp*** is the type of filesystem to be mounted. If this argument is omitted, it defaults to the same filesystem type as the root directory.

The optional **-r** argument indicates that the file is to be mounted read-only. Physically write-protected filesystems, such as floppy disks with write-protect tabs, must be mounted in this way or errors occur when access times are updated, whether or not any explicit write is attempted.

The optional **-v** argument displays mount information verbosely.

umount removes the removable filesystem on device ***special-device.*** Any pending I/O for the filesystem is completed and the file structure is marked as clean.

Files

/etc/mnttab	Mount table
/etc/default/filesys	Filesystem data

See also

default(F), **mnt**(C), **mount**(S), **mnttab**(F), **setmnt**(ADM), **umount**(ADM)

Diagnostics

mount issues a warning if ***directory*** does not match the "s_fname" field in the superblock of the filesystem to be mounted. The first six characters in the last component of ***directory*** are compared with the name in "s_fname". (For example, mounting a filesystem named *spool* on */usr/spool* won't cause a warning message, but mounting the same filesystem on */mnt* will.)

Busy filesystems cannot be dismounted with **umount**. A filesystem is busy if it contains an open file or some user's working directory.

Notes

Only the super user can use the **mount** command.

Some degree of validation is done on the filesystem, however it is generally unwise to mount corrupt filesystems.

If the **mount** command is invoked without a target directory, but with a valid filesystem as an argument, **mount** attempts to mount the system on the directory named in its entry in */etc/default/filesys*. For further information, refer to **filesys**(F).

Note that when the system is in single-user mode, the commands that look in */etc/mnttab* for default arguments (for example **df**(C), **ncheck**(ADM), **quot**(C), **mount**, and **umount**) give either incorrect results (due to a corrupt **mnttab**(F) left over from a non-shutdown stoppage) or no results (due to an empty */etc/mnttab* from a **shutdown** stoppage).

When in multi-user mode, this is not a problem; the */etc/rc2* scripts initialize */etc/mnttab* to contain only */dev/root* and subsequent mounts update it appropriately.

The **mount** and **umount** commands use a lock file to guarantee exclusive access to */etc/mnttab*. The other commands which read it (those mentioned above) do not, so it is possible that if they are invoked while a file system is being mounted or unmounted they will return incorrect values. This is not a problem in practice since **mount** and **umount** are not frequent operations.

When mounting a filesystem on a floppy disk you need not use the same ***directory*** each time. However, if you do, the full pathnames for the files are consistent with each use.

Always **unmount** filesystems on floppy disks before removing them from the floppy drive. Failure to do so requires running **fsck**(ADM) the next time the disk is **mounted**.

The directory */etc/fscmd.d/TYPE* contains programs for each filesystem type; **mount** or **umount** invoke the appropriate binary.

Standards conformance

mount is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

mountall, umountall

mount, unmount multiple file systems

Syntax

/etc/mountall [-a]

/etc/umountall [-k]

Description

mountall - mount multiple file systems

umountall - unmount multiple file systems

These commands can be executed only by the super user.

The **mountall** command is used to mount filesystems according to */etc/default/filesys*.

Before each filesystem is mounted, it is checked using **fsstat**(ADM) to see if it appears to be mountable. The default behavior, if the file system does not appear to be mountable, is to check it using **fsck**(ADM) before the mount is attempted. (This behaviour may be modified by */etc/default/filesys* file — see **filesys**(F).)

mountall is called with the **-a** when the system autoboots. The **-a** flag causes output messages to be written to the file */etc/bootlog*, and later mailed to the system administrator (see **boot**(HW)).

The **umountall** command causes all mounted file systems except the root filesystem to be unmounted. The **-k** option sends a **SIGKILL** signal, via **fuser**(ADM), to processes that have files open.

File

/etc/default/filesys	filesystem table

See also

boot(HW), **filesys**(F), **fsck**(ADM), **fsstat**(ADM), **fuser**(ADM), **mount**(ADM), **signal**(S)

Diagnostics

No messages are printed if the filesystems are mountable and clean.

Error and warning messages come from **fsck**(ADM), **fsstat**(ADM), and **mount**(ADM).

mvdir

move a directory

Syntax

/etc/mvdir ***dirname newdirname***

Description

mvdir moves directories within a filesystem. The directory (***dirname***) must be a directory. If there is already a directory or file with the same name as ***newdirname***, **mvdir** fails.

Neither name may be a subset of the other. For example, you cannot move a directory named */x/y* to */x/y/z*, and vice versa.

See also

mkdir(C)

Standards conformance

mvdir is conformant with:

AT&T SVID Issue 2.

ncheck

generate names from inode numbers

Syntax

/etc/ncheck [**-i** *numbers*] [**-a**] [**-s**] [*filesystem*]

Description

ncheck with no argument generates a pathname and inode number list of all files on the set of filesystems specified in */etc/mnttab*. The two characters "**/.**" are appended to the names of directory files.

The options are as follows:

-i limits the report to only those files whose inode numbers follow.

-a allows printing of the names . and .., which are ordinarily suppressed.

-s limits the report to special files and files with set-user-ID (setuid) mode. This option may be used to detect violations of security policy.

A single *filesystem* may be specified rather than the default list of mounted filesystems.

File

/etc/mnttab

See also

fsck(ADM), **sort**(C)

Diagnostics

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

Notes

See "Notes" under **mount**(ADM).

The directory */etc/fscmd.d/TYPE* contains programs for each filesystem type; **ncheck** invokes the appropriate binary.

netconfig

configure networking products

Syntax

netconfig [-racmCeLlndvst] [*chain*] [*element*]

Description

netconfig configures and enables network products. **netconfig** provides a standard way to combine networking products together to form a networking system from compatible networking components. By default, it is only executable by root.

netconfig assembles compatible sets of networking products into functional groups called chains. Configuring (adding) a chain will do everything necessary to enable the functions of the component products in the chain within a single command. Deconfiguring (removing) a chain through **netconfig** does everything necessary to disable the function of the component parts of the chain also in a single command.

A chain consists of a top level product, and one or more lower layer networking products that together produce a functional networking system.

netconfig can be used interactively through a menu-driven interface, or at the command line. When used interactively, **netconfig** presents the user with a list of the currently configured chains as part of the main menu.

netconfig, when used non-interactively, is designed to be used as an engine underneath a higher level User Interface program, whether graphical or character oriented.

The **netconfig** menu has the following options:

Add a Chain — Enable (configure) the specified chain. Chains are specified one element at a time. First, the top level product is specified, and then, on progressive menus, each lower element in the chain is specified until a complete chain is assembled. Then, after a final verification from the user, the chain is enabled. During configuration, **netconfig** invokes each product's initialization procedure which may require you to refer to each product's manual for descriptions of any actions that the installation procedures ask you to perform.

Remove a Chain — This option is not displayed unless one or more chains are configured. Removing a chain will disable the specified chain. The user is given a choice of all currently configured chains to remove. You must refer to the individual product's manual for descriptions of any actions that the removal procedures ask you to perform.

Reconfigure an Element
: This menu entry is only displayed when a chain containing one or more elements exists. Each element in a chain has some configuration associated with it. Usually, during the add chain function, this information is supplied by the user. If you want to reconfigure a part of a chain without removing and re-adding the entire chain then reconfigure is the option to use. Not all chain elements support reconfigure, the **netconfig** menus will inform the user which chain items support this feature.

Quit
: Exit from **netconfig**.

netconfig and **netconfig**-compatible products place their information in the directory */usr/lib/netconfig*. The */usr/lib/netconfig* directory has the following subdirectories:

info
: This directory contains files that describe the product information.

init
: This directory contains all the initialization scripts and all the default value files. The default files have the same name as the initialization scripts plus the *.def* extension.

remove
: This directory contains product removal scripts.

rc0
: This directory contains the stop script for each product. The stop script is copied to the */etc/rc0.d* directory when the product is added using **netconfig**. The script is removed when the product is removed. The scripts in this directory must be named ***KXXproduct_name***, where ***X*** is a digit from 0 to 9.

rc2
: This directory contains the start script for each product. The start script is copied to the */etc/rc2.d* directory when the product is added using **netconfig**. The script is removed when the product is removed. The scripts in this directory must be named ***SXXproduct_name***, where ***X*** is a digit from 0 to 9. This script is necessary only if you want the network product to start at boot time.

bin
: This directory contains scripts that are used by **netconfig**.

tmp
: This is a working directory containing temporary files and scripts created during the addition or removal of a product or chain.

reconf
: Directory used by **netconfig**; contains reconfigure scripts for each product.

src
: Directory used by **netconfig**.

chains
: File used by **netconf**.

Options

-r *chain* Remove. **netconfig** skips the main menu and removes the specified chain. Unless the **-d** option is specified, users will still be required to answer any questions that are asked by the removal scripts. Chain names are specified with a single word comprised of the names of each element in the chain separated by "#" characters: top#middle..#bottom. A single complete, valid chain name must be passed to **netconfig** with the **-r** option.

-l Link. **netconfig** relinks the kernel and installs it without asking (suppresses the link kernel prompt) if changes are made that require re-linking the kernel.

-n No link. **netconfig** will not relink the kernel even if changes have been made that require that the kernel be relinked.

-d Default mode. **netconfig** uses suitable defaults for all prompts. Vendor-specific configuration scripts should supply defaults for all prompts. This essentially entails redirecting the standard input of each vendor's provided init or removal scripts to the default file provided.

-v Version. **netconfig** prints its version number and exits.

-s Status. **netconfig** prints a list of the currently installed chains and exits. This is intended for use in shell scripts that are trying to remove all chains associated with their product.

-t Terse mode. Prints only the terse chain names. This is useful if more than four chains are installed to ensure that all of the chains are displayed without some of the chains scrolling off the top of the screen. (This is used in conjunction with other arguments and is not equivalent to the -s flag.)

-a *chain* Add. Add a chain specified by the user. Designed as part of the command line engine, this option will perform an add (call the add scripts) without any user intervention. If the chain passed in is invalid, then **netconfig** will return an error and output a single line error message suitable for display to the user.

-C -[a | r] *chain*
Check. Check a chain specified by the user. Designed as part of the command line engine, this option operates in conjunction with the -a and -r options and checks if it is valid to perform the intended add or remove operation. The Add and Remove command line options do not enforce the rules checked by the -C option so -C must be called if a user interface is used. If check detects an operation that should not be attempted, check will return an error and output a single line error message suitable for display to the user.

-c *chain element*
Reconfigure an element in a chain. Requires a chain name and element as arguments.

-m *chain* | **-m " "**
Menu. Provide a menu of the allowable next level down chain elements for a partially built chain. If the chain specified is a null word (that is, the argument passed is ""), then the list of top level products is provided.

-e *chain* Element. Output the list of elements in the given chain that support reconfiguring.

-L *chain* List. Given a chain, output a verbose listing of the chain. A list of the chain's elements and their descriptions is output.

See also

configure(ADM), **idtune**(ADM), **link_unix**(ADM), **mtune**(F)

"Tuning System Performance" in the *System Administrator's Guide*

netutil

administer the Micnet network

Syntax

/etc/netutil [*option*] [**-x**] [**-e**]

Description

The **netutil** command allows the user to create and maintain a network of UNIX machines. A Micnet network is a link through serial lines of two or more systems. It is used to send mail between systems with the **mail**(C) command, transfer files between systems with the **rcp**(C) command, and execute commands from a remote system with the **remote**(C) command.

The **netutil** command is used to create and distribute the data files needed to implement the network. It is also used to start and stop the network. The *option* argument may be any one of **install, save, restore, start, stop,** or the numbers 1 through 5 respectively.

The arguments cause **netutil** perform the following actions:

-x This option causes **netutil** to log transmissions.

-e Causes **netutil** to log errors. (For details of the log files, see the *System Administrator's Guide*).

the **-x** and **-e** options work only when they are used in conjunction with **start, stop** or their decimal equivalents (4 and 5).

install Interactively creates the data files needed to run the network.

save Saves these data files on floppy or hard disks, allowing them to be distributed to the other systems in the network. If you save the Micnet files to the hard disk, you can then use **uucp**(C) to transfer the files to the other machines. This option specifies the name of the backup device and prompts for whether this is the desired device to use. The user can specify an alternate device, including a file on the hard disk. The name of the default backup device is located in the file */etc/default/micnet*. This can be changed depending on system configuration.

restore This option copies the data files from floppy disk back to a system.

start Start the network.

stop Stop the network.

1 .. 5 If **netutil** is invoked without an ***option***, it displays a menu of options. Once an option is selected, it prompts for additional information if necessary. The decimal digit options 1 to 5 correspond to the menu entries presented by **netutil** if it is invoked without any options. They are synonymous with the text options listed above.

A network must be installed before it can be started. Installation consists of creating appropriate configuration files with the **install** option. This option requires the name of each machine in the network, the serial lines to be used to connect the machines, the speed of transmission for each line, and the names of the users on each machine. Once created, the files must be distributed to each computer in the network with the **save** and **restore** options. The network is started by using the **start** option on each machine in the network. Once started, mail and remote commands can be passed along the network. A record of the transmissions between computers in a network can be kept in the network log files. Installation of the network is described in the *System Administrator's Guide.*

Files

/bin/netutil
/etc/default/micnet

See also

mail(C), **micnet**(FP), **remote**(C), **rcp**(C), **systemid**(F), **top**(F)

Value added

netutil is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

nictable

process NIC database into channel/domain tables

Syntax

/usr/mmdf/table/nictable -[CDT] [-d *domain*] [-s *service*] [-t *transport*]

Description

nictable is the tool responsible for taking the *hosts.txt* table supplied by the SRI Network Information Center and creating domain and channel tables.

Options to **nictable** are as follows:

-C The **-C** option causes the program to generate a channel table on the standard output.

-D The **-D** option causes **nictable** to create a domain table. It should be combined with the **-d** option (below), which identifies the domain table to be built.

-T The **-T** option creates a "top" or "rootdomain" table. No trailing domain spec is removed from the LHS entry.

-d *domain* The **-d *domain*** option specifies that only hosts in ***domain*** should be output. An exception to this is when **-d** is combined with **-T**. In this case, all entries will be output *except* for those in the domain specified with **-T**. The intention is that you grab all of one domain with **-D**, and then grab everybody else with **-T**.

-s *service* The **-s *service*** option specifies that only hosts that are listed as supporting ***service*** should be output.

-t *transport* The **-t *transport*** option is like **-s** except that it states that only hosts supporting the transport protocol specified should be considered.

Typical usage involves two or three invocations:

nictable -C < /etc/hosts.txt > smtpchannel

nictable -D -d ARPA < /etc/hosts.txt > arpadomain

(and optionally)

nictable -T -d ARPA < /etc/hosts.txt > rootdomain

Value added

nictable is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

nlsadmin

network listener service administration

Syntax

nlsadmin -x

nlsadmin [***options*** **]** ***net_spec***

Description

nlsadmin administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process will work with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a general listener service, serving processes on remote machines, and a terminal login service, for terminals connected directly to a network. The terminal login service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is only available with some networks (for example, the AT&T STARLAN network).

nlsadmin can establish a listener process for a given network, configure the specific attributes of that listener, and start and kill the listener process for that network. **nlsadmin** can also report on the listener processes on a machine, either individually (per network) or collectively.

The following list shows how to use **nlsadmin**. In this list, ***net_spec*** represents a particular listener process. Specifically, ***net_spec*** is the relative pathname of the entry under */dev* for a given network (that is, a transport provider). Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted. The following combination of ***options*** can be used.

no options	Gives a brief usage message
-x	Reports the status of all of the listener processes installed on this machine
net_spec	Prints the status of the listener process for ***net_spec***
-q ***net_spec***	Queries the status of the listener process for the specified network, and reflects the result of that query in its exit code.

If a listener process is active, **nlsadmin** will exit with a status of 0; if no process is active, the exit code will be 1; the exit code will be greater than 1 in case of error.

-v ***net_spec***
Prints a verbose report on the servers associated with ***net_spec***, giving the service code, status, command, and comment for each.

-z ***service_code net_spec***
Prints a report on the server associated with ***net_spec*** that has service code ***service_code***, giving the same information as in the **-v** option.

-q -z ***service_code net_spec***
Queries the status of the service with service code ***service_code*** on network ***net_spec***, and will exit with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

-l addr ***net_spec***
Changes or sets the address on which the listener listens (the general listener service). This is the address generally used by remote processes to access the servers available through this listener (see the **-a** option, below). ***addr*** is the transport address on which to listen and is interpreted using a syntax that allows for a variety of address formats. By default, ***addr*** is interpreted as the symbolic ASCII representation of the transport address. An ***addr*** preceded by a "\x" will let you enter an address in hexadecimal notation. Note that ***addr*** must appear as a single word to the shell and must be quoted if it contains any blanks.

If ***addr*** is just a dash (-), **nlsadmin** will report the address currently configured, instead of changing it.

A change of address will not take effect until the next time the listener for that network is started.

-t ***addr net_spec***
Changes or sets the address on which the listener listens for requests for terminal service, but is otherwise similar to the **-l** option above. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code 1 (see the **-a** option, below).

-i ***net_spec***
Initializes or changes a listener process for the network specified by ***net_spec***; that is, it will create and initialize the files required by the listener. Note that the listener should only be initialized once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addressing or services.

[-m] -a ***service_code*** **[-p** ***modules*****] [-w** ***id*****] -c** ***cmd*** **-y** ***comment net_spec***
Adds a new service to the list of services available through the indicated listener. ***service_code*** is the code for the service, ***cmd*** is the command to be invoked in response to that service code, comprised of the full pathname of the server and its arguments, and ***comment*** is a brief (free-form) description

of the service for use in various reports. Note that ***cmd*** must appear as a single word to the shell, so if arguments are required, the ***cmd*** and its arguments must be surrounded by quotes. Similarly, the ***comment*** must also appear as a single word to the shell. When a service is added, it is initially enabled (see the **-e** and **-d** options below).

If the **-m** option is specified, the entry will be marked as an administrative entry. Service codes 1 through 100 are reserved for administrative entries, which are those that require special handling internally. In particular, code 1 is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

The **-m** option used with the **-a** option indicates that special handling internally is required for those servers added with the **-m** set. This internal handling is in the form of code embedded on the listener process.

If the **-p** option is specified, then ***modules*** will be interpreted as a list of STREAMS modules for the listener to push before starting the service being added. The modules are pushed in the order they are specified. ***modules*** should be a comma-separated list of modules, with no white space included.

If the **-w** option is specified, then ***id*** is interpreted as the user name from */etc/passwd* that the listener should look up. From the user name, the listener should obtain the user ID, the group ID, and the home directory for use by the server. If **-w** is not specified, the default is to use the user ID *listen*.

A service must explicitly be added to the listener for each network on which that service is to be available. This operation will normally be performed only when the service is installed on a machine, or when populating the list of services for a new network.

-r ***service_code net_spec***

Removes the entry for the ***service_code*** from that listener's list of services. This will normally be performed only in conjunction with the de-installation of a service from a machine.

-e ***service_code net_spec***
-d ***service_code net_spec***

Enables or disables (respectively) the service indicated by ***service_code*** for the specified network. The service must have previously been added to the listener for that network (see the **-a** option above). Disabling a service will cause subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running will continue unaffected.

-s ***net_spec***
-k ***net_spec***

Starts and kills (respectively) the listener process for the indicated network. These operations will normally be performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must be defined

for the general listener service (see the **-i** and **-l** options, above). When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected.

The listener runs as user ID *root*, with group ID *sys*. A special ID, user ID *listen* and group ID *adm*, should be entered in the */etc/passwd* file as a default ID for servers. The listener always uses as its home directory */usr/net/nls*, which is concatenated with ***net_spec*** to determine the location of the listener configuration information for each network. The home directory specified in the */etc/passwd* entry for *listener* will be used by servers that run as ID *listen*.

nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super user.

Diagnostics

If the command is not run under the proper ID, an error message will be sent to standard error and the command will terminate.

File

*/usr/net/nls/**net_spec***

See also

Network Programmer's Guide

pipe

list or define pipe filesystem

Syntax

/etc/pipe [-d | -l | -s *path_name*]

Description

This utility is used to manipulate the filesystem on which anonymous pipes reside. A pipe is a first-in first-out (FIFO) data structure used to transfer the output of one program to the input of another. Pipe data structures are normally stored in the buffer cache and accessed via inodes, like regular files. The pipe data structures therefore notionally reside on a filesystem which must be mounted and both readable and writeable; otherwise pipes cannot be used. Pipes borrow inodes from this filesystem, but will only write data to the physical medium under exceptional circumstances.

The following options are available:

-s *path_name* Designates the pipe filesystem to be that on which ***path_name*** resides.

-d Disables pipes (there is no pipe filesystem). Currently existing pipes are not affected.

-l Lists the name of the pipe filesystem (*/dev/xxx*). If there is no pipe filesystem, nothing is output. (This option operates by listing the names of all the block special devices in the */dev* hierarchy which have the same device numbers (both major and minor) as the pipe filesystem.)

Notes

Only one pipe filesystem may be designated at a time. Changes to the pipe filesystem take effect immediately.

Programs attempting to create, write to, or read from a pipe will experience an error condition if no writable or readable pipe filesystem is available.

Named pipes (FIFOs) reside on the filesystem containing their name, and do not use the pipe filesystem.

pkgadd

transfer software package to the system

Syntax

/usr/bin/pkgadd [**-d** *device*] [**-r** *response*] [**-n**] [**-a** *admin*] [*pkginst1* [*pkginst2* [...]]]

/usr/bin/pkgadd -s *spool* [**-d** *device*] [*pkginst1* [*pkginst2* [...]]]

Description

pkgadd transfers the contents of a software package from the distribution medium or directory to install it onto the system. Used without the **-d** option, **pkgadd** looks in the default spool directory for the package (*/usr/spool/pkg*). Used with the **-s** option, it reads the package to a spool directory instead of installing it.

-d *device* Installs or copies a package from *device*. *device* can be a full pathname to a directory or the identifiers for tape, floppy disk or removable disk (for example, */tmp*, */dev/rct0*, */dev/fd096ds15* or */dev/dsk/f03h*). It can also be the device alias (for example, **ctape1** for the cartridge tape drive).

-r *response* Identifies a file or directory, *response*, which contains output from a previous **pkgask** session. This file supplies the interaction responses that would be requested by the package in interactive mode. *response* must be a full pathname.

-n Installation occurs in non-interactive mode. The default mode is interactive.

-a *admin* Defines an installation administration file, *admin*, to be used in place of the default administration file. The token **none** overides the use of any *admin* file, and thus forces interaction with the user. Unless a full path name is given, **pkgadd** looks in the */usr/adm/install/admin* directory for the file.

pkginst Specifies the package instance or list of instances to be installed. The token **all** may be used to refer to all packages available on the source medium. The format *pkginst.** can be used to indicate all instances of a package. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the "*" character.

-s *spool* Reads the package into the directory ***spool*** instead of installing it.

When executed without options, **pkgadd** uses */usr/spool/pkg* (the default spool directory).

Notes

When transferring a package to a spool directory, the **-r**, **-n**, and **-a** options cannot be used.

The **-r** option can be used to indicate a directory name as well as a filename. The directory can contain numerous *response* files, each sharing the name of the package with which it should be associated. This would be used, for example, when adding multiple interactive packages with one invocation of **pkgadd**. Each package would need a *response* file. If you create response files with the same name as the package (that is, *package1* and *package2*), then name the directory in which these files reside after the **-r**.

The **-n** option will cause the installation to halt if any interaction is needed to complete it.

pkgask

store answers to a request script

Syntax

/usr/bin/pkgask [**-d** *device*] **-r** *response* [*pkginst* [*pkginst* [...]]]

Description

pkgask allows the administrator to store answers to an interactive package's request script. Invoking this command generates a *response* file that is then used as input at installation time. The use of this *response* file prevents any interaction from occurring during installation since the file already contains all of the information the package needs.

-d Runs the request script for a package on *device*. *device* can be a full pathname to a directory or the identifiers for tape, floppy disk or removable disk (for example, */tmp*, */dev/fd096ds15* or */dev/dsk/f03h*). The default device is the installation spool directory.

-r Identifies a file or directory, which should be created to contain the responses to interaction with the package. The name must be a full pathname. The file, or directory of files, can later be used as input to the **pkgadd** command.

pkginst Specifies the package instance or list of instances for which request scripts will be created. The token **all** may be used to refer to all packages available on the source medium. The format *pkginst.** can be used to indicate all instances of a package. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.

Notes

The **-r** option can be used to indicate a directory name as well as a filename. The directory name is used to create numerous *response* files, each sharing the name of the package with which it should be associated. This would be used, for example, when adding multiple interactive packages with one invocation of **pkgadd**. Each package would need a *response* file. To create multiple response files with the same name as the package instance, name the directory in which the files should be created and supply multiple instance names with the **pkgask** command. When installing the packages, you will be able to identify this directory to the **pkgadd** command.

See also

installf(ADM), **pkgadd**(ADM), **pkgchk**(ADM), **pkginfo**(ADM), **pkgmk**(ADM), **pkgparam**(ADM), **pkgproto**(ADM), **pkgrm**(ADM), **pkgtrans**(ADM), **removef**(ADM)

pkgchk

check accuracy of installation

Syntax

/usr/bin/pkgchk [**-l** | **-acfqv**] [**-nx**] [**-p** *path1* [*,path2...*] [**-i** *file*] [*pkginst...*]]

/usr/bin/pkgchk -d *device* [**-l** | **v**] [**-p** *path1* [*,path2...*] [**-i** *file*] [*pkginst...*]]

/usr/bin/pkgchk -m *pkgmap* [**-e** *envfile*] [**-l** | **-acfqv**] [**-nx**] [**-i** *file*]
[**-p** *path1* [*,path2...*]]

Description

pkgchk checks the accuracy of installed files or, by use of the **-l** option, displays information about package files. The command checks the integrity of directory structures and files. Discrepancies are reported on standard error along with a detailed explanation of the problem.

The first synopsis defined above is used to list or check the contents and/or attributes of objects that are currently installed on the system. Package names may be listed on the command line, or by default the entire contents of a machine will be checked.

The second synopsis is used to list or check the contents of a package which has been spooled on the specified device, but not installed. Note that attributes cannot be checked for spooled packages.

The third synopsis is used to list or check the contents and/or attributes of objects which are described in the indicated *pkgmap*.

The option definitions are:

-l Lists information on the selected files that make up a package. It is not compatible with the **a**, **c**, **f**, **q**, and **v** options.

-a Audits the file attributes only: does not check file contents. Default is to check both.

-c Audits the file contents only: does not check file attributes. Default is to check both.

-f Corrects file attributes if possible. When **pkgchk** is invoked with this option it creates directories, named pipes, links and special devices if they do not already exist.

-q Quiet mode. Does not give messages about missing files.

-v Verbose mode. Files are listed as processed.

-n Does not check volatile or editable files. This should be used for most post-installation checking.

-x Searches exclusive directories only, looking for files which exist that are not in the installation software database or the indicated ***pkgmap*** file. If used with the **-f** option, hidden files are removed; no other checking is done.

-p Only checks the accuracy of the pathname or pathnames listed. ***pathname*** can be one or more pathnames separated by commas (or by white space, if the list is quoted).

-i Reads a list of pathnames from ***file*** and compares this list against the installation software database or the indicated ***pkgmap*** file. Pathnames which are not contained in ***file*** are not checked.

-d Specifies the device on which a spooled package resides. ***device*** can be a directory pathname or the identifiers for tape, floppy disk or removable disk (for example, */tmp* or */dev/fd096ds15*).

-m Requests that the package be checked against the pkgmap file ***pkgmap***.

-e Requests that the **pkginfo** file named as ***envfile*** be used to resolve parameters noted in the specified pkgmap file.

pkginst Specifies the package instance or instances to be checked. The format ***pkginst.**** can be used to check all instances of a package. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character. The default is to display all information about all installed packages.

Notes

To remove hidden files only, use the **-f** and **-x** options together. To remove hidden files and check attributes and contents of files, use the **-f**, **-x**, **-c**, and **-a** options together.

See also

pkgadd(ADM), **pkgask**(ADM), **pkginfo**(ADM), **pkgrm**(ADM), **pkgtrans**(ADM)

pkginfo

display software package information

Synopsis

pkginfo [**-q** | **x** | **l**] [**-p** | **i**] [**-a** *arch*] [**-v** *version*] [**-c** *category1*, [*category2* [, ...]]] [*pkginst* [,*pkginst* [, ...]]]

pkginfo [**-d** *device* [**-q** | **x** | **l**] [**-a** *arch*] [**-v** *version*] [**-c** *category1*, [*category2* [, ...]]] [*pkginst* [,*pkginst*[, ...]]]

Description

The **pkginfo** command displays information about software packages which are installed on the system (with the first synopsis) or which reside on a particular device or directory (with the second synopsis). No information will be displayed for packages installed using **custom**(ADM). Only the package name and abbreviation will be included in the display for packages installed using **installpkg**(ADM).

The options for this command are:

-q Does not list any information, but can be used from a program to check (that is, query) whether or not a package has been installed.

-x Designates an extracted listing of package information. It contains the package abbreviation, package name, package architecture (if available) and package version (if available).

-l Designates long format, which includes all available information about the designated package(s).

-p Designates that information should be presented only for partially installed packages.

-i Designates that information should be presented only for fully installed packages.

-a Specifies the architecture of the package as ***arch***.

-v Specifies the version of the package as ***version***. All compatible versions can be requested by preceding the version name with a tilde (~). The list produced by **-v** will include packages installed using **installpkg**(ADM). (with which no version numbers are associated). Multiple white spaces are replaced with a single space during version comparison.

-c Selects packages to be displayed based on the category ***category***. (Categories are defined in the "category" field of the *pkginfo* file.) If more than one category is supplied, the package must only match one of the list of categories. The match is not case specific.

pkginst Designates a package by its instance. An instance can be the package abbreviation or a specific instance (for example, *inst.1* or *inst.beta*). All instances of package can be requested by *inst.**. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.

-d Defines a device, ***device***, on which the software resides. ***device*** can be a full pathname to a directory or the identifiers for tape, floppy disk, removable disk, and so on. The special token "spool" may be used to indicate the default installation spool directory.

Notes

Without options, **pkginfo** lists the primary category, package instance, and name of all completely installed and partially installed packages. One line per package selected is produced.

The **-p** and **-i** options are meaningless if used in conjunction with the **-d** option.

The options **-q**, **-x**, and **-l** are mutually exclusive.

pkginfo cannot tell if a package installed using **installpkg**(ADM) is only partially installed. It is assumed that all such packages are fully installed.

See also

pkgadd(ADM), **pkgask**(ADM), **pkgchk**(ADM), **pkgrm**(ADM), **pkgtrans**(ADM)

pkgmk

produce an installable package

Syntax

pkgmk [**-o**] [**-d** *device*] [**-r** *rootpath*] [**-b** *basedir*] [**-l** *limit*] [**-a** *arch*] [**-v** *version*] [**-p** *pstamp*] [**-f** *prototype*] [*variable=value ...*] [*pkginst*]

Description

pkgmk produces an installable package to be used as input to the **pkgadd** command. The package contents will be in directory structure format.

The command uses the package *prototype* file as input and creates a *pkgmap* file. The contents for each entry in the *prototype* file are copied to the appropriate output location. Information concerning the contents (checksum, file size, modification date) is computed and stored in the *pkgmap* file, along with attribute information specified in the *prototype* file.

-o Overwrites the same instance: package instance will be overwritten if it already exists.

-d *device* Creates the package on *device*. *device* can be a full pathname to a directory or the identifiers for a floppy disk or removable disk (for example, */dev/fd096ds15*). The default device is the installation spool directory.

-r *rootpath* Ignores destination paths in the *prototype* file. Instead, uses the indicated *rootpath* with the source pathname appended to locate objects on the source machine.

-b *basedir* Prepends the indicated *basedir* to locate relocatable objects on the source machine.

-l *limit* Specifies the maximum size in 512 byte blocks of the output device as *limit*. By default, if the output file is a directory or a mountable device, **pkgmk** will employ the **df** command to dynamically calculate the amount of available space on the output device. Useful in conjunction with **pkgtrans** to create package with datastream format.

-a *arch* Overrides architecture information provided in the *pkginfo* file with *arch*.

-v *version* Overrides version information provided in the *pkginfo* file with *version*.

-p *pstamp* Overrides the production stamp definition in the *pkginfo* file with *pstamp*.

-f ***prototype*** Uses the file ***prototype*** as input to the command. The default name for this file is either *Prototype* or *prototype*.

variable=value Places the indicated variable in the packaging environment. (See **prototype**(F) for definitions of packaging variables.)

pkginst Specifies the package by its instance. **pkgmk** will automatically create a new instance if the version and/or architecture is different. A user should specify only a package abbreviation; a particular instance should not be specified unless the user is overwriting it.

Notes

Architecture information is provided on the command line with the **-a** option or in the *prototype* file. If no architecture information is supplied at all, the output of **uname -m** will be used.

Version information is provided on the command line with the **-v** option or in the *prototype* file. If no version information is supplied, a default based on the current date will be provided.

Command line definitions for both architecture and version overide the *prototype* definitions.

See also

pkgparam(ADM), **pkgproto**(ADM), **pkgtrans**(ADM), **pkgproto**(F)

pkgparam

display package parameter values

Syntax

pkgparam [**-v**] [**-d** *device*] *pkginst* [*param* [...]]

pkgparam -f *file* [**-v**] [*param* [...]]

Description

pkgparam displays the value associated with the parameter or parameters requested on the command line. The values are located in either the **pkginfo** file for *pkginst* or from the specific file named with the **-f** option.

One parameter value is shown per line. Only the value of a parameter is given unless the **-v** option is used. With this option, the output of the command is in this format:

parameter1='value1'
parameter2='value2'
parameter3='value3'

If no parameters are specified on the command line, values for all parameters associated with the package are shown.

Options and arguments for this command are:

-v Specifies verbose mode. Displays name of parameter and its value.

-d Specifies the *device* on which a *pkginst* is stored. It can be a full pathname to a directory or the identifiers for tape, floppy disk or removable disk (for example, */tmp*, */dev/fd096ds15*, or */dev/dsk/f03h*). The default device is the installation spool directory. If no instance name is given, parameter information for all packages residing in *device* is shown.

-f Requests that the command read *file* for parameter values.

pkginst Defines a specific package instance for which parameter values should be displayed. The format *pkginst.** can be used to indicate all instances of a package. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.

param Defines a specific parameter whose value should be displayed.

Diagnostics

If parameter information is not available for the indicated package, the command exits with a non-zero status.

Notes

The **-f** synopsis allows you to specify the file from which parameter values should be extracted. This file should be in the same format as a **pkginfo** file. As an example, such a file might be created during package development and used while testing software during this stage.

See also

pkgmk(ADM), **pkgproto**(ADM), **pgktrans**(ADM)

pkgproto

generate a prototype(F) file

Syntax

pkgproto [-i] [-c *class*] [*path1* [= *path2*] ...]

Description

pkgproto scans the indicated paths and generates a *prototype* file that may be used as input to the **pkgmk** command.

-i Ignores symbolic links and records the paths as **ftype=f** (a file) versus **ftype=s** (symbolic link)

-c Maps the class of all paths to ***class***.

path1 Path of directory where objects are located.

path2 Path that should be substituted on output for ***path1***.

If no paths are specified on the command line, standard input is assumed to be a list of paths. If the path listed on the command line is a directory, the contents of the directory are searched. However, if input is read from *stdin*, a directory specified as a path will not be searched.

Notes

By default, **pkgproto** creates symbolic link entries for any symbolic link encountered (**ftype=s**). When you use the **-i** option, **pkgproto** creates a file entry for symbolic links (**ftype=f**). The *prototype* file would have to be edited to assign such file types as **v** (volatile), **e** (editable), or **x** (exclusive directory). **pkgproto** detects linked files. If multiple files are linked together, the first path encountered is considered the source of the link.

Examples

The following two examples show uses of **pkgproto** and a partial listing of the output produced.

Example 1:

```
$ pkgproto /bin=bin /usr/bin=usrbin /etc=etc
f none bin/sed=/bin/sed 0711 bin bin
f none bin/sh=/bin/sh 1711 bin bin
f none bin/sort=/bin/sort 0711 bin bin
f none usrbin/sdb=/usr/bin/sdb 0711 bin bin
f none usrbin/shl=/usr/bin/shl 4555 root sys
f none etc/rc=/etc/rc 0744 root sys
```

Example 2:

```
$ find / -type d -print | pkgproto
d none / 755 root bin
d none /usr/bin 755 bin bin
d none /usr 755 root auth
d none /etc 755 bin auth
d none /tmp 1777 sys sys
```

See also

pkgmk(ADM), **pkgparam**(ADM), **pkgtrans**(ADM)

pkgrm

remove a package from the system

Syntax

pkgrm [**-n**] [**-a** *admin*] [*pkginst1* [*pkginst2* [...]]]

pkgrm -s *spool* [*pkginst*]

Description

pkgrm will remove a previously installed or partially installed package from the system. A check is made to determine if any other packages depend on the one being removed. The action taken if a dependency exists is defined in the *admin* file.

The default state for the command is interactive mode, meaning that prompt messages are given during processing to allow the administrator to confirm the actions being taken. Non-interactive mode can be requested with the **-n** option.

The **-s** option can be used to specify the directory from which spooled packages should be removed.

The options and arguments for this command are:

-n Non-interactive mode. If there is a need for interaction, the command will exit. Use of this option requires that at least one package instance be named upon invocation of the command.

-a ***admin*** Defines an installation administration file, ***admin***, to be used in place of the default *admin* file.

-s ***spool*** Removes the specified package(s) from the directory ***spool***.

pkginst Specifies the package to be removed. The format ***pkginst.**** can be used to remove all instances of a package. When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.

See also

installf(ADM), **pkgadd**(ADM), **pkgask**(ADM), **pkgchk**(ADM), **pkginfo**(ADM), **pkgmk**(ADM), **pkgparam**(ADM), **pkgproto**(ADM), **pkgtrans**(ADM), **removef**(ADM)

pkgtrans

translate package format

Syntax

pkgtrans [**-ions**] *device1 device2* [*pkginst1* [*pkginst2* [...]]]

Description

pkgtrans translates an installable package from one format to another. It translates:

- a filesystem format to a datastream
- a datastream to a filesystem format
- a filesystem format to another filesystem format

The options and arguments for this command are:

-i Copies only the ***pkginfo*** and ***pkgmap*** files.

-o Overwrites the same instance on the destination device: package instance will be overwritten if it already exists.

-n Creates a new instance if any instance of this package already exists.

-s Indicates that the package should be written to ***device2*** as a datastream rather than as a filesystem. The default behavior is to write a filesystem format on devices that support both formats.

device1 Indicates the source device. The package or packages on this device will be translated and placed on ***device2***.

device2 Indicates the destination device. Translated packages will be placed on this device.

pkginst Specifies which package instance or instances on ***device1*** should be translated. The token **all** may be used to indicate all packages. ***pkginst.**** can be used to indicate all instances of a package. (When using this format, enclose the command line in single quotes to prevent the shell from interpreting the * character.) If no packages are defined, a prompt shows all packages on the device and asks which to translate.

Notes

Device specifications can be either the special node name (*/dev/fd096ds15*) or the device alias (*fd096ds15*). The device *spool* indicates the default spool directory. Source and destination devices may not be the same.

By default, **pkgtrans** will not transfer any instance of a package if any instance of that package already exists on the destination device. Use of the **-n** option will create a new instance if an instance of this package already exists. Use of the **-o** option will overwrite the same instance if it already exists. Neither of these options are useful if the destination device is a datastream.

If you're transferring a package in datastream format to floppies and the package spans multiple floppies, use the filesystem format. (The datastream format is not supported across multiple floppies.)

pkgtrans depends on the integrity of the */etc/default/device.tab* file to determine whether a device can support a datastream and/or filesystem formats. Problems in transferring a device in a particular format could mean corruption of */etc/default/device.tab*.

Example

The following example translates all packages on the floppy drive */dev/diskette* and places the translations on */tmp*.

pkgtrans /dev/diskette /tmp all

The next example translates packages *pkg1* and *pkg2* on */tmp* and places their translations (that is, a datastream) on the cartridge tape output device.

pkgtrans /tmp ctape1 pkg1 pkg2

The next example translates *pkg1* and *pkg2* on *tmp* and places them on the diskette in a datastream format.

pkgtrans -s /tmp /dev/fd096ds15 pkg1 pkg2

File

/etc/default/device.tab

See also

installf(ADM), **pkgadd**(ADM), **pkgask**(ADM), **pkginfo**(ADM), **pkgmk**(ADM), **pkgparam**(ADM), **pkgproto**(ADM), **pkgrm**(ADM), **removef**(ADM)

profiler: prfld, prfstat, prfdc, prfsnap, prfpr

system profiler

Syntax

/etc/prfld [***system_namelist***]

/etc/prfstat on

/etc/prfstat off

/etc/prfdc ***file*** [***period***] [***off_hour***]

/etc/prfsnap ***file***

/etc/prfpr ***file*** [***cutoff***] [***system_namelist***]

Description

prfld - initializes profiling

prfstat - enables/disables sampling

prfdc - periodically collects data

prfsnap - collects data at time of invocation

prfpr - formats profiler data

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

The **prfld** program is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from ***system_namelist***.

The **prfstat** program is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. **prfstat** will also reveal the number of text addresses being measured.

The **prfdc** and **prfsnap** programs 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** will store the counters into ***file*** every ***period*** minutes and will turn 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***.

The **prfpr** program formats the data collected by **prfdc** or **prfsnap**. Each text address is converted to the nearest text symbol (as found in ***system_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 system namelist file

proto

prototype job file for at, cron and batch

Syntax

/usr/lib/cron/.proto

/usr/lib/cron/.proto.queue

Description

When a job is submitted to **at**(C) or **batch**, the job is constructed as a shell script. First, a prologue is constructed, consisting of:

- A header whether the job is an **at** job or a **batch** job (actually, **at** jobs submitted to all queues other than queue *a*, not just to the batch queue *b*, are listed as **batch** jobs); the header will be

  ```
  : at job
  ```

  for an **at** job, and

  ```
  : batch job
  ```

  for a **batch** job.
- A set of Bourne shell commands to make the environment (see **environ**(M)) for the **at** job the same as the current environment;
- A command to run the user's shell (as specified by the **SHELL** environment variable) with the rest of the job file as input.

at then reads a "prototype file" and constructs the rest of the job file from it.

Text from the prototype file is copied to the job file, except for special "variables" that are replaced by other text:

$d is replaced by the current working directory

$l is replaced by the current file size limit (see **ulimit**(S))

$m is replaced by the current umask (see **umask**(S))

$t is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon

$< is replaced by text read by **at** from the standard input (that is, the commands provided to **at** to be run in the job)

If the job is submitted in queue ***queue***, **at** uses the file */usr/lib/cron/.proto.**queue*** as the prototype file if it exists; otherwise it will use the file */usr/lib/cron/.proto*.

Examples

The standard *.proto* file supplied is:

```
#ident  "@(#)adm:.proto 1.2"
cd $d
ulimit $l
umask $m
$<
```

which causes commands to change the current directory in the job to the current directory at the time **at** was run, to change the file size limit in the job to the file size limit at the time **at** was run, and to change the umask in the job to the umask at the time **at** was run, to be inserted before the commands in the job.

Files

/usr/lib/cron/.proto
*/usr/lib/cron/.proto.**queue***

See also

at(C), **atcronsh**(ADM), **sysadmsh**(ADM)

pwck

check password file

Syntax

pwck [*file*]

Description

pwck scans the password file and checks for 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 default password file is */etc/passwd*.

File

/etc/passwd

See also

group(F), **grpck**(ADM), **passwd**(FP)

Value added

pwck is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

pwconv, pwunconv

install and update the shadow password file
remove the shadow password file

Syntax

pwconv

pwunconv

Description

pwunconv - remove the shadow password file

The **pwconv** command creates and updates */etc/shadow* with information from the Protected Password database and */etc/passwd*. The command populates */etc/shadow* with the user's login name, password, and password aging information.

The following is the format of an entry in */etc/passwd*:

username:passwd,aging:uid:gid:comment:homedir:shell

pwconv creates an entry in */etc/shadow* for every user in */etc/passwd*. The password and password aging information are read from the Protected Password database and */etc/passwd* and stored along with the username in */etc/shadow*. The password and password aging information in */etc/passwd* are replaced with the letter **x**. Any existing entries in */etc/shadow* without a corresponding entry in */etc/passwd* are removed.

The **pwunconv** command is the reverse of **pwconv**. The password and password aging information held in */etc/shadow* are written back to */etc/passwd* and to the Protected Password database. */etc/shadow* is then removed.

Files

/etc/passwd	Password file
/etc/shadow	Shadow Password file
/etc/passwd-t	Temporary password file.
/etc/shadow-t	Temporary shadow password file.
/etc/passwd-o	Previous password file.
/etc/shadow-o	Previous shadow password file.
*/tcb/files/auth/?/**	Protected Password database.

See also

authck(ADM), **passwd**(C)

Diagnostics

The **pwconv** and **pwunconv** commands exit with one of the following values:

0 Success.

1 Not allowed to run this command.

2 Invalid command syntax.

3 Unexpected failure, conversion not done.

4 Unexpected failure, password file(s) missing.

5 Password file(s) busy, try again later.

6 **pwunconv** failed as */etc/shadow* does not exist.

7 A password file entry created would be too long.

An exit status greater than zero is accompanied by an error message describing the problem. If the message is "error reading entry for username", run **authck -p** to fix the problem and try again. If the message is "unable to restore previous password file" the password file has been renamed to */etc/passwd-o* and should be renamed back to */etc/passwd*. A similar message is output for the shadow password file, */etc/shadow-o* should be renamed */etc/shadow*.

Authorization

pwconv and **pwunconv** require the invoking user to be the super user or have the *auth* subsystem authorization, and have both the *chown* and *execsuid* kernel authorizations.

rc0

run commands performed to stop the operating system

Syntax

/etc/rc0

Description

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown".

One system state requires this procedure: state 0 (the system halt state). Whenever a change to this state occurs, the **/etc/rc0** procedure is run. The entry in */etc/inittab* might read:

```
s0:0:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
```

Some of the actions performed by **/etc/rc0** are carried out by files in the directory */etc/shutdown.d* and files beginning with K in */etc/rc0.d*. These files are executed in ASCII order (see "Files" below for more information), terminating some system service. The combination of commands in **/etc/rc0** and files in */etc/shutdown.d* and */etc/rc0.d* determines how the system is shut down.

The recommended sequence for **/etc/rc0** is:

1. Stop system services and daemons.

   Various system services (such as the **lp** spooler) are gracefully terminated.

   When new services are added that should be terminated when the system is shut down, the appropriate files are installed in */etc/shutdown.d* and */etc/rc0.d*.

2. Terminate processes.

   SIGTERM signals are sent to all running processes by **killall**(ADM). Processes stop themselves cleanly if sent **SIGTERM**.

3. Kill processes.

   SIGKILL signals are sent to all remaining processes; no process can resist **SIGKILL**.

   At this point the only processes left are those associated with **/etc/rc0** and processes 0 and 1, which are special to the operating system.

4. Unmount all filesystems.

   Only the *root* filesystem (/) remains mounted.

Notes

This file is intended for execution by **init**. It should not be executed by the user under any circumstances.

Files

The execution by **/bin/sh** of any files in */etc/shutdown.d* occurs in ASCII sort-sequence order. See **rc2**(ADM) for more information.

See also

killall(ADM), **rc2**(ADM), **shutdown**(ADM)

rc2

run commands performed for multiuser environment

Syntax

/etc/rc2

Description

This file is executed via an entry in */etc/inittab* and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multiuser" state.

The actions performed by **/etc/rc2** are found in files in several directories and are executed in a prescribed order to ensure proper initialization. **/etc/rc2** performs the following functions in the order in which they appear:

1. Runs the script */etc/conf/bin/idmkenv*. This script sets up the new kernel environment if a new kernel has been configured, calls **idmkinit** to rebuild the */etc/inittab* file, and links files to the */etc/idrc.d* and */etc/idsd.d* directories to be run by **/etc/rc2**.
2. Runs the system setup scripts in the directory */etc/rc2.d*. Some of the scripts in this directory are front-end scripts to run other scripts in the subdirectories of */etc/rc.d*.
3. Runs system setup scripts in the directory */etc/rc.d*. This directory exists for XENIX compatibility. It contains subdirectories named with the numerals 0 to 9. Each subdirectory contains scripts that perform certain system startup functions (for example, the directory */etc/rc.d/3* contains scripts that handle crash recovery). All of these scripts are run by the front-end scripts in */etc/rc2.d*. Any other individual scripts in the directory are run.
4. Runs the system setup scripts in the directory */etc/idrc.d*, which contains scripts from the driver packages linked from */etc/conf/rc.d*.
5. Runs the scripts in */etc/idsd.d*, which contains shutdown scripts linked from */etc/conf/sd.d*.
6. Runs the script */etc/rc*. This script exists for XENIX compatibility. It is an empty file, but you can add initialization commands to the file. These commands are run last during the initialization.

   The setup scripts are executed by **/bin/sh** in ASCII sort-sequence order (see "Files" for more information). When functions are added that need to be initialized when the system goes multiuser, an appropriate file should be added in */etc/rc2.d*.

Other functions can be added, as required, to support the addition of hardware and software features.

Examples

The following are prototypical files found in */etc/rc2.d*. These files are prefixed by an *S* and a number indicating the execution order of the files.

MOUNTFSYS

```
#    Set up and mount file systems
cd /
/etc/mountall
```

uucp

```
#    clean up uucp locks, status, and temporary files
rm -rf /usr/spool/locks/*
```

/etc/rc2 also sets certain environment variables, including the **TZ** variable, by reading */etc/TIMEZONE*, thus establishing the default environment for all commands that follow.

Files

Here are some hints about files in */etc/rc.d*:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, the first character of the filename is a sequence indicator that helps keep the proper order. Thus, files starting with the following characters would run accordingly:

[0-9]	very early
[A-Z]	early
[a-n]	later
[o-z]	last

Files in */etc/rc.d* that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them.

Files in */etc/rc2.d* must begin with an *S* or a *K* followed by a number and the rest of the filename. Upon entering run level 2, files beginning with *S* are executed with the **start** option; files beginning with *K* are executed with the **stop** option. Files beginning with other characters are ignored.

Notes

This file is intended for execution by **init**. It must never be executed directly by a user.

See also

shutdown(ADM), **init**(M), **rc0**(ADM)

"Starting and stopping the system" in the *System Administrator's Guide*

reduce

perform audit data analysis and reduction

Syntax

/tcb/bin/reduce [**-s** ***session***] [**-p** ***selection_file***]

Description

reduce performs selective audit data reduction on compacted audit output files that were written by the audit daemon. Each audit record from the compaction files is examined during reduction to see if it meets the selectivity criteria established by the audit administrator. If so, the record is formatted and sent to standard output.

Reduction is performed on all files written by the audit daemon during a specified boot ***session***. Each time the audit subsystem is enabled and disabled, a new session number is generated. This session number is used to stamp the filenames generated during the session so that they are easily recognizable. The audit daemon records each filename to which it writes compacted data in a log file. The log file is always written to the secure directory, */tcb/files/audit*. Each session log file is uniquely named with the prefix **CAFLOG.** followed by the session number. Thus, by specifying a session number for reduction, **reduce** is able to locate the log file and read it to determine certain setup parameters and the list of input files to be reduced.

Data is reduced based on a set of input selection criteria that governs the selection of records for printing. Records may be selected based on event types, time of event occurrence, user ID of record, group ID of record, or by specific object type. To selectively reduce, **auditsh**(ADM) is used to set up the audit selection file. This file is then specified to **reduce** upon invocation. Time interval selection allows for records to be selected only if they occurred within a certain time period. Event type selection allows records to be selected only if the specified event type is desired. Both user ID and group ID selection allow records that were generated by certain users or groups to be selected. Lastly, object selection applies to those record types referring to a specific file. Some records refer to multiple files and a single match for those record types will result in the record being selected. Time and event type selection always take precedence over user/group ID and object selection (for example, if a record has an event type that is not selected but the user ID is, the record will be discarded). If a record is selected based on time and event type and if any of user ID, group ID, or object matches a field in the record, the record is selected. If only time and event types are specified, all records of matching event types in the interval are selected. If only event type selection is requested, all matching events are selected from every record produced in that session. (For example, if the event mask enables selection for all events and no time interval is specified, all records will be listed.)

The format of the reduced data varies on the type of event being processed. Each record will include the process ID of the process being audited, the date and time of the event, the type of audit event, an indication of success or failure for the event, and if applicable, the object names that were accessed.

Items that are displayed for events include the following:

Process ID — The process ID of the process that generated the audit record.

User IDs — The login user ID, effective user ID, real user ID, effective group ID, and the real group ID are output for the process generating the audit record.

Date/Time — Each audit record is time stamped at generation time. The time value is formatted to produce a date/time string similar to that printed by **ctime**(S).

Event type — Each audit record is classified into a certain event depending on what type of system call was performed or what type of action was taken by a trusted application.

Action — Many event types are broad categories into which certain actions are classified. The reduction program makes use of other data in the record to provide further discrimination between process actions that fall into the category. For system calls, the actual system call audited is output. For applications, a more specific action identifier is provided.

Object(s) — Many events involve files or special devices which are classified as objects. The name of the objects affected by process actions are recorded for data reduction. Depending on the event and action type, some output records may include one or more object names.

Modes — For certain event types, the modes of a file or an IPC object may be modified. For these records, the old and new values of the owner, group, and the object mode are displayed.

Username — Some events are user-account oriented such as login and logoff as well as certain administrative functions. These output records include the username of the account that was responsible for the audited action.

Result — Each output record carries an indicator of whether the action was successful or not. Unsuccessful actions are sometimes more important that successful ones since they may indicate attempts to penetrate the system. For system calls that fail, the specific error number and error message is output. For applications, an error message describing the failure is output.

See also

audit(HW), **auditd**(ADM), **auditsh**(ADM)

"Using the audit subsystem" in the *System Administrator's Guide*

Diagnostics

Upon successful completion, the program exits with status 0.

Value added

reduce is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

relax

change system security defaults

Syntax

/etc/relax *level*

Description

relax changes selected system-wide security defaults to one of several predefined levels located in */tcb/lib/relax*. Each level is named and defined by the directory that contains three files:

default specifies fields to be updated in the file */etc/auth/system/default*.

etc_def specifies files and values to be updated or removed in the directory */etc/default*.

script is a shell script which is run to make other changes to such settings as kernel parameters using **configure**(ADM), default umask settings by edits to various files, and other changes.

Example

relax takes one argument, which should be the name of a directory in */tcb/lib/relax*, for example:

relax improved

This reconfigures the system security settings to the "improved" level of security.

See also

configure(ADM), **umask**(C)

System Administrator's Guide

Files

*/tcb/lib/relax/**
/etc/auth/system/files
*/etc/default/**
/etc/initscript
/etc/profile
/etc/cshrc

Note

If the script for the level you select makes changes to the kernel configuration, these changes will not come into effect until UNIX is rebooted.

Value added

relax is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

relogin

rename login entry to show current layer

Syntax

/usr/lib/layersys/relogin [-s] [*line*]

Description

The **relogin** command changes the terminal ***line*** field of a user's *utmp*(F) entry to the name of the windowing terminal layer attached to standard input. **write**(C) messages sent to this user are directed to this layer. In addition, the **who**(C) command will show the user associated with this layer. The **relogin** command may only be invoked under **layers**(C).

relogin is invoked automatically by **layers** to set the *utmp* entry to the terminal line of the first layer created upon startup and to reset the *utmp* entry to the real line on termination. It may be invoked by a user to designate a different layer to receive **write** messages.

-s Suppress error messages.

line Specifies which *utmp* entry to change. The *utmp* file is searched for an entry with the specified ***line*** field. That field is changed to the line associated with the standard input. To learn what lines are associated with a given user, say *jdoe*, enter:

ps -f -u jdoe

and note the values shown in the "TTY" field (see **ps**(C)).

File

/etc/utmp database of users versus terminals

Diagnostics

Returns 0 upon successful completion, 1 otherwise.

See also

layers(C), **mesg**(C), **ps**(C), **who**(C), **write**(C), **utmp**(F)

Notes

If ***line*** does not belong to the user issuing the **relogin** command, or if standard input is not associated with a terminal, **relogin** will fail.

removef

remove a file from software database

Syntax

removef *pkginst path1* [*path2* ...]

removef -f *pkginst*

Description

removef checks whether a pathname can be removed safely from a particular package. Output from **removef** is the list of input pathnames that may be removed safely (no other packages have a dependency on them).

After all files have been processed, **removef** should be invoked with the **-f** option to indicate that the removal phase is complete.

Example

The following shows the use of **removef** in an optional pre-install script:

```
echo "The following files are no longer part of this package
        and are being removed."
removef $PKGINST /dev/xt[0-9][0-9][0-9] |
while read pathname
do
        echo "$pathname"
        rm -f $pathname
done
removef -f $PKGINST || exit 2
```

See also

installf(ADM), **pkgadd**(ADM), **pkgask**(ADM), **pkgchk**(ADM), **pkginfo**(ADM), **pkgmk**(ADM), **pkgparam**(ADM), **pkgproto**(ADM), **pkgtrans**(ADM)

restore

incremental filesystem backup restore

Syntax

restore [**-c**] [**-i**] [**-o**] [**-t**] [**-d** *device*] [*pattern* [*pattern*] ...]

Description

This utility acts as a front end to **cpio**(C), and thus reads **cpio**-format tapes or floppies. This utility should only be used to restore backups made with the AT&T **backup**(ADM) utility, not **xbackup**(ADM).

-c Complete restore. All files on the tape are restored.

-i Gets the index file off the medium. This only works when the archive was created using **backup**. The output is a list of all the files on the medium. No files are actually restored.

-o Overwrite existing files. If the file being restored already exists, it will not be restored unless this option is specified.

-t Indicates that the tape device is to be used. **Must** be used with the **-d** option when restoring from tape.

-d *device* is the raw device to be used. It defaults to */dev/rdsk/f0q15d* (the 1.2M floppy).

When doing a restore, one or more ***pattern***s can be specified. These patterns are matched against the files on the tape. When a match is found, the file is restored. Since backups are done using full pathnames, the file is restored to its original directory. Metacharacters can be used to match multiple files. The patterns should be in quotes to prevent the characters from being expanded before they are passed to the command. If no patterns are specified, all files are restored. If a pattern does not match any file on the tape, a message is printed.

When end of medium is reached, the user is prompted for the next medium. The user can exit at this point by entering **q**. (This may cause files to be corrupted if a file happens to span a medium.) In general, quitting in the middle is not a good idea.

If the file already exists and an attempt is made to restore it without the **-o** option, the following message is printed:

```
current <file> newer or same age
```

This ***file*** will not be restored.

In order for multi-volume restores to work correctly, the raw device **must** be used.

rmail

submit remote mail received via UUCP

Syntax

rmail *user* ...

Description

rmail interprets incoming mail received via **uucp**(C), passing the processed mail onto **submit**(ADM) for processing by the MMDF mail system. **rmail** is explicitly designed for use with UUCP and the MMDF **submit** program. It is not intended for use by regular users.

rmail performs several conversions on the incoming mail before calling **submit**. The conversions change addresses from the UUCP routing style (lists of hosts separated by the character "!") to the domain style of address used within the MMDF mail system. The incoming message is dealt with in the following manner:

1) The initial "From" (or ">From") line is processed to discover the originating site and the sender of the message. Some UUCP mailers do not supply this information as part of the message body. If the originating site cannot be found from this information, the program environment is inspected for the variable **ACCTSYS**; this is set to the originating system by some implementations of UUCP. The originating system is used as a table lookup value into the MMDF table *rmail.chans*, the file contains site/channel pairs. If a match is found, the resulting channel is used for the submit phase. The default UUCP channel is used if no match is found. The default channel name is specified in the *conf.c* source and can be run-time tailored. Typically, it is UUCP. The existence of this channel is **mandatory** to prevent dropping mail from unknown hosts.

2) The body of the message is inspected looking for any header lines containing addresses; the lines are "From:", "To:", "Cc:", "Bcc:" and "Sender:". By scanning the address chains, the addresses in these lines are converted into "user@known-site.domain" form using the MMDF tables to evaluate whether the mailer knows the site. For this to work properly, the unqualified name of all sites should exist in the appropriate domain tables. The scanning stops when an unknown site is discovered; a composite address will then be created. The "From:" line is treated specially to preserve any comment information that may have been inserted by the originating mailer.

3) The 'Date:' line is re-written into ARPA standard form.

Before **submit** is called, the message is re-written into RFC822/733 form with all addresses obeying the appropriate convention. Any missing header lines are supplied. The destination address for the message is taken from the argument to **rmail**, so the header re-writing which is done does not affect the routing of the message.

See also

mail(C), **submit**(ADM), **uucp**(C)

removepkg

remove installed package

Syntax

removepkg [*software_package*]

Description

The **removepkg** command will remove the AT&T-style ***software_package*** specified as an argument to **removepkg** or will remove the software package the user selects if no argument is given to **removepkg.**

If an argument is specified, **removepkg** will search the list of previously installed packages and remove the first name matched by ***software_package.*** If no name is matched, the user is given an error message.

If no argument is specified, **removepkg** will query the user, via a menu, as to which package to remove.

Notes

You must invoke **removepkg** on the console.

This command does not work on packages installed with **custom**(ADM).

See also

displaypkg(ADM), **installpkg**(ADM)

rmuser, rmgroup, rmpasswd

remove user accounts

Syntax

/tcb/bin/rmuser *users*

Description

rmuser removes user accounts from the system. A user account consists of a line in */etc/passwd*, entries in */etc/group* and a Protected Password database file. **rmuser** removes all three entities from the system.

If no users are specified on the command line then **rmuser** will read standard input for account names, one per line.

rmuser checks there are no currently running processes for the account before removing it.

rmuser uses **ale**(ADM) and two underlying shell scripts, **rmpasswd** and **rmgroup** to do the actual removal and **authck**(ADM) to rebuild the subsystem databases. **ale** and **authck** require the invoking user to have the *auth* subsystem authorization and the *chown* and *execsuid* kernel authorizations.

Files

/etc/passwd	password file
/etc/group	group file
*/tcb/files/auth/?/**	protected password database
/tcb/lib/auth_scripts/rmpasswd	user script
tcb/lib/auth_scripts/rmgroup	group script

See also

ale(ADM), **authcap**(F),

Diagnostics

rmuser returns an exit status of 1 if it was interrupted.

Notes

Because removing users is not allowed on a C2 system, **rmuser** checks for **UIDREUSE=YES** in */etc/default/login* before removing any accounts.

rmuser does not remove all traces of an account: home directories are left intact, any **cron** jobs are not removed and the name of the account is left in the Terminal Control database and some Protected Password entries. In the Terminal Control database, the deleted account name is not removed from the last (un)successful login, and last logout fields of a terminal entry. In the Protected Password entries, the account name is left in the owner field of accounts which the removed account owned, and the password user field of any accounts for which the removed account was authorized to change the password. These remnants in the C2 database files do not affect the system.

Value added

rmuser is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

runacct

run daily accounting

Syntax

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

Description

runacct is the main daily accounting shell procedure. It is normally initiated via **cron**(C). **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**(C)) 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	Any installation-dependent accounting programs can be included here.
CLEANUP	Clean up 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** will rerun the accounting. The 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(ADM), **acctcms**(ADM), **acctcom**(ADM), **acctcon**(ADM), **acct**(FP), **acctmerg**(ADM), **acctprc**(ADM), **acct**(S), **acctsh**(ADM), **cron**(C), **fwtmp**(ADM), **mail**(C), **utmp**(F)

Notes

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 will not be complete.

Standards conformance

runacct is conformant with:

AT&T SVID Issue 2.

sag

system activity graph

Syntax

sag [*options*]

Description

The **sag** command graphically displays the system activity data stored in a binary data file by a previous **sar**(ADM) run. Any of the **sar** data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. The **sag** command invokes **sar** and finds the desired data by string-matching the data column header (run **sar** to see what is available). These *options* are passed through to **sar**:

-s *time* Select data later than *time* in the form *hh* [*:mm*]. *Default is 08:00.*

-e *time* Select data up to *time*. Default is 18:00.

-i *sec* Select data at intervals as close as possible to *sec* seconds.

-f *file* Use *file* as the data source for **sar**. Default is the current daily data file */usr/adm/sa/sadd*.

Other *options*:

-T *term* Produce output suitable for terminal *term*. See **tplot**(ADM) for known terminals. Default for *term* is **$TERM**.

-x *spec* x axis specification with *spec* in the form: "*name* [*op name*] ... [*lo hi*]"

-y *spec* y axis specification with *spec* in the same form as above.

name is either a string that will match a column header in the **sar** report, with an optional device name in square brackets, for example, **r+w/s[dsk-1]**, or an integer value. *op* is +, -, * or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and - have precedence over * and /. Evaluation is left to right. Thus **A / A + B * 100** is evaluated **(A/(A+B))*100**, and **A + B / C + D** is **(A+B)/(C+D)**. *lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *specs* separated by ";" may be given for **-y**. Enclose the **-x** and **-y** arguments in quotation marks (" ") if blanks or line continuations (\ ⟨CR⟩) are included. The **-y** default is:

-y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

Examples

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=``date +%H:%M``
sar -o tempfile 60 15
TE=``date +%H:%M``
sag -f tempfile -s $TS -e $TE -y "r+w/s[dsk]"
```

File

/usr/adm/sa/sadd daily data file for day ***dd***.

See also

sar(ADM), **tplot**(ADM)

sar, sa1, sa2, sadc

system activity report package

Syntax

sar [-ubdycwaqvmnprDSAC] [-o *file*] *t* [*n*]

sar [-ubdycwaqvmnprDSAC] [-s time] [-e time] [-i sec] [-f *file*]

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

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

/usr/lib/sa/sa2 [-ubdycwaqvmnprDSAC] [-s time] [-e time] [-i sec]

Description

sar, in the first instance, samples cumulative activity counters in the operating system at ***n*** intervals of ***t*** seconds, where ***t*** should be 5 or greater. If the **-o** option is specified, it saves the samples in ***file*** in binary format. The default value of ***n*** is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded ***file***, either the one specified by the **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa*dd*** for the current day ***dd***. The start and end times of the report can be bounded via the **-s** and **-e time** arguments in the form **hh[:mm[:ss]]**. The **-i** option selects records at ***sec*** second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-u Report CPU utilization (the default):
%usr, %sys, %wio, %idle portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **-D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

-b Report buffer activity:
bread/s, bwrit/s - transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s - accesses of system buffers;
%rcache, %wcache - cache hit ratios, that is, (1-bread/lread) as a percentage;
pread/s, pwrit/s - transfers via raw (physical) device mechanism. When used with **-D**, buffer caching is reported for locally-mounted remote resources.

-d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification **dsk-** is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque - portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s - number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv - average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency, and data transfer times).

-n Report name cache statistics. The activity reported is:
c_hits, cmisses - number of name cache hits and misses;
hit% - the hit ratio as a percentage.

-y Report TTY device activity:
rawch/s, canch/s, outch/s - input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s - receive, transmit and modem interrupt rates.

-c Report system calls:
scall/s - system calls of all types;
sread/s, swrit/s, fork/s, exec/s - specific system calls;
rchar/s, wchar/s - characters transferred by read and write system calls.
When used with **-D**, the system calls are split into incoming, outgoing, and strictly local calls.

-w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s - number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s - process switches.

-a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc - run queue of processes in memory and runnable;
swpq-sz, %swpocc - swap queue of processes swapped out but ready to run.

-v Report status of process, inode, file tables:
proc-sz, inod-sz, file-sz, lock-sz - entries/size for each table, evaluated once at sampling point;
ov - overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
msg/s, sema/s - primitives per second.

-p Report paging activities:
vflt/s - address translation page faults (valid page not in memory);
pflt/s - page faults from protection errors (illegal access to page) or "copy-on-writes";
pgfil/s - vflt/s satisfied by page-in from file system;
rclm/s - valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:
freemem - average pages available to user processes;
freeswap - disk blocks available for process swapping.

-D Report Remote File Sharing activity:
When used in combination with **-u**, **-b**, or **-c**, it causes **sar** to produce the remote file sharing version of the corresponding report. **-Du** is assumed when only **-D** is specified.

-S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

-A Report all data. Equivalent to **-udqbwcayvmprSDC**.

-C Report Remote File Sharing buffer caching overhead:
snd-inv/s - number of invalidation messages per second sent by your machine as a server.
snd-msg/s - total outgoing RFS messages sent per second.
rcv-inv/s - number of invalidation messages received from the remote server.
rcv-msg/s - total number of incoming RFS messages received per second.
dis-bread/s - number of buffer reads that would be eligible for caching if caching were not turned off. (Indicates the penalty of running uncached.)
blk-inv/s - number of buffers removed from the client cache.

Examples

To see today's CPU activity so far:

sar

To watch CPU activity evolve for 10 minutes and save data:

sar -o temp 60 10

To later review disk and tape activity from that period:

sar -d -f temp

Data gathering

The operating system contains several counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, paging, and Remote File Sharing.

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, with an interval of **t** seconds between samples, and writes in binary format to **ofile** or to standard output. The sampling interval **t** should be greater than 5 seconds; otherwise, the activity of **sadc** itself may affect the sample. If **t** and **n** are omitted, a special record is written. This facility is used at system boot time, when booting to a multi-user state, to mark the time at which the counters restart from zero. For example, the */etc/init.d/perf* file writes the restart mark to the daily data by the command entry:

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

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/root** (see **cron**(C)):

```
0 * * * 6,0 /usr/lib/sa/sa1 3600
0 8-17 * * 1-5 /usr/lib/sa/sa1 3600
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script **sa2**, a variant of **sar**, writes a daily report in file */usr/adm/sa/sardd*. The **/usr/spool/cron/crontabs/root** entry:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

will report 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 */
      struct minfo mi;    /* defined in sys/sysinfo.h */
      struct dinfo di;    /* RFS info defined in sys/sysinfo.h */
      struct rcinfo rc;   /* Client cache info defined in sys/sysinfo.h */
      int minserve, maxserve;     /* RFS server low and high water marks */
      int  szinode;       /* current size of inode table  */
      int  szfile;        /* current size of file table  */
      int  szproc;        /* current size of proc table  */
      int   szlckf;       /* current size of file record header table */
      int   szlckr;       /* current size of file record lock table */
      int  mszinode;      /* size of inode table  */
      int  mszfile;       /* size of file table  */
      int  mszproc;       /* size of proc table  */
      int   mszlckf;      /* maximum size of file record header table */
      int   mszlckr;      /* maximum size of file record lock table */
      long  inodeovf;     /* cumulative overflows of inode table  */
      long  fileovf;      /* cumulative overflows of file table  */
      long  procovf;      /* cumulative overflows of proc table  */
      time_t  ts;         /* time stamp, seconds  */
      int  apstate;
      long  devio[NDEVS][4];      /* device unit information  */
      int                 cachehits;/* number of name cache hits */
      int                 cachemisses;/* number of name cache misses */
#define IO_OPS            0       /* cumulative I/O requests  */
#define IO_BCNT           1       /* cumulative blocks transferred */
#define IO_ACT            2       /* cumulative drive busy time in ticks  */
#define IO_RESP           3       /* cumulative I/O resp time in ticks */
};
```

Files

/usr/adm/sa/sadd	daily data file
/usr/adm/sa/sardd	daily report file
/usr/lib/sa/sa.adrfl	address file

Notes

Output files created with this version of **sar** cannot be interpreted by earlier versions. However, this version interprets older output files correctly.

See also

cron(C), **sag**(ADM), **timex**(ADM)

Standards conformance

sa1, sa2, sadc and **sar** are conformant with:

AT&T SVID Issue 2.

schedule

database for automated system backups

Description

The *schedule* database is used in conjunction with **fsphoto**(ADM) to partially automate system-wide backups. For each filesystem to be backed up, a cyclical schedule of **xbackup**(ADM) or **cpio**(C) levels is specified. (**fsphoto** uses **cpio** for UNIX filesystems and **xbackup** for XENIX filesystems.)

This cyclical schedule (or "cycle") is a list of backup levels to perform (including no backup at all) and a pointer to the last-used element of that list. The pointer is advanced to the next element of the list on a regular basis (each time **fsphoto** is run, usually once per day), starting at the beginning each time it falls off the end. It is advanced, however, only on success — the desired backup must have been successful.

Each entry in the file is on a separate line. Blank and comment lines (beginning with "#") may be placed anywhere. Several keywords are recognized:

site ***sitename***
: ***sitename*** is passed to **fsave**(ADM) as a description to place on each tape label. Usually, ***sitename*** is the name of the company or a building number.

media ***drive*** **k** ***size*** [***size...***] [***format***]
: Device ***drive*** is a floppy disk or tape drive capable of handling volumes with any of the listed ***sizes*** (in kilobytes). If specified, ***format*** is the command used to format the described floppies. This also applies to standard cartridge tapes.

media ***drive*** **d** ***density size*** [***size...***] [***format***]
: Device ***drive*** is a ***density*** BPI magtape drive capable of handling tapes of any of the indicated ***sizes*** (in feet). As with floppy drives, ***format*** is the optional command used to format the described tape.

[0-9] ***size savetime importance marker***
: Description of each backup level, as described in **fsave**(ADM). The defaults are:

```
Level   Size    Savetime        Importance      Marker
 0       -      "1 year"        critical        none
 1       -      "3 months"      necessary       none
 2...7   -      "1 month"       important       none
 8       -      "2 weeks"       useful          none
 9       -      "1 week"        precautionary   none
```

All four fields must be specified. A ***size*** means to use the first size listed in the appropriate **media** ***sizes*** list.

Keywords should be placed before any filesystem backup schedules. A filesystem backup schedule is of the form:

/dev/rfilesys cycle
: The filesystem resident on device */dev/filesys* is to be backed-up according to *cycle*, which is a space-separated list of backup levels (the digits **0** to **9**, passed to **backup**), or the letter **x**, meaning no backup should occur.

  A backup *cycle* must have at least one member, but it may be of any length. Different filesystems may have *cycles* of different lengths.

Here is the default *schedule* file:

```
# SYSTEM BACKUP SCHEDULE
site machinename

# Media Entries
#
# 96 tpi 1.2 MB floppy 0
media /dev/rfd096ds15 k 1200 format /dev/rfd096ds15
# 96 tpi 1.2 MB floppy 1
media /dev/rfd196ds15 k 1200 format /dev/rfd196ds15
# 135 tpi 1.44 MB floppy 0
media /dev/rfd0135ds18 k 1440 format /dev/rfd0135ds18
# 135 tpi 1.44 MB floppy 1
media /dev/rfd1135ds18 k 1440 format /dev/rfd1135ds18
# Cartridge tape 1
# media /dev/rct0 k 60000 125000 150000 tape erase
# Mini cartridge drive (10 MB)
# media /dev/rctmini k 8800 format /dev/rctmini
# Mini cartridge drive (20 MB)
# media /dev/rctmini k 17200 format /dev/rctmini
# Mini cartridge drive (40 MB)
# media /dev/rctmini k 37500 format /dev/rctmini
# 9-track tape drive
# media /dev/rmt0 d 1600 2400 1200 600
#
# Backup Descriptor Table
#        Backup   Vol.   Save for    Vitality        Label
#        level    size   how long    (importance)    marker
         0        -      "1 year"    critical        "a red sticker"
         1        -      "4 months"  necessary       "a yellow sticker"
         2        -      "3 weeks"   useful          "a blue sticker"
         3        -      "1 week"    precautionary   none
# Schedule table
#                 1 2 3 4 5    6 7 8 9 0    1 2 3 4 5    6 7 8 9 0
# Filesystem      M T W T F    M T W T F    M T W T F    M T W T F Method
/dev/rroot        0 3 3 3 3    2 3 3 3 3    1 3 3 3 3    2 3 3 3 3 cpio
# Alternative Schedule for systems with /u filesystems
# /dev/rroot      0 x 3 x 3    2 x 3 x 3    1 x 3 x 3    2 x 3 x 3 cpio
# /dev/ru         3 0 3 3 3    3 2 3 3 3    3 1 3 3 3    3 2 3 3 3 cpio
```

In the first example (no additional filesystems), */dev/rroot* is backed up each day. Once a month a level 0 is done, and level 3 backups are done on most days. Each following Monday, a level 1 or 2 is done to ensure full redundancy.

In the alternate example, */dev/rroot* is backed-up using a level 0 backup the first time **fsphoto** is run (on a Monday), and if that backup is successful, the next (second) time it runs (Tuesday), no backup is performed. If doing nothing is successful, the third time it runs (Wednesday) a level 3 backup occurs. If that backup succeeds, no backup occurs the fourth time (Thursday), but the fifth time **fsphoto** is run (Friday), a level 3 backup is made.

Each time a successful backup at the specified level happens, the pointer advances so that the next run of **fsphoto** (on the next weekday) will do the next backup scheduled for that filesystem. If however, a backup fails (or is interrupted or postponed by the operator) the pointer is not advanced; hence, the next time **fsphoto** is attempted, the same level backup will again be tried so the sequence will not be broken (but the timing may be off).

The larger and more rapidly changing filesystem */dev/ru* is backed up more frequently (each time **fsphoto** is run - once a day - instead of every other time), and the levels used are staggered to prevent having to perform two full-scale backups (like levels 0 or 1) of the large filesystems on the same day. The backup cycle period is also shorter, two weeks instead of four.

The "Method" field defines the backup utility to be used. **cpio** works for both XENIX and UNIX filesystems, but **xbackup** works only on XENIX filesystems.

See also

fsave(ADM), **fsphoto**(ADM), **xbackup**(ADM)

Notes

Keywords and filesystem names must not be preceded by any spaces or tabs.

It is not necessary to specify the name of the "raw" (*/dev/r**) device for each filesystem, but the backups are faster if this is done.

Value added

schedule is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

sd, sdd

start a no-LUID daemon

Syntax

sd *command* [*argument*]

/tcb/files/no_luid/sdd

Description

The **sd** utility is used to start certain daemons in a way consistent with the enforcement of the login user identifier (LUID) (in accordance with C2 requirements). Use of **sd** is only necessary if the kernel is configured to enforce LUID restrictions. If LUID restrictions are in effect, it is not possible to start daemon processes that set their own LUID (for example, when executing a login procedure) directly from a logged-in terminal.

Daemons are normally started from */etc/rc2.d* and set their LUID using the **su**(C) command. Daemons like **cron** that must run specifically without an LUID should be run via **sdd**.

sdd is itself a daemon process, started from **inittab** (see **init**(M)). **sd** sends requests to **sdd** for other daemon processes to be started.

sdd only starts a process if an authorization check is successful. The authority required for each daemon is specified by the file */tcb/files/no_luid/cmdtable*. This file contains entries for daemons, one per line, as follows:

name*:*path*:*subsystem

where ***name*** is the command name passed as the first argument to **sd**, ***path*** is the full path name of the command that will be executed, and ***subsystem*** is the subsystem authorization that the invoking user is required to have. The special value "*" for ***subsystem*** specifies that any user can issue that command.

Example

The default *cmdtable* file includes the following line:

```
cron:/etc/cron:cron
```

If the system cron daemon were to die for any reason, an administrator who had been granted the cron authorization could restart it by issuing the command:

sd cron

See also

su(C), **subsystems**(S)

Files

*/tcb/files/no_luid/**
/etc/inittab

Value added

sd is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

setclock

set the system real-time (time of day) clock

Syntax

setclock [***time***]

Description

The **setclock** command sets the battery-powered, real-time time of day clock to the given ***time***. If ***time*** is not given, the current contents of the battery-powered clock are displayed. The ***time*** must be a combination of digits with the form:

MMddhhmmyy

where ***MM*** is the month, ***dd*** is the day, ***hh*** is the hour, ***mm*** is the minute, and ***yy*** is the last two digits of the year. If ***yy*** is not given, it is taken from the current system time. For example, the command:

setclock 0826150385

sets the time of day clock to 15:03 on August 26, 1985.

File

/etc/setclock

See also

clock(F)

Note

Not all computers have battery-powered real-time time of day clocks. Refer to your computer's hardware reference manual.

Value added

setclock is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

setmnt

establish /etc/mnttab table

Syntax

/etc/setmnt

Description

setmnt creates the */etc/mnttab* table (see *mnttab*(F)), which is needed for both the **mount**(ADM) and **umount**(ADM) commands. **setmnt** reads the 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*** (for example, "hd0") and ***node*** is the root name of that file system. Thus ***filesys*** and ***node*** become the first two strings in the **mnttab**(F) entry.

File

/etc/mnttab

See also

mnttab(F)

Notes

If ***filesys*** or ***node*** are longer than 128 characters, errors can occur.

setmnt silently enforces an upper limit on the maximum number of **mnttab** entries.

setmnt is normally invoked by the */etc/rc2* scripts when the system boots up.

settime

change the access and modification dates of files

Syntax

settime [***mmddhhmm*** [***yy***]] [**-f** ***fname***] ***name*** ...

Description

The **settime** command sets the access and modification dates for one or more files. The dates are set to the specified date, or to the access and modification dates of the file specified via **-f**. Only one of these methods must be used to specify the new date(s). The first ***mm*** is the month number; ***dd*** is the day number in the month; ***hh*** is the hour number (24 hour system); the second ***mm*** is the minute number; ***yy*** is the last two digits of the year and is optional. For example:

settime 1008004583 ralph pete

sets the access and modification dates of files *ralph* and *pete* to Oct 8, 12:45 AM, 1983. Another example:

settime -f ralph john

This sets the access and modification dates of the file *john* to those of the file *ralph*.

Note

Use of **touch**(C) in place of **settime** is encouraged.

See also

touch(C)

sfmt

perform special formatting

Syntax

/etc/sfmt ***device_name***

Description

The **sfmt** command performs low-level formatting, initializes non-standard disk parameters, and performs initial processing of manufacturer-supplied defect lists of the disk ***device_name***. ***device_name*** should be the character-special device representing the whole disk, for example, */dev/rhd00*.

The **sfmt** command must be issued from the "Boot:" prompt, and should be used only if the "type=E" banner appears during power-up.

Low-level disk formatting is usually performed on bundled systems before delivery. If this formatting has not been done, you must format the disk before installing it. You must know the hard disk parameters before you invoke **sfmt**.

File

/dev/rhd?0

Value added

sfmt is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

shutdown

terminate all processing

Syntax

/etc/shutdown [-y] [-g[*hh*:]*mm*] [-i[0156sS]] [-f "*mesg*"] [-f *file*] [su]

Description

The primary function of **shutdown** is to terminate all currently running processes in an orderly and cautious manner. **shutdown** goes through the following steps:

1. All users logged on the system are notified to log off the system by a broadcast message.
2. */etc/init* is called to perform the the actual shutdown.

Only the super user can execute the **shutdown** command.

The options are as follows:

-y Runs the command silently. If this option is not specified, **shutdown** will prompt for confirmation to shut down the system.

-g[*hh*:]*mm* Specifies the number of hours and minutes before shutdown (maximum: 72 hours). 1 minute is the default. (To shut down the system immediately without a grace period, use */etc/haltsys* or */etc/reboot*. Note that these commands should not be used if NFS, TCP/IP or certain other services are running.)

-i[0156sS] Specifies the init level to bring the system to (see **init**(M)). By default, the system is brought to level 0.

-f *"mesg"* *mesg* is a message enclosed in double quotes (" ") to be sent to all terminals warning of the imminent shutdown during the grace period.

-f *file* Similar to the **-f *mesg*** option, but ***file*** is the pathname for a file containing the message.

The optional **su** argument lets the user go single-user without completely shutting down the system. (This option is identical to **-i1** and is present for backwards compatibility with XENIX). Broadcast messages, whether default or defined, are displayed at regular intervals during the grace period. The closer the shutdown time, the more frequent the message:

Time left until shutdown	Frequency of message
Greater than 1 hour	Every hour
Greater than 15 minutes	Every 15 minutes
Less than 15 minutes	Every minute

In general, if no options are specified, **shutdown** behaves as follows:

1. Prompt for confirmation
2. 60-second grace period
3. Bring the system to init level 0
4. Broadcast default message prior to shutdown.

See also

boot(HW), **wall**(ADM)

Diagnostics

The most common error diagnostic that will occur is `device busy`. This diagnostic appears when a particular filesystem could not be unmounted. See **umount**(ADM).

Notes

Once **shutdown** has been invoked, it must be allowed to run to completion and must *not* be interrupted by pressing BREAK or DEL.

shutdown does not work when executed from within a shell layer.

shutdown locks the hard disk heads.

Credit

shutdown was developed at the University of California, Berkeley, and is used with permission.

strace

print STREAMS trace messages

Syntax

strace [***mid sid level***] ...

Description

strace without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver (**log**(M)). If arguments are provided they must be in triplets of the form ***mid, sid, level***, where ***mid*** is a STREAMS module id number, ***sid*** is a sub-id number, and ***level*** is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/driver, sub-id (usually indicating minor device), and priority level equal to or less than the given level. The token ***all*** may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

seq time ticks level flags mid sid text

seq trace sequence number

time time of message in hh:mm:ss

ticks time of message in machine ticks since boot

level tracing priority level

flags E : message is also in the error log
F : indicates a fatal error
N : mail was sent to the system administrator

mid module id number of source

sid sub-id number of source

text formatted text of the trace message

Once initiated, **strace** will continue to execute until terminated by the user.

Examples

Output all trace messages from the module or driver whose module id is 41:

strace 41 all all

Output those trace messages from driver/module id 41 with sub-ids 0, 1, or 2:

strace 41 0 1 41 1 1 41 2 0

Messages from sub-ids 0 and 1 must have a tracing level less than or equal to 1. Those from sub-id 2 must have a tracing level of 0.

See also

log(M)

Diagnostics

Due to performance considerations, only one **strace** process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the **strace** process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance. Running **strace** will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the **strace** process. If trace messages are generated faster than the **strace** process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

strclean

STREAMS error logger cleanup program

Syntax

strclean [-d *logdir* **] [-a** *age* **]**

Description

The **strclean** command is used to clean up the STREAMS error logger directory on a regular basis (for example, by using **cron**(C)). By default, all files with names matching *error.** in */usr/adm/streams* that have not been modified in the last 3 days are removed. A directory other than */usr/adm/streams* can be specified using the **-d** option. The maximum age in days for a log file can be changed using the **-a** option.

Example

strclean -d /usr/adm/streams -a 3

has the same result as running **strclean** with no arguments.

Notes

strclean is typically run from **cron**(C) on a daily or weekly basis.

File

*/usr/adm/streams/error.**

See also

cron(C), **strerr**(ADM)

strerr

STREAMS error logger daemon

Syntax

strerr

Description

The **strerr** daemon receives error log messages from the STREAMS log driver (see **log**(M)) and appends them to a log file. The error log files produced reside in the directory */usr/adm/streams*, and are named *error.mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

seq time ticks flags mid sid text

seq	error sequence number	
time	time of message in hh:mm:ss	
ticks	time of message in machine ticks since boot priority level	
flags	T	the message was also sent to a tracing process
	F	indicates a fatal error
	N	send mail to the system administrator
mid	module id number of source	
sid	sub-id number of source	
text	formatted text of the error message	

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via **mail**(C). The priority level usually has no meaning in the error log but will have meaning if the message is also sent to a tracer process.

Once initiated, **strerr** will continue to execute until terminated by the user. Commonly, **strerr** would be executed asynchronously.

Notes

Only one **strerr** process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

File

*/usr/adm/streams/error.**mm-dd***

See also

log(M)

submit

MMDF mail queue manager

Syntax

/usr/mmdf/bin/submit [-L...*V...*Wbcdf...*g...*hi...*jk...*lmnqrstuvwxyz]

Description

All mail is entered into the MMDF mail transport environment through the **submit** program. This document is intended to provide the specific information needed to control **submit**. While it can be called directly from a user's terminal, access to **submit** is most conveniently performed through a program such as **mail**(C).

Basic modes

submit permits considerable flexibility with respect to batching multiple submissions, response and error handling, and address source specification.

Multiple submissions

1. Terminate after one submission, such as is carried out by the mail command, or

2. Permit multiple message submissions, as is done by the SMTP channel.

The first mode is specified by passing any initialization information in the submit invocation line (that is, the **exec**(S) call). In the second mode, the initialization information is given as the first input line, for each submission. The format of this information is the same for both modes.

Response & error handling

1. Accept input until error or end of message, but terminate on any error, or

2. Notify result for each ***segment*** and continue.

Response mode #1 is mandatory with Multiple mode #1. Response mode #2 is called "protocol mode". During it, each address produces a status reply and the message text produces a reply. The domain of the term ***segment*** depends on the error. Simple addressing errors cause rejection only of the erroneous address. Other errors may cause rejection of the entire message, but permit submission of following messages.

Addresses

1. Extracted from components of the message text,

2. Explicit list given, ahead of message text, or

3. Both of the above (extracted and explicit addresses)

The first mode is common when mode #1 (non-protocol) is also in force for the Interaction and the Verification option. The second mode is commonly in force when the second modes apply for the other options (protocol mode).

Initialization

A message's initialization information is specified through a single string, passed either in the process-invocation argument list or in the first line of **submit** input. Hence, the string may be terminated either by a null or newline. Spaces and tabs in the line are ignored, unless part of a literal. Specification is only required for non-defaults.

	Option	Value	Literal
1.	Relay source for the "Via" or "Received" field	a. none b. source channel c. source host	(default) i...* h...*
2.	From/Sender authentication	a. reject on error b. trust c. no trust (disclaim)	(default) t u
3.	"Source-Info" field	a. not included b. disclaim author c. user text	(default) u f...*
4.	Address list source	a. explicit list b. extract from components c. both (extract and explicit)	(default) g...*
5.	Address verification	a. abort on invalid b. report on each address	(default) v
6.	Delivery destination	a. mailbox b. user's tty c. mailbox and tty	m (default) y b
7.	Delivery attempt (combinable)	a. leave for daemon b. deliver local now c. deliver netmail now	(default) l n

(Continued on next page)

(Continued)

	Option	Value	Literal
8.	Observation of immediate attempts	a. none b. user will watch	(default) w
9.	Return address	a. send to submittor b. send to "Sender:" c. do not return d. as specified	r s q (next line)
10.	Returned mail contents	a. entire original b. citation only	(default) c
11.	Warnings	a. send warnings b. do not send warnings	(default) z
12.	Delay channel usage	a. enable delay channel b. don't use delay	(default) d
13.	Delay channel indicator	a. not delay channel b. delay channel	(default) j
14.	Nameserver timeouts	a. short timeouts b. as specified	(default) k...*
15.	Submission tracing	a. not shown b. watch submission	(default) W
16.	Logging file	a. as per msglog b. as specified	(default) L...*
17.	Logging level	a. as per msglog b. as specified	(default) V...*

Comments

General:

Literals shown as characters, followed by an ellipsis, followed by an asterisk (for example x...*), represent a string. The first character specifies the nature of the setting. The value for the setting is placed between that character and the asterisk. The value may be any string not containing an asterisk, null, or newline. The values for settings **x** and **g** are comma-separated lists of strings. These strings may not contain asterisks, nulls, newlines, or commas.

Specific:

1. Relaying
   This is used when the calling program is interfacing with another distribution system, effecting relaying. The literal after the **i** specifies the channel the message is coming from. The **h** may be used, in conjunction with **i**, to specify the source host. The literal is the name of the host.

2. From/Sender authentication
   Normally, the message must correctly identify its sender. Anyone may send "anonymous" (unsigned) mail, but they must use the **u** setting which bypasses authentication. However, it also causes MMDF to include, in the "Source-Info:" component, a statement noting the absence of authentication. Only *root* or relays may use the **t** setting, which bypasses authentication and does not add a disclaimer. Others requesting it get **u** treatment.

3. Source-Info field
   In addition to the action explained above, "Source-Info:" can directly receive text, from the user, through the **f** setting. The value string is replicated on a separate line in the field.

4. Address list source
   An explicit list has one address per line. When **x** or **g** are specified, they list the names of message components, such as "To:" and "CC:", which are to be searched for addresses.

5. Address verification
   Normally, any illegal address will cause the entire message to be rejected. In **v** (verify) mode, the acceptability of each message is reported and encountering an illegal address does not abort submission.

6. Delivery destination
   Mail may be delivered to a recipient's *mailbox* (file), online terminal (if the recipient is logged in), or a combination of the two. There is no default. For each message, its delivery mode must be specified.

7. Delivery attempt
   An immediate attempt causes a special **deliver** process to be forked and it will attempt to process the indicated mail immediately. (The **n** setting does not allow more granularity, for historical reasons.) Otherwise, the system's background daemon will get to it eventually. The daemon also handles mail that initially could not be delivered/relayed. A channel's descriptor structure (in *chan.c* or the runtime tailor file) specifies a channel as being Active, Passive, or Background. Only the first is processed by any request for immediate delivery. The second indicates a Post Office Box-style channel. The third limits the channel to processing by the background **deliver** daemon, which may be necessary for restricting access to special channels, such as dial-out telephones.

8. Observation
   If an immediate attempt is requested, the user may elect to watch its progress. **deliver** and its children will report assorted aspects of their activity. If a quiet attempt is requested, **submit** returns as soon as submission is completed. That is, a quiet attempt is performed detached.

9. Return address
   If the invoker of **submit** is not to receive return mail (e.g., notification of delivery failure) then the next input line (the first, if settings are specified in the **exec**(S) call), contains an address that should receive the notification. It is not validated. If either the **r** or the **s** switch is given, **submit** will not read a line for the return address. If no return mail should be sent, the return address line should be empty (i.e., consist of a newline, only.) If the **q** switch is given, a return address is read from the next line of input but the local system will not return mail if delivery problems are encountered. The return address given may be used by other systems (if there are mail relays between the local system and the recipient).

10. Returned mail contents
    Normally, a copy of the entire message is sent with a delivery-failure notice. Using the **c** switch causes a citation, comprising the message header and first three lines of non-blank lines of the body, to be sent. If more than 12 addresses are specified, for a message, citation-only is automatically set. In addition, no warning message will be sent for addresses which take a long time to process (a site dependent value); the final failure notice will always be sent, if there are addresses that are never fully processed.

11. Warnings
    Normally MMDF will send a non-delivery warning if a message has been undelivered after a small period (typically 12 to 72 hours, depending on the site). Deliver attempts continue until a timeout period is reached. This is typically after 3 to 10 days, depending on the site.

12. Delay channel usage
    The delay channel is used to process mail submissions that could not be queued because necessary nameserver information was unavailable and therefore an authoritative decision on the validity of the address was not possible. If the **d** option is specified, use of the delay channel is prohibited. If the nameserver fails, an error is returned, rather than a conditional **OK**.

13. Delay channel indicator
    This option is intended only to be used by the delay channel itself to indicate to **submit** that the invoking process *is* the delay channel. This option implies the **d** option above.

14. Nameserver timeouts
    By default, MMDF uses a short timeout algorithm. This is suitable for user interface programs which do not want to wait a long time for dead nameservers. The **k** option allows a different timeout to be set. The value given is the number of seconds to wait for the nameserver lookup to complete.

15. Submission tracing
    The **W** option causes submit to print a detailed description of its activities on file descriptor 2. It will indicate, for each addressee, the channel and addresses queued. This can generate a great deal of output if a mailing list is encountered, so it should be used with caution.

16. Logging file
    The **L** option allows the specification of an alternate logging file at runtime. The string following the **L** should be the name of the logfile to be used. It can be terminated by a "*" or the end of the arguments. This option is only available to the super user or MMDF.

17. Logging level
    The **V** option allows the setting of the logging level at runtime. The string following the **V** should be one of the valid MMDF logging level strings such as **FTR** or **BST**. It can be terminated by a "*" or the end of the arguments. This option is only available to the super user or MMDF.

Input stream

The following augmented BNF characterizes **submit**'s input (file descriptor zero) format:

stream:	*(init-seq '\n' msg-info null) [null]
init-seq:	*{ switches listed above }
msg-info:	[ret-addr] '\n' [addr-seq '!' '\n'] { rfc822-format message }
ret-addr:	{ rfc822-format (return) address }
addr-seq:	*{ rfc822-address }

Address format

Addresses are expected to conform to the ARPANET mail standard known as RFC-822, available from the Network Information Center at SRI International. **submit** (and MMDF in general) also continues to support RFC-733 style mail for compatibility with earlier mail systems.

In addition to those in RFC-822, the following address delimiters are recognized within the local part of addresses (in order of precedence):

@
%
!
.

The "!" delimiter is interpreted as "host!user" while the others are interpreted as "user?host". For example, the address "a.b!user%c@localhost" would be queued for "a.b!user@c". The address "a.b!user@localhost" would be queued for "user@a.b". The address "user.a@localhost" would be queued for "user@a". Note that recognition of the "." delimiter is a site-selectable option.

Also, addresses may be indirectly referenced, through a file specification of the form:

"*<filename*" or "*:include:filename*"

where the angle-bracket must be the first non-blank character of the specification (to distinguish it from the "<...>" usage, above).

Addresses in the file may be separated by commas or newlines.

Example interactions

Phases involve Invocation (Invoke), data sent into **submit** via its file descriptor zero (To), data returned from **submit** via its file descriptor one (From), iteration back to the specified phase (Loop), and process exit value (Exit).

1. Simple, single-message command:

a. Invoke:	Parameters, "-mlrxto,cc*", indicate that the message is to be sent to recipients' mailboxes, local mail should be sent immediately, return mail goes to the submittor, and addresses are to be extracted from the "To:" and "cc:" components.
b. To:	The entire message
c. From:	Error messages
d. Exit:	Process return value, in wait(&val), indicating submission status.

2. Standard, multi-message protocol:

a. Invoke:	No parameters
b. To:	Initialization information line. A typical user program might have "mlrv", indicating the message is to be sent to mailboxes, local mail sent immediately, return mail goes to the sender, and each address verification is to be reported. A relay program might have "mlntviVGR.BRL.MIL*," with "mlv" as above and the other settings indicating that mail for non-local channels is to be sent immediately, the author information is to be trusted, and the "Received:" component should cite the mail as being relayed via Internet host VGR.BRL.MIL.
c. To:	One address, terminated by a newline ('\n').
d. From:	Status character, from *mmdf.h*, plus human-oriented text plus newline.
e. Loop:	Back to (c). Terminate with address line having only an exclamation mark (!), with newline.
f. To:	Message text, in Internet RFC #822 format. Multi-line, terminated by null ('\0').
g. From:	Status character, text, newline.
h. Loop:	Back to (b). Terminate with initialization line having only a null, without newline.

Channels

When MMDF is used in conjunction with the DARPA domain nameserver system, a "delay" channel should be configured to allow queuing of addresses that fail verification temporarily due to nameserver failures (unavailability). Two other special channels that can be configured are the "badusers" and "badhosts" channels. Mail to unknown users or unknown hosts will be queued to these channels if they are configured. The bad channels have no special code associated with them. The channel configuration should reference whatever table and parogram is necessary to reach a smarter host which can deliver or forward the mail. The channel should have the "host=" parameter set to this host name. The channel names given above are reserved.

Files

Numerous. Generally under the MMDF login directory.

See also

deliver(ADM), **mmdf**(S)

Return codes

The following, excerpted from MMDF source, lists the return codes.

```
/*                      Reply Codes for MMDF

 *  Based on: "Revised FTP Reply Codes", by Jon Postel&Nancy Neigus Arpanet
 *      RFC 640 / NIC 30843, in the "Arpanet Protocol Handbook", E.  Feinler
 *      and J. Postel (eds.), NIC 7104, Network Information Center, SRI
 *      International:  Menlo Park, CA.  (NTIS AD-A0038901)
 *
 *  Actual values are different, but scheme is same.  Codes must fit into
 *  8-bits (to pass on exit() calls); fields are packed 2-3-3 and
 *  interpreted as octal numbers.
 *
 *  Basic format:
 *
 *      0yz: positive completion; entire action done
 *      1yz: positive intermediate; only part done
 *      2yz: Transient negative completion; may work later
 *      3yz: Permanent negative completion; you lose forever
 *
 *      x0z: syntax
 *      x1z: general; doesn't fit any other category
 *      x2z: connections; truly transfer-related
 *      x3z: user/authentication/account
 *      x4x: mail
 *      x5z: file system
 *
 *      3-bit z field is unique to the reply.  In the following,
 *      the RP_xVAL defines are available for masking to obtain a field.
 */
/* **************  FIELD DEFINITIONS & BASIC VALUES  ****************** */

/*          Field 1:  Basic degree of success (2-bits)                  */

#define RP_BTYP '200'           /* good vs. bad; on => bad             */
#define RP_BVAL '300'           /* basic degree of success             */
#define RP_BOK  ' 00'           /* went fine; all done                 */
#define RP_BPOK '100'           /* only the first part got done        */
#define RP_BTNO '200'           /* temporary failure; try later        */
#define RP_BNO  '300'           /* not now, nor never; you lose        */


/*          Field 2:  Basic domain of discourse (3-bits)                */

#define RP_CVAL ' 70'            /* basic category (domain) of reply   */
#define RP_CSYN '\000'           /* purely a matter of form            */
#define RP_CGEN '\010'           /* couldn't find anywhere else for it */
#define RP_CCON '\020'           /* data-transfer-related issue        */
```

```
#define RP_CUSR '\030'            /* pertaining to the user            */
#define RP_CMAI '\040'            /* specific to mail semantics        */
#define RP_CFIL '\050'            /* file system                       */
#define RP_CLIO '\060'            /* local i/o system                  */
/*          Field 3:  Specific value for this reply (3-bits)          */

#define RP_SVAL '\007'            /* specific value of reply           */

/* ********************  SPECIFIC SUCCESS VALUES  ******************** */

/*                        Complete Success                            */

#define RP_DONE (RP_BOK | RP_CGEN | '\000') /* done (e.g., w/trans.)   */
#define RP_OK   (RP_BOK | RP_CGEN | '\001') /* general-purpose OK      */
#define RP_MOK  (RP_BOK | RP_CMAI | '\000')
                                  /* message is accepted (w/text)      */
#define RP_DOK  (RP_BOK | RP_CGEN | '\003')
                                 /* accepted for the delayed submission channel  */


/*                        Partial Success                             */

#define RP_MAST (RP_BPOK| RP_CGEN | '\000') /* you are the requestor   */
#define RP_SLAV (RP_BPOK| RP_CGEN | '\001') /* you are the requestee   */
#define RP_AOK  (RP_BPOK| RP_CMAI | '\000') /* message address accepted */
#define RP_HOK  (RP_BPOK| RP_CMAI | '\001') /* host processing complete */

/* ********************  SPECIFIC FALURE VALUES  ********************* */

/*                        Partial Failure                             */

#define RP_AGN  (RP_BTNO | RP_CGEN | '\000') /* not now; maybe later   */
#define RP_TIME (RP_BTNO | RP_CGEN | '\001') /* timeout                */
#define RP_NOOP (RP_BTNO | RP_CGEN | '\002') /* no-op; nothing done    */
#define RP_EOF  (RP_BTNO | RP_CGEN | '\003') /* encountered an EOF     */
#define RP_NET  (RP_BTNO | RP_CCON | '\000') /* channel went bad       */
#define RP_BHST (RP_BTNO | RP_CCON | '\001') /* foreign host screwed up */
#define RP_DHST (RP_BTNO | RP_CCON | '\002') /* host went away         */
#define RP_NIO  (RP_BTNO | RP_CCON | '\004') /* general net i/o problem */
#define RP_NS   (RP_BTNO | RP_CCON | '\005') /* temp nameserver failure */
#define RP_FIO  (RP_BTNO | RP_CFIL | '\000') /* err reading/writing file */
#define RP_FCRT (RP_BTNO | RP_CFIL | '\001') /* unable to create file  */
#define RP_FOPN (RP_BTNO | RP_CFIL | '\002') /* unable to open file    */
#define RP_LIO  (RP_BTNO | RP_CLIO | '\000') /* general local i/o problem */
#define RP_LOCK (RP_BTNO | RP_CLIO | '\001') /* resource currently locked */

/*                        Complete Failure                            */

#define RP_MECH (RP_BNO | RP_CGEN | '\000')
                                  /* bad mechanism/path; try alternate?  */
#define RP_NO   (RP_BNO | RP_CGEN | '\001') /* general-purpose NO      */
#define RP_PROT (RP_BNO | RP_CCON | '\000') /* general prototocol error */
#define RP_RPLY (RP_BNO | RP_CCON | '\001')
                                  /* bad reply code (PERMANENT ERROR)   */
#define RP_NAUTH (RP_BNO | RP_CUSR  | '\001') /* bad authorisation     */
                                 /* SEK this will be used for user checks */
```

```
#define RP_NDEL (RP_BNO | RP_CMAI | '\000') /* couldn't deliver         */
#define RP_HUH  (RP_BNO | RP_CSYN | '\000') /* couldn't parse request   */
#define RP_NCMD (RP_BNO | RP_CSYN | '\001') /* no such command defined  */
#define RP_PARM (RP_BNO | RP_CSYN | '\002') /* bad parameter            */
#define RP_UCMD (RP_BNO | RP_CSYN | '\003') /* command not implemented  */
#define RP_USER (RP_BNO | RP_CUSR | '\000') /* unknown user             */

/*                      STRUCTURE OF A REPLY STRING                     */

struct rp_construct             /* for constant reply conditions      */
{
    char    rp_cval;
    char    rp_cline[50];
};

#define RP_LINEBUF_MAX 256

struct rp_bufstruct             /* for reading reply strings          */
{
    char    rp_val;
    char    rp_line[RP_LINEBUF_MAX];
};

typedef struct rp_bufstruct RP_Buf;

#define rp_conlen(bufnam) (strlen(bufnam.rp_cline) + sizeof(bufnam.rp_cval))

/*              PSEUDO-FUNCTIONS TO ACCESS REPLY INFO                   */

#define rp_gval(val)    ((char) (val))
                                /* get the entire return value        */

/*  The next three give the field's bits, within the whole value       */

#define rp_gbval(val)   (rp_gval (val) & RP_BVAL)
                                /* get the basic part of return value */
#define rp_gcval(val)   (rp_gval (val) & RP_CVAL)
                                /* get the domain part of value       */
#define rp_gsval(val)   (rp_gval (val) & RP_SVAL)
                                /* get the specific part of value     */

/*  The next three give the numeric value withing the field            */

#define rp_gbbit(val)   ((rp_gval (val) >> 6) & 03)
                                /* get the basic part right-shifted   */
#define rp_gcbit(val)   ((rp_gval (val) >> 3 ) & 07)
                                /* get the domain part right-shifted  */
#define rp_gsbit(val)   (rp_gval (val) & 07)
                               /* get the specific part right-shifted */

/*  The following works with SIGNED or UNSIGNED chars!  */
#define rp_isgood(val)  (! rp_isbad(val))
                                /* is return value positive?          */
#define rp_isbad(val)   (rp_gval(val) & 0200)
                                /* is return value negative?          */
```

```
extern char *rp_valstr ();
```

Credit

MMDF was developed at the University of Delaware and is used with permission.

sulogin

access single-user mode

Syntax

sulogin

Description

sulogin is automatically invoked by **init** when the system is first started. It prompts the user to type the root password to enter system maintenance mode (single-user mode) or to type ⟨Ctrl⟩**d** for normal startup (multi-user mode). **sulogin** should never be directly invoked by the user.

File

/bin/sulogin

See also

init(M)

Value added

sulogin is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

swap

swap administrative interface

Syntax

/etc/swap -a ***swapdev swaplow swaplen***

/etc/swap -d ***swapdev swaplow***

/etc/swap -l

Description

The **swap** command provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

-a Add the specified swap area. ***swapdev*** is the name of the block special device, for example, */dev/dsk/1s0*. ***swaplow*** is the offset in 512-byte blocks into the device where the swap area should begin. ***swaplen*** is the length of the swap area in 512-byte blocks. This option can only be used by the super user. Swap areas are normally added by the system start-up routine */etc/rc* when going into multiuser mode.

-d Delete the specified swap area. ***swapdev*** is the name of a block special device, for example, */dev/dsk/1s0*. ***swaplow*** is the offset in 512-byte blocks into the device where the swap area should begin. This option can only be used by the super user.

-l List the status of all the swap areas. The output has four columns:

DEV The ***swapdev*** special file for the swap area if one can be found in the */dev/dsk* or */dev* directories, and its major/minor device number in decimal.

LOW The ***swaplow*** value for the area in 512-byte blocks.

LEN The ***swaplen*** value for the area in 512-byte blocks.

FREE The number of free 512-byte blocks in the area.

Note

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

sync

update the super block

Syntax

sync

Description

The **sync** command executes the **sync** system primitive. If the system is to be stopped, **sync** must be called to ensure filesystem integrity. Note that **shutdown**(ADM) automatically calls **sync** before shutting down the system.

See also

sync(S)

Standards conformance

sync is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

sysadmsh

menu driven system administration utility

Syntax

sysadmsh

Description

sysadmsh is an easy-to-use menu interface designed to provide novice users with the tools needed for day-to-day system administration of the UNIX system.

WARNING: **sysadmsh** does not replace the documentation. It provides an overview of available system administration features and a reminder of tasks which need to be performed regularly. An understanding of the *Installation Guide*, the *System Administrator's Guide*, and the *User's Guide* is necessary to use **sysadmsh**.

Usage

sysadmsh menus can be invoked by logging in as the super user (root) and entering:

sysadmsh

at the shell prompt.

Once you are in **sysadmsh**, on-line instructions for its use may be obtained by selecting the ⟨F1⟩ key.

Some **sysadmsh** options must be run from the system console device. Some options must be run while in single-user (system maintenance) mode. Check the documentation manual page referenced by the menu selection for more information.

Environment variables

sysadmsh uses the following environment variables:

SYSADM is used to find the O/A prompt file *libstrs*, plus the menu, form and help files.

There are three environment variables which **sysadmsh** considers to locate the editor it calls. **SA_EDITOR** is tried first, if this is null then **VISUAL** is tried, then **EDITOR.**

If none of the editor environment variables are set, then one of the following editors is chosen: **/usr/bin/lyrix, /bin/vi** or **/bin/ed** (listed in order of preference).

The following additional environment variables are used:

SA_MAIL If not set, the default mailer is SCO Portfolio **email** if installed, or UNIX **mail**(C) if not.

SA_PRINT If not set, the default printer device is */dev/lp*.

SA_USERAPPS The name of the extensible menu file which describes the User menu area in the top level of **sysadmsh**(ADM). If this is empty or unset, the file *$HOME.sysadmmenu* is used instead.

See also

acctcom(ADM), **accton**(ADM), **asktime**(ADM), **at**(C), **badtrk**(ADM), **checklist**(F), **chgrp**(C), **chmod**(S), **chown**(C), **configure**(ADM), **copy**(C), **cron**(C), **csh**(C), **custom**(ADM), **df**(C), **diff**(C), **dircmp**(C), **disable**(C), **diskcmp**(C), **diskcp**(C), **dmesg**(ADM), **dos**(C), **dtype**(C), **du**(C), **enable**(C), **fdisk**(ADM), **find**(C), **finger**(C), **fixperm**(ADM), **format**(C), **fsck**(ADM), **grpck**(ADM), **init**(M), **kill**(C), **login**(M), **lp**(C), **lpadmin**(ADM), **lpstat**(C), **mail**(C), **mkdev**(ADM), **more**(C), **mount**(ADM), **netutil**(ADM), **ps**(C), **quot**(C), **shutdown**(ADM), **sysadmmenu**(F), **systemid**(F), **tar**(C), **umount**(ADM), **uuinstall**(ADM), **vi**(C), **wall**(ADM), **who**(C), **write**(C)

Installation Guide
System Administrator's Guide
User's Guide

Notes

A knowledge of **vi**(C) is assumed for file edit selections, although the SCO Lyrix® editor is used when available.

Acknowledgements

This utility takes its design from the SCO Lyrix® Word Processing System.

Value added

sysadmsh is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

sysdef

output values of tunable parameters

Syntax

/etc/sysdef [***system_namelist*** [***conf***]]

Description

The **sysdef** command outputs the values of all tunable parameters. It generates the output by analyzing the named operating system file (***system_namelist***) and extracting the configuration information from the name list itself.

Files

/unix	default operating system file (where the system namelist is)
*/etc/conf/**	default directory containing master files

See also

nlist(S)

Diagnostics

`internal name list overflow`
If the master table contains more than an internally specified number of entries for use by **nlist**(S).

Standards conformance

sysdef is conformant with:

AT&T SVID Issue 2.

tcbck, smmck, authckrc

trusted computing base checker
single-user mode tcb check script
multi-user mode tcb check script

Syntax

tcbck

Description

tcbck checks the files in the trusted computing base for files that were caught in the process of being updated when the system went down, and for files that have been removed. **tcbck** is invoked by the scripts **smmck** during system maintenance mode, and by **authckrc** when the system enters multi-user mode. The check proceeds as follows:

1. **smmck** runs **tcbck** to clean up any database files that were left in an interim state while being updated (files are created with **-o** (old) and **-t** (new) suffixes, respectively). When this process is interrupted, **-o** and **-t** files are left and must be reconciled before the system will function properly. **tcbck** checks the */etc/auth/system*, */etc/auth/subsystems*, */tcb/files/auth/** directories and the */etc/passwd* and the */etc/group* files. If there are multiple versions of a file, the extra files are removed. When a **-t** file is found, the following is displayed:

   ```
   /etc/tcbck: file file missing, saved file-t as file
   ```

   This message is repeated for all files found in that state in the specified directories.

2. **tcbck** then checks that key system files are present and that they are not of zero length. If a file is missing (or zero length) then a message similar to this is displayed:

   ```
   /etc/tcbck: file file is missing or zero length
   ```

   This process is repeated for each of the following files:

   /etc/auth/system/default †
   /etc/auth/system/files
   /etc/auth/system/devassign
   /etc/auth/system/authorize †
   /tcb/files/auth/r/root †
   /etc/group
   /etc/passwd †

   When this process is complete, if any files were missing or empty **-t** files were substituted for real files, the following message is displayed:

   ```
   /etc/smmck: restore missing files from backup or distribution.
   ```

3. If critical database files have been removed or corrupted (files marked with a dagger (†) in the previous file list are considered critical) then the system enters maintenance mode automatically without asking for the root password. If no critical database files were lost, the system prompts for maintenance mode or normal operation.

4. **tcbck** then removes the files */etc/auth/system/pw_id_map* and */etc/auth/system/gr_id_map* because the modification times of these files are compared with those of */etc/passwd* and */etc/group* and problems can occur when the system clock is reset. **tcbck** then tries to rebuild the map files using **cps**(ADM). If this fails then either the File Control database (*/etc/auth/system/files*) is missing, or the the File Control database entry for "*/*" is missing, or there are syntax errors in */etc/passwd*, or */etc/group*.

5. After the system goes to init level 2, **authckrc** reinvokes **tcbck** to confirm that the files reported missing previously have been restored: Any missing files are listed, followed by this message:

```
/etc/authckrc: Log in on the OVERRIDE tty and restore
     the missing files from a backup or the distribution disks.
```

   Missing files will have to be replaced when the system comes up multiuser.

6. **authckrc** then runs **passwdupd**(ADM) to check that all users in */etc/passwd* have Protected Password database entries. **authck**(ADM) is then run to check the subsystem databases for errors. Any errors found are repaired automatically. Finally, **ttysupd**(ADM) is run to check that all ttys in */etc/inittab* have entries in the Terminal Control database (*/etc/auth/system/ttys*).

Notes

authckrc, **tcbck**, and **smmck** can only be run as *root*.

Value added

tcbck is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

timex

time a command; report process data and system activity

Syntax

timex [***options*** **]** ***command***

Description

When you use the **timex** utility, the given ***command*** is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the ***command*** and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of **timex** is written on standard error.

Options are:

-p List process accounting records for ***command*** and all its children. This option works only if the process accounting software is installed. Suboptions **f**, **h**, **k**, **m**, **r**, and **t** modify the data items reported. The options are as follows:

- **-f** Print the **fork/exec** flag and system exit status columns in the output.
- **-h** Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as: (total CPU time)/(elapsed time).
- **-k** Instead of memory size, show total kcore-minutes.
- **-m** Show mean core size (the default).
- **-r** Show CPU factor (user time/(system-time + user-time).
- **-t** Show separate system and user CPU times. The number of blocks read or written and the number of characters transferred are always reported.

-o Report the total number of blocks read or written and total characters transferred by ***command*** and all its children. This option works only if the process accounting software is installed.

-s Report total system activity (not just that due to ***command***) that occurred during the execution interval of ***command***. All the data items listed in **sar**(ADM) are reported.

See also

sar(ADM)

Warning

Process records associated with ***command*** are selected from the accounting file */usr/adm/pacct* by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

Examples

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a subshell:

```
timex -opskmt sh
session commands
EOT
```

Standards conformance

timex is conformant with:

AT&T SVID Issue 2.

tplot

graphics filters

Syntax

tplot [-T*terminal* [-e*raster*]]

Description

This command reads plotting instructions (see **plot**(FP)) from the standard input and in general produces, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **$TERM** (see **environ**(M)) is used. Known *terminal*s are:

300 DASI 300.

300S DASI 300s.

450 DASI 450.

4014 Tektronix 4014.

ver VERSATEC D1200A.

This version of **plot** places a scan-converted image in */usr/tmp/raster$$* and sends the result directly to the plotter device, rather than to the standard output. The **-e** option causes a previously scan-converted file *raster* to be sent to the plotter.

Files

/usr/lib/t300
/usr/lib/t300s
/usr/lib/t450
/usr/lib/t4014
/usr/lib/vplot
/usr/tmp/raster$$

See also

plot(FP), **plot**(S), **term**(M)

ttyupd, termupd

update the Terminal Control database

Syntax

/tcb/bin/ttyupd

Description

ttyupd attempts to create Terminal Control database entries for terminals present in the */etc/inittab* file but not present in the Terminal Control database. **ttyupd** calls **ale**(ADM) passing the Terminal Control database and the script **termupd** as parameters.

termupd produces an updated version of the Terminal Control database in the lockfile created by **ale**. **termupd** generates a list of the terminals in */etc/inittab* and substitutes the name of the real terminal for any alias (in the Device Assignment database). Next **termupd** makes a list of the terminals in the Terminal Control database. Finally the Terminal Control database is copied to the lockfile and any terminals appearing in the first list but not the second are added to the end of the lockfile.

Files

/etc/auth/system/files	File Control database
/etc/auth/system/devassign	Device Assignment database
/tcb/lib/auth_scripts/termupd	update shell script
/etc/inittab	script for init process

See also

ale(ADM), **authcap**(F)

Diagnostics

ttyupd returns the exit status of **ale**.

termupd returns 0 if entries are added, 1 if it detects an error and 2 if there are no entries to add. Errors cause appropriate error messages to be displayed.

Notes

Although **ttyupd** is installed with execute permission for all, because it calls **ale**, the *auth* subsystem and *chown* kernel authorizations are required to successfully execute **ttyupd**.

Value added

ttyupd is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

uadmin

administrative control

Syntax

/etc/uadmin ***command function***

Description

The **uadmin** command provides control for basic administrative functions. This command is tightly coupled to the system administration procedures and is not intended for general use. It may only be invoked by the super user.

The arguments ***command*** and ***function*** are converted to integers and passed to the **uadmin** system call.

See also

uadmin(S)

umount

unmount a filesystem

Syntax

/etc/umount ***special-device***

Description

The **umount** command announces to the system that the removable file structure previously mounted on device ***special-device*** is to be removed. Any pending I/O for the file system is completed, and the file structure is flagged as clean. For a detailed explanation of the mounting process, see **mount**(ADM).

File

/etc/mnttab Mount table

See also

mount(ADM), **mount**(S), **mnttab**(F)

Diagnostics

`device busy` An executing process is using a file on the named filesystem, often caused by someone working in the filesystem.

Standards conformance

umount is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

unretire, chtype

change the usertype of an account

Syntax

/tcb/bin/unretire [-t *usertype*] *users*

Description

unretire changes the usertype of an account. By default (without the **-t** flag) **unretire** expects the accounts specified on the command line to be currently "retired" and sets their type back to "general", or "pseudo" if the account has an owner.

Specifying a usertype overides owned accounts being unretired to usertype "pseudo": the other usertypes are: sso, operator and admin. (See **addxusers**(ADM) for an explanation of usertypes.)

unretire can also be used to retire users by specifying a usertype of "retired" (assuming the account is not already retired). When an account is retired, the encrypted password is set to an asterisk (*), further ensuring that the account can no longer be used. Accounts which are logged in cannot have their usertype changed.

If no users are specified on the command line then **unretire** will read standard input for account names, one per line.

unretire uses **ale**(ADM) and the underlying **chtype** shell script. **ale** requires the invoking user to have the *auth* subsystem authorization and the *chown* and *execsuid* kernel authorizations.

Files

*/tcb/files/auth/?/**	protected password database
/tcb/lib/auth_scripts/chtype	change type script

See also

ale(ADM), **authcap**(F), **passwdupd**(ADM)

Diagnostics

unretire returns an exit status of 1 if it was interrupted

Notes

Because the re-use of a user account is not allowed on a C2 system, **unretire** checks for **UIDREUSE=YES** in */etc/default/login* before reactivating an account.

Currently the TCB does not distinguish between pseudo, sso, operator or admin usertypes. They all indicate that the account is not intended to be logged into directly.

Value added

unretire is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

uucheck

check the UUCP directories and permissions file

Syntax

/usr/lib/uucp/uucheck [-v] [-x *debug_level*]

Description

uucheck checks for the presence of the files and directories required by the UUCP system. It also checks for some obvious errors in the *Permissions* file (*/usr/lib/uucp/Permissions*). When executed with the **-v** option, it gives a detailed explanation of how the UUCP programs will interpret the *Permissions* file. The **-x** option is used for debugging. ***debug-option*** is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that **uucheck** can only be used by the super user or *uucp*.

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
*/usr/spool/uucp/**
*/usr/spool/uucppublic/**

See also

uucico(ADM), **uucp**(C), **uusched**(ADM), **uustat**(C), **uux**(C)

Note

The program does not check file/directory modes or some errors in the *Permissions* file such as duplicate login or machine name.

uucico

file transport program for the UUCP system

Syntax

/usr/lib/uucp/uucico [**-r** *role_number*] [**-x** *debug_level*]
[**-i** *interface*] [**-d** *spool_directory*] [**-s**] [**-S**] *system_name*

Description

uucico is the file transport program for UUCP work file transfers. Role numbers for the **-r** are the digit 1 for master mode or 0 for slave mode (default). The **-r** option should be specified as the digit 1 for master mode when **uucico** is started by a program or **cron**. **uux** and **uucp** both queue jobs that will be transferred by **uucico**. It is normally started by the scheduler, **uusched**, but can be started manually; this is done for debugging. For example, the shell **uutry** starts **uucico** with debugging turned on. A single digit must be used for the **-x** option with higher numbers for more debugging.

The **-i** option defines the interface used with **uucico**. This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

The **-d** option can be used to specify the *spool* directory: the default is */usr/spool/uucp*.

If **-s** is specified, a call to the specified site is made even if there is no work for site **sitename** in the spool directory, but call only when times in the *Systems* file permit it. This is useful for polling sites that do not have the hardware to initiate a connection.

The **-S** option can be used to specify the system name, overriding the call schedule given in the *Systems* file. For example, **-S** can be used to call a system which is said to be "Never" called in the *Systems* file.

Changing packet parameters

An additional feature is the ability to change two specialized parameters contained in the **uucico** program without having to recompile the source. (The **uucico** binary is provided unstripped so that patches can be applied using the **/etc/_fst** tool.) The first is a parameter called **windows**, which specifies the size of window that the sliding-window protocol should use (how many packets it can send before getting any ack/nack's from the remote site). **windows** can be changed using a variation of the following patch, which set the value of **windows** to 7:

```
/etc/_fst -w uucico << FST_EOF
$d
_windows/w 7
$q
FST_EOF
```

In addition, the parameter **pktime** can be altered. This is the number of seconds **uucico** should wait before giving up and re-transmitting the packet being sent sent. This interval can be as long as 35 seconds, which can be costly with overseas phone connections. **pktime** can be changed using a variation of the following patch. In this example, **pktime** is set to 5:

```
/etc/_fst -w uucico << FST_EOF
$d
_pktime/w 5
$q
FST_EOF
```

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
*/usr/spool/uucp/**
*/usr/spool/uucppublic/**

See also

cron(C), **uucp**(C), **uusched**(ADM), **uustat**(C), **uutry**(ADM), **uux**(C)

uuclean

UUCP spool directory clean-up

Syntax

/usr/lib/uucp/uuclean [**-C***time*] [**-D***time*] [**-W***time*] [**-X***time*]
[**-m***string*] [**-o***time*] [**-s***system*] [**-x***debug_level*]

Description

uuclean will scan the spool directories for old files and take appropriate action as described below:

- Inform the requestor of send/receive requests for systems that cannot be reached.
- Return mail, which cannot be delivered, to the sender.
- Delete or execute **rnews** for **rnews**-type files (depending on where the news originated, locally or remotely).
- Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that **uuclean** will process as if all option ***times*** were specified to the default values unless ***time*** is specifically set.

The following options are available.

-C*time* Any *C.* files greater than or equal to ***time*** days old will be removed with appropriate information to the requestor. (default 7 days)

-D*time* Any *D.* files greater than or equal to ***time*** days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

-W*time* Any *C.* files equal to ***time*** days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote system. The message includes the JOBID, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**-m** option). (default 1 day)

-X*time* Any *X.* files greater than or equal to ***time*** days old will be removed. The *D.* files are probably not present (if they were, the *X.* could get executed). But if there are *D.* files, they will be taken care of by D. processing. (default 2 days)

-m*string* This line will be included in the warning message generated by the **-W** option. The default line is "See your local administrator to locate the problem".

-o*time* Other files whose age is more than ***time*** days will be deleted. (default 2 days)

-s*system* Execute for ***system*** spool directory only.

-x*debug_level*
The **-x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information.

This program is typically started by the shell **uudemon.clean**, which should be started by **cron**(C). **uuclean** can only be executed by the super user or uucp.

Files

/usr/lib/uucp	directory with commands used by **uuclean** internally
/usr/spool/uucp	spool directory

See also

cron(C), **uucp**(C), **uudemon**(ADM), **uux**(C)

uudemon: uudemon.admin, uudemon.clean, uudemon.hour, uudemon.poll, uudemon.poll2

UUCP administrative scripts

Description

uudemon.admin - collect uustat data

uudemon.clean - merge log files and clean UUCP directories

uudemon.hour - check spool directory for work

uudemon.poll - control polling of passive sites

uudemon.poll2 - alternative polling scheme

UUCP communications and file maintenance can be automated with the use of the **uudemon.hour**, **uudemon.poll**, **uudemon.poll2**, **uudemon.admin**, and **uudemon.clean** shell scripts. While in multi-user mode, **cron** scans files in */usr/spool/cron/crontabs* once each minute for entries to execute at this time. An example crontabs file, *crontab.eg*, is provided to activate these daemons. The system administrator should copy these from */usr/lib/uucp* to */usr/spool/cron/crontabs/uucp*. To do this, log in as user *uucp*, edit the *crontab.eg* file to make any changes, and then enter the following command:

crontab crontab.eg

This will replace the original **crontab** entry.

uudemon.admin

The **uudemon.admin** shell script, as delivered, runs the **uustat**(C) command with **-p** and **-q** options. The **-q** reports on the status of work files (C.), data files (D.), and execute files (X.) that are queued. The **-p** prints process information for networking processes listed in the lock files (*/usr/spool/locks*). It sends resulting status information to the UUCP administrative login (*uucp*) via **mail**(C).

The default **crontab** entry for **uudemon.admin** is:

```
48 10,14 * * 1 - 5 /bin/su uucp -c \
        "/usr/lib/uucp/uudemon.admin" > /dev/null
```

uudemon.clean

The **uudemon.clean** shell script, as delivered, takes log files for individual machines from the */usr/spool/.Log* directory, merges them, and places them in the */usr/spool/.Old* directory with other old log information. If log files get large, the **ulimit** may need to be increased. It also removes work files (C.) 7 days old or older, data files (D.) 7 days old or older, and execute files (X.) 2 days old or older from the spool files. **uudemon.clean** mails a summary of the status information gathered during the current day to the UUCP administrative login (*uucp*).

The default **crontab** entry for **uudemon.clean** is:

```
45 23 * * * ulimit 5000; /bin/su uucp -c \
        "/usr/lib/uucp/uudemon.clean" > /dev/null
```

uudemon.hour

The **uudemon.hour** shell script calls the **uusched**(ADM) program to search the spool directories for work files (C.) that have not been processed and schedules these files for transfer to a remote machine. It then calls the **uuxqt**(ADM) daemon to search the spool directories for execute files (X.) that were transferred to your computer and were not processed at the time they were transferred.

This is the default *root* **crontab** entry for **uudemon.hour**:

```
39,9 * * * * /usr/lib/uucp/uudemon.hour > /dev/null
```

This script runs twice per hour (at 39 and 9 minutes past).

uudemon.poll

uudemon.poll uses the *Poll* (or the alternative *Poll.hour* and *Poll.day*) file (see **poll**(F)) for polling remote computers. The **uudemon.poll** script controls polling but does not actually perform the poll. It merely sets up a polling file (*C.sysnxxxx*) in the */usr/spool/uucp/**nodename*** directory, where ***nodename*** is replaced by the name of the machine. This file will in turn be acted upon by the scheduler (started by **uudemon.hour**). The **uudemon.poll** script is scheduled to run twice an hour just before **uudemon.hour** so that the work files will be there when **uudemon.hour** is called. The default *root* **crontab** entry for **uudemon.poll** is as follows:

```
1,30 * * * * "/usr/lib/uucp/uudemon.poll > /dev/null"
```

uudemon.poll2 is an alternative to **uudemon.poll** that uses a different scheme and different poll files. Listing a site in the *Poll* file gives you control over the lower bound on number-of-calls-per-day (at least as many as you specify in *Poll*), but no control on the upper bound. (This is because **uudemon.poll** uses the the time field of the *Systems* file, which is not suited to the purposes of polling). **uudemon.poll2** permits more control of scheduling. To use **uudemon.poll2**, you must remove the call to **uusched** from **uudemon.hour**, and run **uudemon.poll2** in place of **uudemon.poll** from **cron**. **uudemon.poll2** reads *Poll.hour* (or *Poll.day* if called with the **-d** option) to determine whom to poll much like **uudemon.poll**, but **uudemon.poll2** calls **uucico** directly, using the **-S** option, thus overriding the time field of the *Systems* file.

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/uudemon.admin
/usr/lib/uucp/uudemon.clean
/usr/lib/uucp/uudemon.hour
/usr/lib/uucp/uudemon.poll
/usr/lib/uucp/uudemon.poll2
/usr/lib/uucp/Poll
/usr/lib/uucp/Poll.hour
/usr/lib/uucp/Poll.day

See also

cron(C), **poll**(F), **systems**(F), **uucico**(ADM), **uuclean**(ADM), **uucp**(C), **uusched**(ADM), **uustat**(C)

Value added

uudemon.poll2 is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Standards conformance

uudemon is conformant with:

AT&T SVID Issue 2.

uuinstall

administer UUCP control files

Syntax

/etc/uuinstall [-r]

Description

The **uuinstall** program is used to manage the content of the control files used by the UUCP communications system. It allows users to change the contents of these files without using a text editor. Users need not know the detailed format of each of the control files, although they must be familiar with the function of the various fields within the files. These details are explained in the *System Administrator's Guide.*

The **uuinstall** program can only be executed by the super user. When invoked with the optional **-r** flag, **uuinstall** will not allow any of the files to be modified whether or not the user has made changes to the files.

If **uuinstall** finds any of the required *uucp* control files missing from the system, it will create them with the correct access permissions and ownership.

Files

/etc/systemid
/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices

uulist

convert a UUCP routing file to MMDF format

Syntax

/usr/mmdf/table/tools/uulist

Description

uulist is a conversion utility to produce MMDF-compatible UUCP routing files from the UUCP routing file.

After installing MMDF with **custom**, restore */usr/lib/uucp/Systems* from backup media. Log in as *root* and run the conversion script **/usr/mmdf/table/tools/uulist** from the */usr/mmdf/table* directory. You now have UUCP domain and channel files, *uucp.dom* and *uucp.chn*, in the current directory. Use the **chown** command to make these files owned by *mmdf*. Log out from the super user account.

After creating these files in */usr/mmdf/table*, you must rebuild the MMDF hashed database. Log in as *mmdf* and run **dbmbuild** from */usr/mmdf/table*.

Files

/usr/lib/uucp/Systems
/usr/mmdf/table/uucp.chn
/usr/mmdf/table/uucp.dom

See also

dbmbuild(ADM), *tables*(F)

"Setting up electronic mail" in the *System Administrator's Guide*

Value added

uulist is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

uusched

the scheduler for the UUCP file transport program

Syntax

/usr/lib/uucp/uusched [-x *debug_level*] [-u *debug_level*]

Description

uusched is the UUCP file transport scheduler. It is usually started by the daemon **uudemon.hour** that is started by **cron**(C) from an entry in */usr/spool/cron/crontabs/root*:

```
39,9 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour" > /dev/null
```

The two options are for debugging purposes only; **-x *debug_level*** will output debugging messages from **uusched** and **-u *debug_level*** will be passed as **-x *debug_level*** to **uucico**. The ***debug_level*** is a number between 0 and 9; higher numbers give more detailed information.

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
*/usr/spool/uucp/**
*/usr/spool/uucppublic/**

See also

cron(C), **uucico**(ADM), **uucp**(C), **uudemon**(ADM), **uustat**(C), **uux**(C)

uutry

try to contact remote system with debugging on

Syntax

/usr/lib/uucp/uutry [**-x** *debug_level*] [**-r**] *system_name*

Description

The **uutry** program is a shell script that invokes **uucico** to call a remote site. Debugging is automatically enabled at default level 5; **-x** overrides this value. If **uutry** successfully connects to the remote system, **uutry** stores the debugging output in the file */tmp/system*, where *system* is the name of the remote system. In addition, **uutry** uses **tail -f** to print the last 10 lines of the debugging output to the standard output.

To break out of the shell created by **uutry**, press ⟨Del⟩ or ⟨Break⟩. This returns control to the terminal while **uucico** continues to run, sending the output to */tmp/system*.

The **-r** option overrides the retry time in */usr/spool/uucp/.status*.

Files

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
*/usr/spool/uucp/**
*/usr/spool/uucppublic/**
/tmp/system

See also

uucico(ADM), **uucp**(C), **uux**(C)

uuxqt

execute remote command requests

Syntax

/usr/lib/uucp/uuxqt [**-s** *system*] [**-x** *debug_level*]

Description

uuxqt is the program that executes remote job requests from remote systems generated by the use of the **uux**(C) command. (The **mail**(C) command uses **uux** for remote mail requests). **uuxqt** searches the spool directories looking for *X.* files. For each *X.* file, **uuxqt** checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The *Permissions* file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the **uuxqt** command is executed:

UU_MACHINE is the machine that sent the job (the previous one).

UU_USER is the user that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The **-x** ***debug_level*** is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

Files

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
*/usr/spool/uucp/**

See also

mail(C), **uucico**(ADM), **uucp**(C), **uustat**(C), **uux**(C)

vectorsinuse

display the list of vectors currently specified in the sdevice file

Syntax

/etc/conf/cf.d/vectorsinuse

Description

This script searches the *sdevice* file and displays a list of the interrupt vectors already in use.

You must move to */etc/conf/cf.d* to execute **vectorsinuse**.

When installing a device driver with the Link Kit, you can use **vectorsinuse** to find an available interrupt vector for the driver. When you invoke the **configure** program to modify the system configuration files with the new driver information, use the **-v** option to indicate the vectors on which this device interrupts.

The **-V** option to **configure** performs a function similar to that of **vectorsinuse**. You specify a particular vector on which the device is capable of interrupting (refer to the device's hardware manual), and **configure** tells you if another device is already using that interrupt vector.

File

/etc/conf/cf.d/sdevice

See also

configure(ADM), **sdevice**(F)

Value added

vectorsinuse is an extension to AT&T System V developed by The Santa Cruz Operation, Inc.

volcopy

make literal copy of UNIX filesystem

Syntax

/etc/volcopy [*options*] *fsname srcdevice volname1 destdevice volname2*

Description

The **volcopy** command makes a literal copy of the UNIX filesystem using a blocksize matched to the device. The 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.

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 filesystem is too large to fit on one reel, **volcopy** will prompt 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 (for example, **labelit**) and return to **volcopy** by exiting the new shell.

The *fsname* argument represents the mounted name (for example, *root*, *u1*, etc.) of the filesystem being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (for example: */dev/dsk/0s1* etc.).

The *volname* is the physical volume name (for example: *pk3*, *t0122*, etc.) and should match the external label sticker. Such label names are limited to 6 or fewer characters. *volname* may be "-" (dash) to use the existing volume name.

srcdevice and *volname1* are the device and volume from which the copy of the filesystem is being extracted. *destdevice* and *volname2* are the target device and volume.

fsname and *volname* are recorded in the last 12 characters of the super block (`char fsname[6], volname[6];`).

File

/etc/log/filesave.log a record of filesystems/volumes copied

See also

filesystem(FP), **labelit**(ADM), **sh**(C)

Standards conformance

volcopy is conformant with:

AT&T SVID Issue 2.

wall

write to all users

Syntax

/etc/wall

Description

The **wall** command reads a message from the standard input until an end-of-file. It then sends this message to all users currently logged in preceded by `Broadcast Message from ....` **wall** is used to warn all users, for example, prior to shutting down the system.

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

File

*/dev/tty**

See also

mesg(C), **write**(C)

Diagnostics

`Cannot send to ...` The open on a user's tty file has failed.

wtinit

object downloader for the 5620 DMD terminal

Syntax

/usr/lib/layersys/wtinit [**-d**] [**-p**] *file*

Description

The **wtinit** utility downloads the named *file* for execution in the AT&T TELETYPE 5620 DMD terminal connected to its standard output. The *file* must be a DMD object file. **wtinit** performs all necessary bootstrap and protocol procedures.

There are two options:

-d Prints out the sizes of the text, data, and bss portions of the downloaded *file* on standard error.

-p Prints the downloading protocol statistics and a trace on standard error.

The environment variable **JPATH** is the analog of the shell's **PATH** variable to define a set of directories in which to search for *file*.

If the environment variable **DMDLOAD** has the value **hex, wtinit** will use a hexadecimal download protocol that uses only printable characters.

Terminal Feature Packages for specific versions of AT&T windowing terminals will include terminal-specific versions of **wtinit** under those installation subdirectories. **/usr/lib/layersys/wtinit** is used for **layers**(C) initialization only when no Terminal Feature Package is in use.

Diagnostics

Returns 0 upon successful completion, 1 otherwise.

Notes

Standard error should be redirected when using the **-d** or **-p** options.

See also

layers(C)

xbackup

perform XENIX incremental filesystem backup

Syntax

xbackup [***key*** [***arguments***] ***filesystem***]

Description

xdump - link to xbackup.

xbackup copies all files changed after a certain date in the ***filesystem.*** **xbackup** is used for XENIX filesystems; use **backup**(ADM) for UNIX filesystems. (**xdump** is a link to **xbackup**, retained for historical reasons.) The ***key*** specifies the date and other options about the **xbackup**, where a ***key*** consists of characters from the set *0123456789kfusd*. The meanings of these characters are described below:

f Places the backup on the file specified by the next ***argument*** instead of the default device.

u If the **xbackup** completes successfully, writes the date of the beginning of the **xbackup** to the file */etc/ddate*. This file records a separate date for each filesystem and each *xbackup level.*

0-9 This number is the xbackup level. Backs up all files modified since the last date stored in the file */etc/ddate* for the same filesystem at lesser levels. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire filesystem to be backed up.

s For xbackups to magnetic tape, the size of the tape is specified in feet. The number of feet is taken from the next ***argument***. When the specified size is reached, **xbackup** will wait for reels to be changed. The default size is 2,300 feet.

d For xbackups to magnetic tape, the density of the tape, expressed in BPI is taken from the next ***argument***. This is used in calculating the amount of tape used per write. The default is 1600.

k The size (in K-bytes) of the volume being written is taken from the next ***argument***. If the **k** argument is specified, any **s** and **d** arguments are ignored. The default is to use **s** and **d**.

If no arguments are given, the ***key*** is assumed to be *9u* and a default filesystem is backed up to the default device.

The first xbackup should be a full level-0 xbackup:

xbackup 0u

Next, periodic level 9 xbackups should be made on an exponential progression of tapes or floppies:

xbackup 9u

This progression is shown as follows:

1 2 1 3 1 2 1 4 ...

where xbackup 1 is used every other time, xbackup 2 every fourth, xbackup 3 every eighth, etc.) When the level-9 incremental xbackup becomes unmanageable because a tape is full or too many floppies are required, a level-1 xbackup should be made:

xbackup 1u

After this, the exponential series should progress as if uninterrupted. These level-9 xbackups are based on the level-1 xbackup, which is based on the level-0 full xbackup. This progression of levels of xbackups can be carried as far as desired.

The default filesystem and the xbackup device depend on the settings of the variables **DISK** and **TAPE** respectively, in the file */etc/default/backup*.

Files

/etc/ddate	Records xbackup dates of filesystem/level
/etc/default/backup	Default xbackup information

See also

backup(ADM), **cpio**(C), **default**(F), **restore**(ADM), **sddate**(C), **xbackup**(F), **xdumpdir**(ADM), **xrestore**(ADM)

System Administrator's Guide.

Diagnostics

If the xbackup requires more than one volume (where a volume is likely to be a floppy disk or tape), you will be asked to change volumes. Press ⟨Return⟩ after changing volumes.

Notes

Sizes are based on 1600 BPI for blocked tape. Although the **s** and **d** options are used by default, they are not commonly used; the **k** option is more popular because it specifies size in K-bytes. Write errors to the backup device are usually fatal. Read errors on the filesystem are ignored.

If the default archive medium specified in */etc/default/xbackup* or */etc/default/restor* is block structured, (that is, floppy disk) then the volume size in Kbytes must be specified on the command line. Neither utility works correctly without this information. For example, using the default device (below) with the **xbackup** command, enter the following:

xbackup k 360

The default device entry for **/etc/default/xbackup** (tape=/dev/xxx) and **/etc/default/restor** (archive=/dev/xxx) is **/dev/rfd02.**

It is not possible to successfully **restore** an entire active root filesystem.

When backing up to floppy disks, be sure to have enough **formatted** floppies ready before starting an **xbackup**. You must also be sure to close the floppy door when inserting floppy disks. If you fail to do so in a multi-floppy **xbackup**, the entire **xbackup** will fail and you will have to begin again.

You should never **xbackup** more than one filesystem to the tape devices */dev/nrct0* and */dev/nrct2*. This is because, although **xbackup** can write more than one filesystem to */dev/nrct0* or */dev/nrct2*, **restore** may not be able to restore more than one filesystem from these devices.

Warning

If you have a XENIX filesystem, or have been administering one, it is important to realize that you cannot use backups created by the **xbackup**(ADM) utility. These backups do not allow downsizing when you restore. This is true even if the backed-up filesystem is not full. For example, if you back up a 20 megabyte filesystem that is only 50 percent full, you still won't be able to restore the backup volumes onto a 15 megabyte filesystem. The reinstallation chapter explains that you must use **cpio**(C)-based utilities (such as the **sysadmsh** Backups ⇨ Create selection) to back up your system.

Value added

xbackup is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

xdumpdir

print the names of files on a XENIX backup archive

Syntax

xdumpdir [**f** *filename*]

Description

xdumpdir is used to list the names and inode numbers of all files and directories on an archive written with the **xbackup** command. This is most useful when attempting to determine the location of a particular file in a set of backup archives.

The **f** option causes *filename* to be used as the name of the backup device instead of the default. The backup device depends on the setting of the variable **TAPE** in the file */etc/default/xdumpdir*. The device specified as **TAPE** can be any type of backup device supported by the system (for example, a floppy drive or cartridge tape drive).

File

*rst** Temporary files

See also

default(F), **xbackup**(ADM), **xrestore**(ADM)

Value added

xdumpdir is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

xinstall

XENIX installation shell script

Syntax

/etc/xinstall [*device*]

Description

xinstall is the **sh**(C) script used to install XENIX distribution (or application program) floppies. It performs the following tasks:

Prompts for insertion of floppies.

Extracts files using the **tar**(C) utility.

Executes */once/init.** programs on each floppy after they have been extracted.

Removes any */once/init.** programs when the installation is finished.

The optional argument to the command specifies the device used. The default device is */dev/xinstall* and this is normally linked to */dev/rdsk/f0q15dt*.

Files

/etc/xinstall
*/once/init.**

See also

custom(ADM), **fixperm**(ADM), **installpkg**(ADM)

Notes

xinstall is provided for use with any existing XENIX packages you may have that you wish to install on a UNIX system. **xinstall** does not work with XENIX system applications (use **installpkg**(ADM) to install UNIX system applications).

Value added

xinstall is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

xrestore, xrestor

invoke XENIX incremental filesystem restorer

Syntax

xrestore ***key*** [***arguments***]

xrestor ***key*** [***arguments***]

Description

The **xrestore** command is used to read archive media backed up with the **xbackup**(ADM) command.

The ***key*** specifies what is to be done. ***key*** is one of the characters **cC, rR, tT,** or **xX** optionally combined with **k** and/or **f** or **F**. **xrestor** is an alternate spelling for the same command.

c,C Verify (check) a dump tape. Used after a dump is made to make sure the tape has no I/O errors or bad checksums. **C** is the same as **c** except that it provides a higher level of checking.

f Uses the first ***argument*** as the name of the archive (backup device */dev/**) instead of the default.

F **F** is the number of the first file on the tape to read. All files up to that point are skipped.

k Follow this option with the size of the backup volume. This allows for reading multi-volume dumps from media such as floppies.

r,R The archive is read and loaded into the file system specified in ***argument***. This should not be done without proper consideration (see below). If the key is **R**, **xrestore** asks which archive of a multi-volume set to start on. This allows **xrestore** to be interrupted and then restarted (an **fsck** must be done before the restart).

t Prints the date the archive was written and the date the file system was backed up.

T Prints a full listing of a dump tape. Similar to **t**.

x Each file on the archive named by an ***argument*** is extracted. The filename has all "mount" prefixes removed; for example, if */usr* is a mounted file system, */usr/bin/lpr* is named */bin/lpr* on the archive.

The extracted file is placed in a file with a numeric name supplied by **xrestore** (actually the inode number). In order to keep the amount of archive read to a minimum, the following procedure is recommended:

1. Mount volume 1 of the set of backup archives.
2. Type the **xrestore** command with the appropriate key and arguments.
3. **xrestore** will check **xdumpdir**, then announce whether or not it found the files, give the numeric name that it will assign to the file, and in the case of a tape, rewind to the start of the archive.
4. It then asks you to "mount the desired tape volume". Type the number of the volume you choose. On a multi-volume backup, the recommended procedure is to mount the last through the first volumes, in that order. **xrestore** checks to see if any of the requested files are on the mounted archive (or a later archive, thus the reverse order). If the requested files are not there, **xrestore** doesn't read through the tape. If you are working with a single-volume backup or if the number of files being **xrestore**d is large, respond to the query with **1** and **xrestore** will read the archives in sequential order.

X Same as x except that files are replaced in original location. When you use this option, omit the initial slash (/) in the filename on the **xrestore** command line.

The **r** option should only be used to xrestore a complete backup archive onto a clear file system, or to xrestore an incremental backup archive onto a file system so created. It should not be used to xrestore a backup archive onto the root file system. Thus:

/etc/mkfs /dev/hd1 10000
xrestore r /dev/hd1

is a typical sequence to xrestore a complete backup. Another **xrestore** can be done to get an incremental backup in on top of this.

A **xbackup** followed by a **mkfs** and an **xrestore** is used to change the size of a file system.

Files

*rst**	Temporary files
/etc/default/restor	Name of default archive device

The default archive unit varies with installation.

Notes

It is not possible to successfully **xrestore** an entire active root file system.

Note also that **xrestore** may be unable to xrestore more than one filesystem from the tape devices */dev/nrct0* and */dev/nrct2*.

Diagnostics

There are various diagnostics involved with reading the archive and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one disk or tape, **xrestore** may ask you to change disks or tapes. Reply with a newline when the next unit has been mounted.

See also

fsck(ADM), **mkfs**(ADM), **sddate**(C), **xbackup**(ADM), **xdumpdir**(ADM)

Value added

xrestor and **xrestore** are extensions of AT&T System V developed by The Santa Cruz Operation, Inc.

xtd

extract and print xt driver link structure

Syntax

xtd [**-f**] [*-n* ...]

Description

The **xtd** command is a debugging tool for the **xt**(HW) driver. It performs an **XTIOCDATA** **ioctl**(S) call on its standard input file to extract the `link` data structure for the attached group of channels. This call will fail if data extraction has not been configured in the driver or the standard input is not attached to an **xt**(HW) channel. The data is printed one item per line on the standard output. The output should probably be formatted via **pr -3**.

The optional flags affect output as follows:

-n *n* is a number in the range 0 to 7. Channel **n** is included in the list of channels to be printed. The default prints all channels, whereas the occurrence of one or more channel numbers implies a subset.

-f Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

Diagnostics

Returns 0 upon successful completion; 1 otherwise.

See also

ioctl(S), **pr**(C), **xt**(HW), **xts**(ADM), **xtt**(ADM), **xtproto**(M)

xts

extract and print xt driver statistics

Syntax

xts [-f]

Description

The **xts** command is a debugging tool for the **xt**(HW) driver. It performs an **XTIOCSTATS ioctl**(S) call on its standard input file to extract the accumulated statistics for the attached group of channels. This call will fail if statistics have not been configured in the driver, or the standard input is not attached to a **xt**(HW) channel. The statistics are printed one item per line on the standard output.

-f Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

Diagnostics

Returns 0 upon successful completion; 1 otherwise.

See also

ioctl(S), **xt**(HW), **xtd**(ADM), **xtproto**(M), **xtt**(ADM)

xtt

extract and print xt driver packet traces

Syntax

xtt [-f] [-o]

Description

The **xtt** command is a debugging tool for the **xt**(HW) driver. It performs an **XTIOCTRACE ioctl**(S) call on its standard input file to turn on tracing and extract the circular packet trace buffer for the attached group of channels. This call will fail if tracing has not been configured in the driver, or the standard input is not attached to an **xt**(HW) channel. The packets are printed on the standard output.

The optional flags are:

-f Causes a "formfeed" character to be put out at the end of the output for the benefit of page-display programs.

-o Turns off further driver tracing.

Diagnostics

Returns 0 upon successful completion; 1 otherwise.

Note

If driver tracing has not been turned on for the terminal session by invoking **layers**(C) with the **-t** option, **xtt** will not generate any output the first time it is executed.

See also

ioctl(S), **layers**(C), **layers**(M), **xt**(HW), **xtd**(ADM), **xts**(ADM),

File Formats (F)

Intro

introduction to file formats

Description

This section outlines the formats of various files. The C `struct` declarations for the file formats are given where applicable. Usually, these structures can be found in the directories */usr/include* or */usr/include/sys*. Note that *include* files are part of the Development System.

aio

AIO tunable parameters

Description

The Asynchronous I/O (AIO) facility is an installable package typically used by special purpose applications, such as database servers. There are a set of parameters in the *mtune* file associated with AIO; if necessary (for example, console messages indicate running out of AIO resources), these can be altered by using **idtune**(ADM). The major tunable parameters are as follows:

NAIOPROC the number of processes which may be simultaneously doing AIO. The default value is 5.

NAIOREQ the maximum number of pending AIO requests. The default value is 120.

NAIOREQPP The maximum number of AIO requests that a single process can have pending. The default value is 120, meaning that one process can potentially exhaust all AIO resources.

The AIO tunable parameters that follow are unlikely to need changing, but are documented here for completeness:

NAIOBUF The number of AIO buffers. In the current implementation, this should always be set to the same value as **NAIOREQ**.

NAIOHBUF The number of AIO hash queues (internal).

NAIOLOCKTBL number of entries in the internal kernel table for AIO lock permissions. The default value is 10. If there are many entries in the */usr/lib/aiomemlock* file, this number might need to be increased.

File

/usr/lib/aiomemlock

See also

aio(M), **aiolkinit**(ADM), **aiomemlock**(F), **idtune**(ADM), **mtune**(F), **stune**(F)

aiomemlock

AIO memory lock permissions file

Description

The Asynchronous I/O (AIO) facility allows a system administrator to specify certain privileged users who are allowed to lock a piece of physical memory for AIO usage. The */usr/lib/aiomemlock* file is read by the **aiolkinit**(ADM) utility to set up the AIO locking permissions.

aiomemlock file entries have the following format:

user name ***total lockable memory***

The following entry allows user Lucinda to lock slightly less than 4 megabytes of memory. This is the maximum total shared among all processes running with the UID of "lucinda."

lucinda 4000000

A memory amount of 0 disables any locking for that user. If there is more than one entry for the same user, the second entry overrides the first.

Comment lines have a number sign (#) as the first character.

Notes

The super user can lock memory whether or not root has entry in the *aiomemlock* file.

Memory locked by a user is removed from the general free memory pool. Allowing locks should be done with care; for example, locking four megabytes on an eight megabyte system would severely degade multiuser performance.

File

/usr/lib/aiomemlock

See also

aio(M), **aioinfo**(ADM), **aiolkinit**(ADM)

archive

default backup device information

Description

/etc/default/archive contains information on system default backup devices for use by **sysadmsh**(ADM). The device entries are in the following format:

name=value [***name=value***] ...

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

The following names are defined for */etc/default/archive*:

bdev	Name of the block interface device.
cdev	Name of the character interface device.
size	Size of the volume in either blocks or feet.
density	Volume density, such as 1600. If this value is missing or null, then **size** is in blocks; otherwise the **size** is in feet.
format	Command used to format the archive device.
blocking	Blocking factor.
desc	A description of the device, such as "Cartridge Tape".

See also

sysadmsh(ADM)

authcap

authentication database

Description

The database contains authentication and identity information for users, kernels, and Trusted Computing Base files as well as system-wide parameters. It is intended to be used by programs to interrogate user and system values, as well as by authentication programs to update that information.

Structure of the hierarchies

The complete database resides in two hierarchies: */tcb/files/auth* and */etc/auth*. The first hierarchy deals with user-specific files, and has subdirectories of one letter each of which is the starting letter for user name. Within each of these directories are files, each containing an **authcap**(F) format file for a particular user. Thus, all user names beginning with x have their respective authentication and identity information in a file in directory */tcb/files/auth/x*.

The directories within */etc/auth* contain system-wide information. The global system settings reside in the */etc/auth/system* directory. The subsystem authorizations associated with each protected subsystem (a protected subsystem is privileged but does not require global authority to perform actions) are located in the */etc/auth/subsystems* directory.

The following database files are contained in the *system* directory:

default	Default Control
files	File Control
ttys	Terminal Control
authorize	Primary and Secondary Authorization Control File
devassign	Device Assignment

A subsystem filename is the group name associated with the protected subsystem. The owner of all files is *auth* and the group is the group of the subsystem. Only the owner and group of this file may view the contents. The file *dflt_users* lists the users granted default subsystem authorizations.

Format of a file

Each data file in the hierarchy, whether system-wide or user-specific, has the same format. Each user file consists of one virtual line, optionally split into multiple physical lines with the "\" character present at the very end of all lines but the last. For instance, the line

```
blf:u_name=blf:u_id#16:u_encrypt=a78/a1.eitfn6:u_type=sso:chkent:
```

may be split into:

```
blf:u_name=blf:u_id#16:\
        :u_encrypt=a78/a1.eitfn6:\
        :u_type=sso:chkent:
```

Note that all capabilities must be immediately preceded and followed with the ":" separator; multiple line entries require additional ones — one more per line. Multiple entries are separated by a newline:

```
drb:u_name=drb:u_id#75:u_maxtries#9:u_type=general:chkent:
blf:u_name=blf:u_id#76:u_maxtries#5:u_type=general:chkent:
```

For subsystem files, the file is a set of lines, each containing a user name terminated by a colon, followed by a comma-separated list of primary and secondary authorizations defined for that subsystem.

Format of a line

The format of a line (except for subsystem files) is briefly as follows:

***name* | *alt name(s)* | *description*:*cap1*:*cap2*:*cap3*:...:*capn*:chkent:**

The entry can be referenced by the name or any of the alternate names. A description field may document the entry. The entry name(s) and description are separated by the " | " character. The end of the name/description part of the entry is terminated by the ":" character. Alternate names and the description fields are optional.

At the end of each entry is the "chkent" field. This is used as an integrity check on each entry. The **authcap**(S) routines will reject all entries that do not have "chkent" at the very end.

Each entry has 0 or more capabilities, each terminated with the ":" character. Each capability has a unique name. Numeric capabilities have the format:

id#num

where ***num*** is a decimal or (0 preceded) octal number. Boolean capabilities have the format:

id or ***id@***

where the first form signals the presence of the capability and the second form signals the absence of the capability. String capabilities have the format:

id=string

where ***string*** is 0 or more characters. The "\" and ":" characters are escaped as "\\" and "\:" respectively. Although it is not recommended, the same ***id*** may be used for different numeric, boolean, and string capabilities.

See also

getprpwent(S), **getdvagent**(S), **getprtcent**(S), **getprfient**(S)

Value added

authcap is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

btld

contents of a boot time loadable device driver disk

Disk contents

/install/INSTALL
*/**pkg**/install/btld*
[*/**pkg**/install/drivers*]
[*/**pkg**/install/**pkg**.name*]
[*/**pkg**/install/copyright*]
[*/**pkg**/install/preinstall*]
[*/**pkg**/install/postinstall*]
[*/**pkg**/install/bootstring*]
[*/**pkg**/new/...*]
[.../***xnamex***/*Master*]
[.../***xnamex***/*System*]
[.../***xnamex***/*Bootload*]
files
filename
...
tune
field symbol size
...
patch
symbol size value
...
[.../***xnamex***/*Driver.o*]
[.../***xnamex***/*Space.o*]
[.../***xnamex***/*Space.c*]
[.../***xnamex***/*Stubs.c*]
[.../***xnamex***/*Node*]
[.../***xnamex***/*Inittab*]
[.../***xnamex***/*Rc*]
[.../***xnamex***/*Shutdown*]

Description

boot(HW) can link-edit additional device drivers into the UNIX kernel being booted. These modules are known generically as "Boot Time Loadable Drivers" or BTLDs.

The link-edited modules must be arranged, usually on a floppy disk, in a series of "packages". Each package (***pkg***) contains one or more drivers. There can be more than one package on a disk, but each package must be fully contained on one disk.

Each disk must be a mountable filesystem of a type recognized by both **boot** and **installpkg**(ADM); these include XENIX, S51K, and AFS - see **filesystem**(F). Each package has its own directory hierarchy */pkg* in this filesystem. Optional and required files on a BTLD disk are described below. Note that optional files are indicated with [square brackets] in the preceding "Disk contents" section.

/install/INSTALL

This file must exist and be an executable Bourne shell (**sh**(C)) script. It is run by **installpkg** with three arguments:

device the name of block special floppy diskette device (for example, */dev/fd096ds15*)

rootdir the floppy filesystem's root directory

prompt a string describing ***device*** (useful when prompting)

There can only be one **INSTALL** script per diskette. It typically just invokes **btldinstall**(ADM):

```
exec /etc/btldinstall "$2"
```

btldinstall asks which packages on this diskette are to be installed, and then adds the appropriate drivers from the requested packages to the system's Link Kit (*/etc/conf/...*).

/pkg/install/btld

If this file exists, then both **boot** and **btldinstall** assume */pkg* is a **boot**-time-loadable package hierarchy. This file contains, one per line, the pathnames of the directories containing each driver which **boot** is to link-edit into the kernel.

If this file does not exist, both **boot** and **btldinstall** will ignore the entire */pkg* hierarchy.

Each driver must have its own directory. The **basename**(C) of each such directory must be the same as the "internal name" (***xnamex***) of the driver as defined by column 1 of *mdevice*(F). Conventionally, each driver's directory is named */pkg/driver/xnamex*, and that is what is specified in the *btld* file. However, any pathname ending in the driver's internal name ***xnamex*** is acceptable.

/pkg/install/drivers

If this file exists, then it contains, one per line, the pathnames of the directories containing each driver which **btldinstall** is to add to the system's Link Kit. The basename of each listed directory must be the same as the internal name of the driver. The directories listed in this file do not have to be the same as those listed in */pkg/install/btld*.

If this file is missing, **btldinstall** does not install any drivers from this package into the Link Kit.

/pkg/install/pkg.name
An optional one-line description of this package.

This file, if it exists, will be installed by **btldinstall** as */usr/options/**pkg.name***. Both **btldinstall** and **displaypkg**(ADM) list the contents of this file.

/pkg/install/copyright
An optional Bourne shell script run by **btldinstall** to print out copyright and licensing information for this package. If this script exists, it must have execute permission.

/pkg/install/preinstall
An optional Bourne shell script run by **btldinstall** prior to installing the drivers listed in */**pkg**/install/drivers*. This script might be used to check the system version, or to check for the presence of optional software packages.

/pkg/install/postinstall
An optional Bourne shell script run by **btldinstall** after installing both the drivers listed in */**pkg**/install/drivers* and the */**pkg**/new* hierarchy. Typically, this script runs **fixperm**(ADM) to adjust the ownership and permissions of the installed files.

/pkg/install/bootstring
If this file exists, the first non-empty line not starting with an asterisk (*) or hash (#) is appended to the **bootstring** passed by **boot** to the booted UNIX kernel.

For example, **bootstring** might contain:

```
hd=xnamex
```

to specify that **boot**-linked driver *xnamex* (part of this package) is the primary hard disk.

/pkg/new/...
An optional hierarchy installed by **btldinstall** as if *new* were /. The owners, permissions, and contents of all files and directories in this hierarchy are copied. This is typically used to install a **mkdev**(ADM) script in */usr/lib/mkdev*, a **fixperm**-list in */etc/perms*, plus other assorted commands and data files specific to this package as required. However, drivers should not be added to the Link Kit (*/etc/conf/...*) in this manner.

.../xnamex
Directory containing the files specific to the driver whose internal name is ***xnamex***. Conventionally, this directory is */**pkg**/driver/**xnamex***, but the */**pkg**/install/btld* and */**pkg**/install/drivers* files may specify otherwise. This directory should not be in the */**pkg**/new* hierarchy.

.../xnamex/Master
File containing the *mdevice*(F) entries for driver ***xnamex***.

If this "driver" is listed in */pkg/install/drivers*, then this file must exist. The lines dealing with driver ***xnamex*** will be extracted and added to the system's *mdevice* file by **idinstall**(ADM) when run by **btldinstall.**

If this "driver" is listed in */pkg/install/btld* and really is a device driver (that is, if it must have entries added to the various kernel dispatch tables such as *bdevsw*), then this file should exist. For the first ***xnamex*** entry only, the **boot** link-editor adds appropriate entries to the indicated dispatch tables. Only some characteristics (*mdevice* column 3) and function tables (*mdevice* column 2) are recognized:

Characteristic	Description
b	Block device driver
c	Character device driver
S	STREAMS device driver
G	Check but do not install the interrupt handler
t	The device is a tty (character devices only)
C	Scatter/gather (block devices only)

Function	Description
o	**open** routine (device drivers only)
c	**close** routine (device drivers only)
r	**read** routine (character devices only)
w	**write** routine (character devices only)
i	**ioctl** routine (character devices only)
I	**init** routine
P	**pminit** routine
s	**start** routine
p	**poll** routine
h	**halt** routine
E	**kenter** routine
X	**kexit** routine
S	**swtch** routine

Block devices are assumed to always have **open, close, strategy,** and **print** routines.

When the kernel is built, extra space is left in the appropriate tables by **idconfig** according to the *mtune*(F) parameters:

Function or Characteristic	Parameter	Description
b	**MAX_BDEV**	At least this many block device entries
c	**MAX_CDEV**	At least this many character device entries
I P s p h E X S	**EXTRA_NDEV**	Extra unoccupied entries available for **boot** to fill

.../xnamex/System

An *sdevice*(F) file for driver ***xnamex***.

If this "driver" is listed in */pkg/install/drivers*, then this file must exist. It will be installed as */etc/conf/sdevice.d/**xnamex*** by **idinstall**.

If this "driver" is listed in */pkg/install/btld* and uses interrupts (or tunable parameters), then this file should exist. For the first entry only (which must be for ***xnamex***), the **boot** link-editor adjusts the interrupt dispatch tables accordingly.

.../xnamex/Bootload

Optional file used by **boot** to guide the link-editing. This file, if it exists, contains a series of directives. Each directive is one of the following keywords on a line, followed by additional lines specific to that keyword. Empty lines and lines beginning with asterisk (*) or number sign (#) are ignored.

The keywords include:

files Subsequent lines list the names of COFF object modules **boot** is to link-edit into the UNIX System kernel. The default is:

```
files
Driver.o
Space.o
```

The filenames are relative to the ***.../xnamex*** directory.

tune Subsequent lines list the *sdevice*(F) ***field*** for which the user is prompted, and the name and ***size*** of the initialized data (not BSS or text) ***symbol*** whose value is to be patched to be the user's answer:

field symbol size

Only 2 and 4 byte ***sizes*** are understood. The known ***field*** names include:

Field	mdevice/sdevice column	Legal range (inclusive)	Description
units	*sdevice* 3	† *decimal*	Number of peripherals connected to controller
vector	*sdevice* 6	1-255 *decimal*	Interrupt vector number *
DMAchan	*mdevice* 9	0-32767 *decimal*	DMA channel
SIOA	*sdevice* 7	1-0x3FF *hexadecimal*	Start I/O Address
EIOA	*sdevice* 8	SIOA-0x3FF *hexadecimal*	End I/O Address
SCMA	*sdevice* 9	0xA0000-0xFBFFF *hexadecimal*	Start Controller Memory Address
ECMA	*sdevice* 10	SCMA-0xFBFFF *hexadecimal*	End Controller Memory Address

† The minimum and maximum number of units are specified by columns 7 and 8 (respectively) of *mdevice*(F.)

* Most architectures only use the first 16 or so interrupt vectors. Interrupt vector 0 is always reserved for the system's clock. The tuned vector overrides *sdevice*(F) column 6, and so is used by **boot**.

As an example, the directive:

```
tune
SIOA xx_iobase 2
SCMA xx_ramloc 4
```

would cause the user to be prompted for the starting I/O address (SIOA) and starting controller memory address (SCMA). The answers supplied would be used to patch the initialized data:

```
short xx_iobase = 0x302;
long  xx_ramloc = 0xC8000;
```

The default answers (provided they are in range) are those specified in the appropriate *mdevice* (*Master*) or *sdevice* (*System*) column. The C-code initialized values are not used and are always overwritten.

patch Subsequent lines of the form:

symbol size value

cause the first ***size*** bytes of the kernel's definition of ***symbol*** to be set to ***value***. Only 2 and 4 byte ***size***s are understood; ***value*** is a (optionally signed "-" or "+") hexadecimal (**0x**), octal (**0**), or decimal integer. Only initialized data — not BSS or text — should be patched.

.../xnamex/Driver.o
The driver's relocatable COFF object module.

If this driver is listed in */**pkg**/install/drivers*, (that is, if it is installed by **btldinstall** into the Link Kit), then this file must exist. It is installed in the Link Kit as */etc/conf/pack.d/**xnamex**/Driver.o* by **idinstall**.

If this driver is listed in */**pkg**/install/btld*, and there is not a **files** directive in the *Bootload* file (to specify otherwise), then this file should exist.

.../xnamex/Space.o
An additional, configuration-dependent, relocatable COFF object module.

If this driver is listed in */**pkg**/install/drivers* (that is, if it is installed by **btldinstall** into the Link Kit), then this is installed in the Link Kit as */etc/conf/pack.d/**xnamex**/space.o* (note the change in capitalization) by **idinstall**.

If this driver is listed in */**pkg**/install/btld* and there is not a **files** directive in the *Bootload* file (to specify otherwise), then this file should exist.

.../xnamex/Space.c
C source to *Space.o*.

If this driver is listed in */**pkg**/install/drivers* and this file exists, then it is installed in the Link Kit as */etc/conf/pack.d/**xnamex**/space.c* (note the change in capitalization) by **idinstall**. This file is compiled and linked into the kernel along with any *Driver.o* whenever this driver is configured into the kernel being built.

.../xnamex/Stubs.c
If this C source exists, it is installed in the Link Kit as */etc/conf/pack.d/**xnamex**/stubs.c* (note the change in capitalization) by **idinstall**. This file is compiled and linked into the kernel whenever this driver is not configured into the kernel being built.

.../xnamex/Node
Description of the */dev* special files associated with this driver.

If this driver is listed in */**pkg**/install/drivers* and this file exists, it is installed in the Link Kit as */etc/conf/node.d/**xnamex*** by **idinstall**, and used by **idmknod**(ADM) when run from **idbuild**. The UNIX system installation also uses the *Node* file to determine which */dev* special files must exist: if package ***pkg*** is listed in the packages string (*/dev/string/cfg*) then for each driver listed in */**pkg**/install/btld* that has a *Node* file, the indicated */dev* special files are created.

.../xnamex/Inittab
Lines to add to */etc/inittab* to run various system startup and shutdown commands associated with this driver.

If this driver is listed in */pkg/install/drivers* and this file exists, it is installed in the Link Kit as */etc/conf/init.d/**xnamex*** by **idinstall**, and used by **idmkinit**(ADM) when run from **idbuild**.

.../xnamex/Rc
Bourne shell script run by */etc/rc2* when entering **init**(M) state 2 (multiuser operation).

If this driver is listed in */pkg/install/drivers* and this file exists, it is installed in the Link Kit as */etc/conf/rc.d/**xnamex*** by **idinstall**, and used by **idmkinit** when run from **idbuild**.

.../xnamex/Shutdown
Bourne shell script run by **/etc/rc0** when entering **init** state 0 (system shutdown).

If this driver is listed in */pkg/install/drivers* and this file exists, it is installed in the Link Kit as */etc/conf/sd.d/**xnamex*** by **idinstall**, and used by **idmkinit** when run from **idbuild**.

See also

basename(C), **boot**(HW), **btldinstall**(ADM), **configure**(ADM), **displaypkg**(ADM), **filesystem**(F), **fixperm**(ADM), **hd**(HW), **idbuild**(ADM), **idinstall**(ADM), **idmkinit**(ADM), **idmknod**(ADM), **idtune**(ADM), **init**(M), **installpkg**(ADM), **ld**(CP), **mdevice**(F), **mkdev**(ADM), **mtune**(F), **sdevice**(F), **sh**(C), **string**(M), **stune**(F)

Device Driver Writer's Guide

Notes

Lines in the *btld*, *bootstring*, *Bootload*, *Master*, and *System* files should be less then 60 characters in length.

The following *mdevice*(F) characteristics are silently ignored:

Characteristic	Description
i	Driver can be installed (assumed)
o	Only one *sdevice*(F) entry (**boot**(HW) only processes the first entry)
H	Driver controls hardware (not relevant)

Other *mdevice* characteristics always cause the **boot**-linking to fail:

Characteristic	Description
a	Driver is automatically installed
n	Driver is not installable
I	Ignore driver's *pack.d* directory
N	No *Driver.o* or *Space.c* files

Unrecognized characteristics and function tables cause a warning but are then ignored.

If the hardware controlled by the **boot**-linked driver has jumpers or switches for setting parameters such as the IRQ (interrupt vector), base I/O address, or memory address, these should be specified as tuneable parameters in the *Bootload* file. Users can then configure the hardware for their machine and add the required driver to the kernel so that it will use that configuration. This avoids the requirement for a specific configuration during system installation.

STREAMS modules, filesystems, event drivers, line disciplines and so forth cannot be **boot**-linked. Some versions of **boot** have facilities for linking some kernel debuggers; this is not supported and may change.

boot cannot check for I/O or Controller Memory Address conflicts with other devices. Not all interrupt vector or device major number conflicts can be resolved; **boot** presents the possible resolutions and suggests the one most likely to work (if any are liable to work).

The **-p** option to **idbuild** can be used to build *space.o* from *space.c* and thus obtain a *Space.o* (and *Space.c*) when making a BTLD diskette.

Some relocatable COFF object modules that **idbuild** can add to a kernel either cannot be added by **boot**, or are added in a slightly different manner. Such modules include those with multiple *mdevice* or *sdevice* entries, those with entries in function tables not patched by **boot**, or those with references to functions or data defined in other drivers. In general, it is inadvisable to **boot**-link a driver if that driver calls any external routine not defined in section K of the *Device Driver Writer's Guide* or if that driver expects to have an entry installed in any table not listed above.

checklist

list of file systems processed by fsck

Description

The */etc/checklist* file contains a list of the file systems to be checked when **fsck**(ADM) is invoked without arguments. The list contains a maximum of 15 *special file* names. Each *special file* name must be on a separate line and must correspond to a filesystem.

See also

fsck(ADM)

clock

the system real-time (time of day) clock

Description

The **clock** file provides access to the battery-powered, real-time time of day clock. Reading this file returns the current time; writing to the file sets the current time. The time, 10 bytes long, has the following form:

MMddhhmmyy

where ***MM*** is the month, ***dd*** is the day, ***hh*** is the hour, ***mm*** is the minute, and ***yy*** is the last two digits of the year. For example, the time:

0826150392

is 15:03 on August 26, 1992.

File

/dev/clock

See also

setclock(ADM)

Note

Not all computers have battery-powered real-time time of day clocks. Refer to your computer's hardware reference manual.

compver

compatible versions file

Description

compver is an ASCII file used to specify previous versions of the associated package which are upward compatible. It is created by a package developer.

Each line of the file specifies a previous version of the associated package with which the current version is backward compatible.

Since some packages may require installation of a specific version of another software package, compatibility information is crucial. For example, a package called "A" requires version "1.0" of application "B" as a prerequisite for installation. If the customer installing "A" has a newer version of "B" (version 1.3), the **compver** file for "B" must indicate that "1.3" is compatible with version "1.0" in order for the customer to install package "A".

Note

The comparison of the version string disregards white space and tabs. It is performed on a word-by-word basis. Thus:

"Version 1.3" and "Version 1.3"

would be considered the same.

Example

A sample **compver** file is shown below.

```
Version 1.3
Version 1.0
```

copyright

copyright information file

Description

copyright is an ASCII file used to provide a copyright notice for a package. The text can be in any format. The full file contents (including comment lines) are displayed on the terminal at the time of package installation.

cpio

format of cpio archive

Description

The *header* structure, when the **-c** option of **cpio**(C) is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        ushort  h_ino,
                h_mode,
                h_uid,
                h_gid;
        short   h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to word]; } Hdr;
```

When the **-c** option is used, the *header* information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
&Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
&Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
&Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

`Longtime` and `Longfile` are equivalent to `Hdr.h_mtime` and `Hdr.h_filesize`, respectively. The contents of each file are recorded in an element of the array of varying length structures, archive, together with other items describing the file. Every instance of `h_magic` contains the constant 070707 (octal). The items `h_dev` through `h_mtime` have meanings explained in **stat**(S). The length of the null-terminated path name `h_name`, including the null byte, is given by `h_namesize`.

The last record of the archive always contains the name "TRAILER!!!". Special files, directories, and the trailer are recorded with `h_filesize` equal to zero.

See also

cpio(C), **find**(C), **stat**(S)

Standards conformance

cpio is conformant with:

AT&T SVID Issue 2;
and X/Open Portability Guide, Issue 3, 1989.

default

default program information directory

Description

The files in the directory */etc/default* contain the default information used by system commands such as **xbackup**(ADM) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches a file whenever it has been invoked without sufficient information. Each file contains zero or more entries which define the default information. Each entry has the form:

keyword

or

keyword=value

where ***keyword*** identifies the type of information available and ***value*** defines its value. Both ***keyword*** and ***value*** must consist of letters, digits, and punctuation. The exact spelling of a keyword and the appropriate values depend on the command and are described with the individual commands.

Any line in a file beginning with a number sign "#" is considered a comment and is ignored.

File

*/etc/default/**

See also

archive(F), **authsh**(ADM), **boot**(HW), **cleantmp**(ADM), **cron**(C), **dos**(C), **filesys**(F), **format**(C), **goodpw**(ADM), **idleout**(ADM), **lock**(C), **login**(M), **lp**(C), **man**(C), **mapchan**(F), **mapchan**(M), **mapkey**(M), **passwd**(C), **purge**(C), **remote**(C), **su**(C), **tape**(C), **tar**(C), **usemouse**(C), **xbackup**(ADM), **xdumpdir**(ADM), **xrestore**(ADM)

Note

Not all commands use */etc/default* files. Please refer to the manual page for a specific command to determine if */etc/default* files are used, and what information is specified.

Value added

default is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

depend

software dependencies files

Description

depend is an ASCII file used to specify information concerning software dependencies for a particular package. The file is created by a software developer.

Each entry in the *depend* file describes a single software package. The instance of the package is described after the entry line by giving the package architecture and/or version. The format of each entry and subsequent instance definition is:

```
type pkg name
        (arch)version
        (arch)version
        ...
```

The fields are:

type Defines the dependency type. Must be one of the following characters:

- P Indicates a prerequisite for installation, for example, the referenced package or versions must be installed.
- I Implies that the existence of the indicated package or version is incompatible.
- R Indicates a reverse dependency. Instead of defining the package's own dependencies, this designates that another package depends on this one. This type should be used only when an old package does not have a *depend* file but it relies on the newer package nonetheless. Therefore, the present package should not be removed if the designated old package is still on the system since, if it is removed, the old package will no longer work.

pkg Indicates the package abbreviation.

name Specifies the full package name.

(arch)version Specifies a particular instance of the software. A version name cannot begin with a left parenthesis. The instance specifications, both ***arch*** and ***version***, are completely optional but must each begin on a new line that begins with white space. A null version set equates to any version of the indicated package.

Example

Here is a sample *depend* file:

```
I msvr Messaging Server
P ctc Cartridge Tape Utilities
P dfm Directory and File Management Utilities
P ed Editing Utilities
P ipc Inter-Process Communication Utilities
P lp Line Printer Spooling Utilities
P shell Shell Programming Utilities
P sys System Header Files
                Release 3.0
P sysadm System Administration Utilities
P term Terminal Filters Utilities
P terminfo Terminal Information Utilities
P usrenv User Environment Utilities
P uucp Basic Networking Utilities
P x25 X.25 Network Interface
                (i386) Issue 1 Version 1
                (i386) Issue 1 Version 2
R cms Call Management System
```

devices

format of UUCP devices file

Description

The *Devices* file (*/usr/lib/uucp/Devices*) contains information for all the devices that can be used to establish a link to a remote computer. These devices include automatic call units, direct links, and network connections. This file works closely with the *Dialers*, *Systems*, and *Dialcodes* files.

Each entry in the *Devices* file has the following format:

type ttyline dialerline speed dialer-token

where:

type can contain one of two keywords (*direct* or *ACU*), the name of a Local Area Network switch, or a system name.

ttyline contains the device name of the line (port) associated with the *Devices* entry. For example, if the Automatic Dial Modem for a particular entry is attached to the */dev/tty11* line, the name entered in this field is *tty11*.

dialerline is useful only for 801 type dialers, which do not contain a modem and must use an additional line. If you do not have an 801 dialer, enter a hyphen (-) as a placeholder.

speed is the speed or speed range of the device. It may contain an indicator for distinguishing different dialer classes.

dialer-token contains pairs of dialers and tokens. Each represents a dialer and an argument to be passed to it. The ***dialer*** portion can be the name of an automatic dial modem, or it may be a *direct* for a direct link device.

For best results, dialer programs are preferred over *Dialers* entries. The following entry is an example of an entry using a dialer binary:

```
ACU  tty***nn*** -  300-2400  /usr/lib/uucp/dialHA24
```

Note that all lines must have at least 5 fields. Use "-" for unused fields. Types that appear in the 5th field must be either built-in functions (801, Sytek, TCP, Unetserver, DK) or standard functions whose name appears in the first field in the *Dialers* file.

Two escape characters can be used in this file:

\D do not translate the phone /token

\T translate the phone /token using the *Dialcodes* file

Both refer to the phone number field in the *Systems* file (field 5). \D should always be used with entries in the *Dialers* file, since the *Dialers* file can contain a \T to expand the number if necessary. \T should only be used with built-in functions that require expansion.

Note that if a phone number is expected and a \D or \T is not present a \T is used for a built-in function, and \D is used for an entry referencing the *Dialers* file.

Examples

The following are examples of common *Devices* files.

Standard modem line

```
ACU tty00 - 1200 801
ACU tty00 - 1200 penril
or
ACU tty00 - 1200 penril \D
```

A direct line

This example will allow **cu -ltty00** to work. This entry could also be used for certain modems in manual mode.

```
Direct tty00 - 4800 direct
```

A ventel modem on a develcon switch

"vent" is the token given to the develcon to reach the ventel modem.

```
ACU tty00 - 1200 develcon vent ventel
ACU tty00 - 1200 develcon vent ventel \D
```

To reach a system on the local develcon switch

```
Develcon tty00 - Any develcon \D
```

A direct connection to a system

```
systemx tty00 - Any direct
```

Streams network examples

A Streams network that conforms to the AT&T Transport Interface with a direct connection to login service (that is, without explicitly using the Network Listener Service dial script):

networkx , eg ***devicex*** - - TLIS \D

The *Systems* file entry looks like:

systemx Any ***networkx*** - ***addressx*** in:--in: nuucp word: nuucp

You must replace ***systemx***, ***networkx***, ***addressx***, and ***devicex*** with system name, network name, network address and network device, respectively. For example, entries for machine *sffo* on a STARLAN NETWORK might look like:

```
sffo Any STARLAN - sffo in:--in: nuucp word: nuucp
```

and:

```
STARLAN,eg starlan - - TLIS \D
```

To use a Streams network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

networkx,`eg` ***devicex*** `- - TLIS \D nls`

To use a non-Streams network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

networkx,`eg` ***devicex*** `- - TLI \D nls`

See also

dialers(F), **uucico**(ADM), **uucp**(C), **uux**(C), **uuxqt**(C)

Note

Blank lines and lines that begin with a Space or Tab are ignored. Protocols can be specified as a comma-subfield of the device type either in the *Devices* file (where device type is field 1) or in the *Systems* file (where it is field 3).

dialcodes

format of UUCP Dialcode abbreviations file

Description

The *Dialcodes* file (*/usr/lib/uucp/Dialcodes*) contains the Dialcode abbreviations that can be used in the "Phone" field of the *Systems* file. This feature allows you to create a standard *Systems* file for distribution among several sites that have different phone systems and area codes.

If two remote sites in a network need to link with the same sites, but have different internal phone systems, each site can share the same *Systems* file, but have different entries in a *Dialcodes* file. Each entry has the following format:

abb dial-seq

where:

abb is the abbreviation used in the *Systems* file "phone" field

dial-seq is the dial sequence that is passed to the dialer when that particular *Systems* file entry is accessed.

The following entry would be set up to work with a "phone" field in the *Systems* file such as **jt7867**:

```
jt 9=847-
```

When the entry containing **jt7867** is encountered, the following sequence is sent to the dialer if the token in the dialer-token pair is *\T*:

```
9=847-7867
```

The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the *Dialcodes* file.

```
NY 9=1212555
```

See also

systems(F), **uucico**(ADM), **uucp**(C), **uux**(C), **uuxqt**(C)

dialers

format of UUCP Dialers file

Description

The *Dialers* file (*/usr/lib/uucp/Dialers*) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

A modem that is used for dialing in and out may require a second *Dialers* entry. This is to reinitialize the line to dial-in after it has been used for dial-out. The name of the dial-in version of a dialer must begin with an ampersand. For example, the *Dialers* file contains a *hayes2400* and a *&hayes2400* entry.

The fifth field in a *Devices* file entry is an index into the *Dialers* file or a special dialer type. Here an attempt is made to match the fifth field in the *Devices* file with the first field of each *Dialers* file entry. In addition, each odd numbered *Devices* field starting with the seventh position is used as an index into the *Dialers* file. If the match succeeds, the *Dialers* entry is interpreted to perform the dialer negotiations. Each entry in the *Dialers* file has the following format:

dialer substitutions expect-send ...

The ***dialer*** field matches the fifth and additional odd numbered fields in the *Devices* file. The ***substitutions*** field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate "=" and "-" into whatever the dialer requires for "wait for dialtone" and "pause."

The remaining ***expect-send*** fields are character strings. Below are some character strings distributed with the UUCP package in the *Dialers* file.

```
penril  =W-P "" \d > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
ventel  =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
hayes   =,-, "" \dAT\r\c OK\r \EATDT\T\r\c CONNECT
rixon   =&-% "" \d\r\r\c $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
vadiac  =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
develcon        "" "" \pr\ps\c est:\007 \E\D\e \007
micom   ""    "" \s\c NAME? \D\r\c GO
direct
att2212c        =+-,    "" \r\c :--: ato12=y,T\T\r\c red
att4000 =,-,    "" \033\r\r\c DEM: \033s0401\c \006 \033s0901\c \
          \006 \033s1001\c \006 \033s1102\c \006 \033dT\T\r\c \006
att2224 =+-,    "" \r\c :--: T\T\r\c red
nls         ""      "" NLPS:000:001:1\N\c
```

The meaning of some of the escape characters (those beginning with "\") used in the *Dialers* file are listed below:

\p	pause (approximately ¼ to ½ second)
\d	delay (approximately 2 seconds)
\D	phone number or token without *Dialcodes* translation
\T	phone number or token with *Dialcodes* translation
\K	insert a BREAK
\E	enable echo checking (for slow devices)
\e	disable echo checking
\r	carriage return
\c	no new-line or carriage return
\n	send new-line
\nnn	send octal number.

Additional escape characters that may be used are listed in the section discussing the *Systems* file.

The penril entry in the *Dialers* file is executed as follows. First, the phone number argument is translated, replacing any "=" with a "W" (wait for dialtone) and replacing any "-" with a "P" (pause). The handshake given by the remainder of the line works as follows:

""	Wait for nothing.
\d	Delay for 2 seconds.
>	Wait for a ">".
s\p9\c	Send an "s", pause for ½ second, send a "9", send no terminating new-line
)-W\p\r\ds\p9\c-)	Wait for a ")". If it is not received, process the string between the "-" characters as follows: send a "W", pause, send a carriage-return, delay, send an "s", pause, send a **9**, without a new-line, and then wait for the ")".
y\c	Send a "y".
:	Wait for a ":".
\E\TP	Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the *Dialcodes* translation and the modem function translation specified by field 2 of this entry. Then send a "P".
>	Wait for a ">".
9\c	Send a "9" without a new-line.
OK	Waiting for the string **OK**.

See also

devices(F), **dial**(ADM), **uucico**(ADM), **uucp**(C), **uux**(C), **uuxqt**(C)

Note

Dialer binaries (located in */usr/lib/uucp*) are preferred over *Dialers* entries. Binaries are more reliable. Refer to the **dial** man page for more information on creating your own dialer binaries.

filesys

default information for mounting filesystems

Description

/etc/default/filesys contains information for mounting filesystems in the following format:

name=value [***name=value***] ...

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

mnt (see **mnt**(C)) and **sysadmsh**(ADM) use the information in the */etc/default/filesys* when the system comes up as multi-user.

Options

The following options can be defined in the */etc/default/filesys* entry for a filesystem:

bdev=*/dev/device*	Name of block device associated with the filesystem. If ***fstyp*=NFS** the device should be of the form: **bdev=*hostname:pathname*.**
cdev=/dev/*device*	Name of character (raw) device associated with the filesystem.
mountdir=*/directory*	The directory the filesystem is to be mounted on.
desc=*name*	A string describing the filesystem.
passwd=*string*	An optional password prompted for at mount request time. Cannot be a simple string; must be in the format permitted by */etc/passwd*. (See "Notes".)
fsck=yes, no, dirty, prompt	If **yes/no**, tells explicitly whether or not to run **fsck**. If **dirty**, **fsck** is run only if the filesystem requires cleaning. If **prompt**, the user is prompted for a choice. If no entry is given, the default value is **dirty**.
fsckflags=*flags*	Any flags to be passed to **fsck**.
rcfsck=yes, no, dirty, prompt	Similar to **fsck** entry, but only applies when the **-r** flag is passed.
maxcleans=*n*	The number of times to repeat cleaning of a **dirty** filesystem before giving up. If undefined, default is 4.
mount=yes, no, prompt	If **yes** or **no**, users are allowed or disallowed to mount the filesystem, respectively. If **prompt**, the user specifies whether the filesystem should be mounted.

rcmount=yes, no, prompt	If **yes**, the filesystem is mounted by */etc/rc2* when the system comes up as multiuser. If **no**, the filesystem is never mounted by */etc/rc2*. With **prompt**, a query is displayed at boot time to mount the filesystem.
mountflags=*flags*	Any flags to be passed to **mount**.
prep=yes, no, prompt	Indicates whether any **prepcmd** entry should always be executed, never executed, or executed as specified by the user.
prepcmd=*command*	An arbitrary shell command to be invoked immediately following password check and prior to running **fsck**.
init=yes, no, prompt	Indicates whether an **initcmd** entry should always be executed, never be executed, or executed as specified by the user.
initcmd=*command*	An optional, arbitrary shell command to be invoked immediately following a successful mount.
fstyp=*type*	Defines the filesystem type. Available types include NFS, S51K, XENIX, and DOS.
nfsopts=*opts*	Defines NFS options for filesystems of type NFS. Available options are described in the **mount**(ADM) manual page.

Any entries containing spaces, tabs, or new lines must be contained in double quotes (").

The only mandatory entries in */etc/default/filesys* are **bdev** and **mountdir**. The **prepcmd** and **initcmd** options can be used to execute another command before or after mounting the filesystem. For example, **initcmd** could be defined to send mail to root whenever a given filesystem is mounted.

When invoked without arguments, **mnt** attempts to mount all filesystems that have the entries *mount=yes* or *mount=prompt*.

Note

The NFS options are only valid if NFS is installed; refer to your NFS documentation for **mount** options that are specific to NFS.

See also

mount(ADM), **mnt**(C), **sysadmsh**(ADM)

fspec

format specification in text files

Description

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs, (that is, tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of ***tabs*** must be one of the following:

- a list of column numbers separated by commas, indicating tabs set at the specified columns
- a **-** (dash) followed by an integer ***n***, indicating tabs at intervals of ***n*** columns
- a **-** (dash) followed by the name of a "canned" tab specification

Standard tabs are specified by **t-8**, or equivalently, **t1,9,17,25,** and so on. The canned tabs that are recognized are defined by the **tabs**(C) command.

s*size* The **s** parameter specifies a maximum line size. The value of ***size*** must be an integer. Size checking is performed after the tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of ***margin*** must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, these defaults are assumed for the entire file.

The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

See also

ed(C), **newform**(C), **tabs**(C)

gettydefs

speed and terminal settings used by getty

Description

The */etc/gettydefs* file contains information used by **getty**(M) to set up the speed and terminal settings for a line. It supplies information on what the login prompt should look like. It also supplies the speed to try next if the user indicates that the current speed is not correct by typing a BREAK character.

Each entry in */etc/gettydefs* has the following format:

label# initial-flags # final-flags # login-prompt #next-label

Each entry must be followed by a carriage return and a blank line. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as ***\nnn***, where ***nnn*** is the octal value of the desired character. The various fields are:

label This is the string against which **getty**(M) tries to match its second argument. It is often the speed, such as 1200, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial **ioctl**(S) settings to which the terminal is to be set if a terminal type is not specified to **getty**(M). The flags that **getty**(M) understands are the same as the ones listed in */usr/include/sys/termio.h* (see **termio**(M)). Normally only the speed flag is required in the ***initial-flags***. **getty**(M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The ***initial-flag*** settings remain in effect until **getty**(M) executes **login**(M).

The */etc/gettydefs* entries for PC-scancode terminals use the flag **SCANCODE** to set the default mapping from scancodes to a character set. The **getty** utility understands the **SCANCODE** flag in */etc/gettydefs* entries and issues an **ioctl** request to set the **KB_ISSCANCODE | KB_XSCANCODE** flags on the tty.

final-flags Sets the same values as the ***initial-flags***.
These flags are set just prior to **getty** executing **login-program**. The speed flag is again required. The composite flag **SANE** is a composite flag that sets the following **termio**(M) parameters:

modes set:
CREAD BRKINT IGNPAR ISTRIP ICRNL IXON ISIG ICANON ECHO ECHOK OPOST ONLCR

modes cleared:
CLOCAL IGNBRK PARMRK INPCK INLCR IUCLC IXOFF XCASE ECHOE ECHONL NOFLSH OLCUC OCRNL ONOCR ONLRET OFILL OFDEL NLDLY CRDLY TABDLY BSDLY VTDLY FFDLY

The other two commonly specified ***final-flags*** are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt

Contains login prompt message that greets users. Unlike the above fields where white space is ignored (a space, tab, or new-line), it is included in the ***login-prompt*** field. The "@" in the login-prompt field is expanded to the first line (or second line if it exists) in */etc/systemid* (unless the "@" is preceded by a "\"). Several character sequences are recognized, including:

\n	Linefeed
\r	Carriage return
\v	Vertical tab
***nnn**	(3 octal digits) Specify ASCII character
\t	Tab
\f	Form feed
\b	Backspace

next-label Identifies the next entry in **gettydefs** for **getty** to try if the current one is not successful. **getty** tries the next label if a user presses the BREAK key while attempting to log in to the system. Groups of entries, for example, for dial-up lines or for TTY lines, should form a closed set so that **getty** cycles back to the original entry if none of the entries is successful. For instance, 2400 linked to 1200, which in turn is linked to 300, which is finally linked to 2400.

If **getty** is called without a second argument, then the first entry of */etc/gettydefs* is used as the default entry. The first entry is also used if **getty** cannot find the specified label. If */etc/gettydefs* itself is missing, there is one entry built into the command which will bring up a terminal at 300 baud.

After modifying */etc/gettydefs*, run it through **getty** with the check option to be sure there are no errors.

File

/etc/gettydefs

See also

stty(C), **ioctl**(S), **getty**(M), **login**(M), **inittab**(F,) **scancode**(HW)

gps

graphical primitive string, format of graphical files

Description

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices as have some higher level graphics programs such as **plot** (in **stat**(1G)) and **vtoc** (in **toc**(1G)) produce GPS format output files.

A GPS is composed of five types of graphical data or primitives:

GPS primitives

lines The **lines** primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a **move** to that location. (A **move** is a relocation of the graphic cursor without drawing.) Each successive point produces a line segment from the previous point. Parameters are available to set **color, weight,** and **style.**

arc The **arc** primitive has a variable number of points to which a curve is fitted. The first point produces a **move** to that point. If only two points are included, a line connecting the points will result; if there are three points, a circular arc is drawn through the points; and if there are more than three points, lines connect the points. (In the future, a spline will be fitted to the points if they number greater than three.) Parameters are available to set **color, weight,** and **style.**

text The **text** primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are **color, font, textsize,** and **textangle.**

hardware The **hardware** primitive draws hardware characters or gives control commands to a hardware device. A single point locates the start of the **hardware** string.

comment The **comment** primitive is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS parameters

color **color** is an integer value set for **arc, lines,** and **text** primitives.

weight **weight** is an integer value set for **arc** and **lines** primitives to indicate line thickness. The value **0** is narrow weight; **1** is bold; and **2** is medium weight.

style **style** is an integer value set for **lines** and **arc** primitives to give one of the five line styles that can be drawn on TEKTRONIX 4010 series storage tubes. They are:

0 solid
1 dotted
2 dot dashed
3 dashed
4 long dashed

font **font** is an integer value set for **text** primitives to designate the text font to be used in drawing a character string. (Currently **font** is expressed as a four-bit **weight** value followed by a four-bit **style** value.)

textsize **textsize** is an integer value used in **text** primitives to express the size of the characters to be drawn. **textsize** represents the height of characters in absolute **universe-units** and is stored at one-fifth this value in the size-orientation (**so**) word.

textangle **textangle** is a signed integer value used in **text** primitives to express rotation of the character string around the beginning point. **textangle** is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (**so**) word as a value 256/360 of its absolute value.

Organization

GPS primitives are organized internally as follows:

lines	*cw points sw*
arc	*cw points sw*
text	*cw point sw so* [*string*]
hardware	*cw point* [*string*]
comment	*cw* [*string*]

cw *cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word count for that primitive.

point(s) *point(s)* is one pair or more of integer coordinates. **text** and **hardware** primitives only require a single *point*. *point(s)* are values within a Cartesian plane or universe having 64K (-32K to +32K) points on each axis.

sw *sw* is the style word and is used in **lines**, **arc**, and **text** primitives. For all three, eight bits contain **color** information. In **arc** and **lines**, eight bits are divided as four bits **weight** and four bits **style**. In the **text** primitive, eight bits of *sw* contain the **font**.

so *so* is the size orientation word used in **text** primitives. Eight bits contain text size and eight bits contain text rotation.

string ***string*** is a null-terminated character string. If the string does not end on a word boundary, another null is added to the GPS file to insure word-boundary alignment.

See also

graphics(ADM), **stat**(ADM), **toc**(ADM)

group

format of the group file

Description

group contains the following information for each group:

Group name
Numerical group ID
Comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory */etc*. Because of the encrypted passwords, it has have general read permission and can be used, for example, to map numerical group IDs to names.

File

/etc/group

See also

newgrp(C), **passwd**(C), **passwd**(F)

Standards conformance

group is conformant with:

X/Open Portability Guide, Issue 3, 1989;
IEEE POSIX Std 1003.1-1990 System Application Program Interface (API) [C Language] (ISO/IEC 9945-1);
and NIST FIPS 151-1.

hs

High Sierra/ISO-9660 CD-ROM filesystem

Description

The hs filesystem module supports the mounting of CD-ROM filesystems conforming the High Sierra/ISO-9660 format.

Notes

The CD-ROM is a read-only device therefore it is only possible to mount CD-ROM filesystems as read-only. The kernel enforces this regardless of whether the **-r** option of **mount**(ADM) is used when the filesystem is mounted.

The command **mkdev high-sierra** is used to configure High Sierra/ISO-9660 filesystem support.

File

*/usr/include/sys/fs/hs**

See also

cdrom(HW), **mkdev**(ADM), **mount**(ADM)

inittab, init.base

script for the init process

Description

The *inittab* file supplies the script to **init**'s role as a general process dispatcher. The process that constitutes the majority of **init**'s process dispatching activities is the line process */etc/getty* that initiates individual terminal lines. Other processes typically dispatched by **init** are daemons and the shell.

The *inittab* file is recreated automatically by **idmkinit** at boot time anytime the kernel has been reconfigured. To construct a new *inittab* file, **idmkinit** concatenates the device driver init files in */etc/conf/init.d* onto the end of */etc/conf/cf.d/init.base* (the default *inittab*).

If you add an entry directly to *inittab*, the change exists only until the kernel is relinked. To add an entry permanently, you must also edit */etc/conf/cf.d/init.base*. The *init.base* file has the same format as *inittab*.

The *inittab* file is composed of entries that are position-dependent and have the following format:

id:rstate:action:process

Each entry is delimited by a new-line; however, a backslash " \ " preceding a new-line indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the "process" field using the **sh**(C) convention for comments. Comments for lines that spawn **getty**s are displayed by the **who**(C) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file. The entry fields are:

id This is up to four characters used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system. That is, each process spawned by **init** is assigned a run-level or run-levels in which it is allowed to exist. The run-levels are represented by a number ranging from *0* through *6*. As an example, if the system is in run-level *1*, only those entries having a *1* in the ***rstate*** field will be processed. When **init** is requested to change run-levels, all processes which do not have an entry in the ***rstate*** field for the target run-level will be sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The ***rstate*** field can define multiple run-levels for a process by selecting more than one run-level in any combination from *0-6*. If no run-level is specified, then the process is assumed to be valid at all run-levels *0-6*. There are three other values, *a*, *b*, and *c*, which can appear in the ***rstate*** field, even though

they are not true run-levels. Entries which have these characters in the ***rstate*** field are processed only when the **telinit** (see **init**(M)) process requests them to be run (regardless of the current run-level of the system). They differ from run-levels in that **init** can never enter run-level *a*, *b*, or *c*. Also, a request for the execution of any of these processes does not change the current run-level. In addition, a process started by an *a*, *b*, or *c* command is not killed when **init** changes levels. They are only killed if their line in */etc/inittab* is marked *off* in the ***action*** field, their line is deleted entirely from */etc/inittab*, or **init** goes into the single user state.

action Key words in this field tell **init** how to treat the process specified in the ***process*** field. The actions recognized by **init** are as follows:

respawn If the process does not exist, then start the process; do not wait for its termination (continue scanning the *inittab* file), and when it dies, restart the process. If the process currently exists, then do nothing and continue scanning the *inittab* file.

wait Upon **init**'s entering the run-level that matches the entry's ***rstate***, start the process and wait for its termination. All subsequent reads of the *inittab* file while **init** is in the same run-level will cause **init** to ignore this entry.

once Upon **init**'s entering a run-level that matches the entry's ***rstate***, start the process; do not wait for its termination. When it dies, do not restart the process. If upon entering a new run-level, where the process is still running from a previous run-level change, the program will not be restarted.

boot The entry is to be processed only at **init**'s boot-time read of the *inittab* file. **init** is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the ***rstate*** should be the default or it must match **init**'s run-level at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time **init** goes from single-user to multi-user state after the system is booted. (If "initdefault" is set to 2, the process will run right after the boot.) **init** starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when **init** receives a power fail signal (**SIGPWR** see **signal**(S)).

powerwait Execute the process associated with this entry only when **init** receives a power fail signal (**SIGPWR**) and wait until it terminates before continuing any processing of **inittab**.

off If the process associated with this entry is currently running, send the warning signal (**SIGTERM**) and wait 20 seconds before forcibly terminating the process via the kill signal (**SIGKILL**). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the "respawn" action. It is functionally identical to "respawn" but is given a different keyword in order to divorce its association with **run-levels**. This is used only with the *a*, *b*, or *c* values described in the ***rstate*** field.

initdefault An entry with this **action** is only scanned when **init** is initially invoked. **init** uses this entry, if it exists, to determine which **run-level** to enter initially. It does this by taking the highest **run-level** specified in the ***rstate*** field and using that as its initial state. If the ***rstate*** field is empty, this is interpreted as *0123456* and so **init** will enter **run-level** *6*. Additionally, if **init** does not find an "initdefault" entry in */etc/inittab*, then it will request an initial **run-level** from the user at reboot time.

sysinit Entries of this type are executed before **init** tries to access the console (that is, before the *Console Login:* prompt). It is expected that this entry will be used only to initialize devices on which **init** might try to ask the run-level question. These entries are executed and waited for before continuing.

process This is a **sh** command to be executed. The entire ***process*** field is passed to a forked **sh** to be run by **initscript**(F). For this reason, any legal **sh** syntax can appear in the ***process*** field. Comments can be inserted with the ***; #comment*** syntax.

Files

/etc/inittab
/etc/conf/cf.d/init.base

Note

Never modify both */etc/conf/init.d/sio* and */etc/conf/cf.d/init.base*, or duplicate *inittab* entries will result.

See also

disable(C), **enable**(C), **exec**(S), **getty**(M,) **idmkinit**(ADM), **init**(M), **initscript**(F), **open**(S), **sh**(C), **signal**(S), **sulogin**(ADM), **who**(C)

issue

issue identification file

Description

The file */etc/issue* contains the **issue** or project identification to be printed as a login prompt. This is an ASCII file which is read by the **getty** program and then written to any terminal spawned or respawned from the */etc/inittab* file.

File

/etc/issue

See also

login(M), **init**(M), **ct**(C), **inittab**(F)

logs

MMDF log files: system status, error, and statistics logging for MMDF

Description

MMDF maintains runtime log files at several levels of activity. The primary distinction is among message-level, channel-level, and link-level information. All logging settings can be overridden by entries in the runtime tailor file . In MMDF, that member is merged with */usr/mmdf/log* to determine the full pathname to the log. Logs are protected so that any process may write into them, but only MMDF may read them (that is, 0622).

The logging files may be the source of some confusion, since the log package entails some complexity. Its three critical factors are coordinated access, restricted file length, and restricted verbosity.

The length of a logging file can be limited to 25-block units. This is extremely important since files can grow very long over a period of time, especially if there are many long messages sent or very verbose logging.

Restricted verbosity is a way of easily tuning the amount of text entered into the log. This is probably the one parameter you need to be most concerned about. Set to full tilt (level=FTR), MMDF becomes noticeably slower and I/O bound. It also shows what it is doing, and helps you to discern the source of errors. When you are used to the code, setting the logging level down is highly recommended. The lowest would be TMP or FAT, for temporary or fatal errors. GEN will log errors and general information. FST logs errors, general and statistics information.

Specific logs

Even with the listed divisions, the logs contain a variety of information. Only the message-level log's format will be explained in significant detail.

msg.log — records enqueue and dequeue transitions, by **submit** and **deliver**. Entries by a background **deliver** process are noted with a "BG-xxxx" tag, where the x's contain the 4 least-significant decimal digits of the daemon's process id. This is to allow distinguishing different daemons. When **deliver** is invoked, by **submit**, for an immediate attempt, the tag begins with "DL" rather than "BG". Entries by **submit** begin with "SB".

Every major entry will indicate the name of the message involved. Entries from **submit** will show "lin" if the submission is from a user on the local machine. In this case, the end of the entry will show the login name of the sender. If the entry is labelled "rin," then the mail is being relayed. The channel name, source host, and sender address are shown. Within parentheses, the number of addressees and the byte-length of the message are listed.

Entries from **deliver** show final disposition of a message/addressee. These are indicated by "end." Then, there is the destination channel and mailbox name. In brackets, the queue latency for the address is shown in hours, seconds, and minutes.

chan.log — records activity by the channel programs, in *chndfldir[]*. Entries have a tag indicating the type of channel making the entry. Different channels record different sorts of information. For example, the local channel shows when a **rcvmail** private reception program is invoked.

See also

mmdf(ADM)

Value added

logs is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

maildelivery

user delivery specification file

Description

The delivery of mail by the local channel can run through various courses, including using a user tailorable file. The delivery follows the following strategy, giving up at any point it considers the message delivered.

1. If the address indicates a pipe or file default, then that is carried out.
2. The file *.maildelivery* in the home directory is read if it exists and the actions in it are followed.
3. If the message still hasn't been delivered, then it is put into the user's normal mailbox.

The format of the *.maildelivery* file is

field **<FS>** ***pattern*** **<FS>** ***action*** **<FS>** ***result*** **<FS>** ***string***

where:

field is name of a field that is to be searched for a pattern. This is any header field that you might find in a message. The most commonly used headers are: From, To, cc, Subject and Sender. As well as the standard headers, there are some pseudo-headers that can also be used. These are:

- **source** the out-of-band sender information. This is the address MMDF would use for reporting delivery problems with the message.
- **addr** the address that was used to mail to you, normally yourname or yourname=string (see below).
- **default** if the message hasn't been delivered yet, this field is matched.
- * this case is always true regardless of any other action.

pattern is some sequence of characters that may be matched in the above ***field***. Case is not significant.

action is one of the mail delivery actions supported by the local channel. Currently the supported actions are **file** or "**>**", which appends the message to the given file, with delimiters; **pipe** or "**|**", which starts up a process with the message as the standard input; and **destroy** which throws the message away.

For piped commands, the exit status of the command is significant. An exit status of 0 implies that the command succeeded and everything went well. An exit status of octal 0300-0377 indicates that a permanent failure occurred and the message should be rejected. Any other exit status indicates a temporary failure and the delivery attempt will be aborted and restarted at a later time.

result is one of the letters A, R or ? which stand for Accept, Reject and "Accept if not delivered yet". They have the following effects:

A If the result of this line's action is OK, then the message can be considered delivered.

R The message is not to be considered delivered by this action.

? This is equivalent to **A** except that the action is not carried out if the message has already been accepted.

The file is always read completely so that several matches can be made, and several actions taken. As a security check, the *.maildelivery* file must be owned by either the user or root, and must not have group or general write permission. In addition the system delivery file has the above restrictions but must also be owned by root. If the field specified does not need a pattern, a dash (-), or similar symbol is usually inserted to show that the field is present but not used. The field separator character can be a tab, space or comma (,). These characters can be included within a field by quoting them with double quotes ("); double quotes can be included preceded by a backslash (\).

MMDF treats local addresses which contain an equal sign (=) in a special manner. Everything in a local address from an equal sign to the "@" is ignored and passed on to the local channel. The local channel will make the entire string available for matching against the "addr" string of the file. For example, if you were to subscribe to a digest as "foo=digest@bar.NET", **submit**(ADM) and the local channel will verify that it is legal to deliver to "foo", but then the entire string "foo=digest" will be available for string matching against the MMDF file for the "addr" field.

Environment

The environment in which piped programs are run contains a few standard features, specifically:

HOME is set to the user's home directory.
USER is set to the user's login name.
SHELL is set to the user's login shell
(defaults to */bin/sh*).

The default umask is set to 077 (a very protective creation mask). A shell script can be run first to set up more complex environments.

There are certain built-in variables that you can give to a piped program. These are **$(sender)**, **$(address)**, **$(size)**, **$(reply-to)** and **$(info)**. **$(sender)** is set to the return address for the message. **$(address)** is set to the address that was used to mail to you, normally 'yourname' or 'yourname=string'. **$(size)** is set to the size in bytes of this message. **$(reply-to)** is set to the Reply-To: field (or the From: field if the former is missing) and so can be used for automatic replies. **$(info)** is the info field from the internal mail header and is probably only of interest to the system administrators.

Example

Here is a rough idea of what a *.maildelivery* file looks like:

```
# lines starting with a ''#'' are ignored.
# as are blank lines
# file mail with mmdf2 in the "To:" line into file mmdf2.log
To    mmdf2    file    A    mmdf2.log
# Messages from mmdf pipe to the program err-message-archive
From    mmdf    pipe    A    err-message-archive
# Anything with the "Sender:" address "uk-mmdf-workers"
# file in mmdf2.log if not filed already
Sender    uk-mmdf-workers    file    ?    mmdf2.log
# "To:" unix - put in file unix-news
To    Unix    >    A    unix-news
# if the address is jpo=mmdf - pipe into mmdf-redist
Addr    jpo=mmdf    |    A    mmdf-redist
# if the address is jpo=ack - send an acknowledgement copy back
Addr    jpo=ack    |    R    resend  -r  $(reply-to)
# anything from steve - destroy!
from    steve    destroy    A    -
# anything not matched yet - put into mailbox
default    -    >    ?    mailbox
# always run rcvalert
*    -    |    R    rcvalert
```

File

$HOME/.maildelivery normal location

See also

mmdftailor(F), **rcvalert**(C), **rcvfile**(C), **rcvprint**(C), **rcvtrip**(C)

Credit

MMDF was developed at the University of Delaware and is used with permission.

mapchan

format of tty device mapping files

Description

mapchan configures the mapping of information input and output.

Each unique **channel** map requires a multiple of 1024 bytes (a 1K buffer) for mapping the input and output of characters. No buffers are required if no **channels** are mapped. If control sequences are specified, an additional 1K buffer is required.

A method of sharing maps is implemented for **channels** that have the same map in place. Each additional, unique map allocates an additional buffer. The maximum number of map buffers available on a system is configured in the kernel, and is adjustable via the link kit **NEMAP** parameter (see **configure**(ADM)). Buffers of maps no longer in use are returned for use by other maps.

Example of a map file

The internal character set is defined by the right column of the input map, and the first column of the output map in place on that line. The default internal character set is the 8-bit ISO 8859/1 character set, which is also known as dpANS X3.4.2 and ISO/TC97/SC2. It supports the Latin alphabet and can represent most European languages.

Any character value not given is assumed to be a straight mapping: only the differences are shown in the ***mapfile***. The left-hand columns must be unique. More than one occurrence of any entry is an error. Right-hand column characters can appear more than once. This is "many to one" mapping. Nulls can be produced with compose sequences or as part of an output string.

It is recommended that no mapping be enabled on the **channel** used to create or modify the mapping files. This prevents any confusion of the actual values being entered due to mapping. It is also recommended that numeric rather than character representations be used in most cases, as these are not likely to be subject to mapping. Use comments to identify the characters represented. Refer to the **ascii**(M) manual page and the hardware reference manual for the device being mapped for the values to assign.

```
#
# sharp/pound/cross-hatch is the comment character
# however, a quoted # ('#') is 0x23, not a comment
#
# beep, input, output, dead, compose and
# control are special keywords and should appear as shown.
#
beep        # sound the bell when errors occur
input

a b
c d

dead p
q r         # p followed by q yields r.
s t         # p followed by s yields t.

dead u
v w         # u followed by v yields w.

compose x   # x is the compose key (only one allowed).
y z A       # x followed by y and z yields A.
B C D       # x followed by B and C yields D.

output
e f         # e is mapped to f.
g h i j     # g is mapped to hij - one to many.
k l m n o   # k is mapped to lmno.

control     # The control sections must be last

input
E  1        # The character E is followed by 1 more unmapped character

output
FG 2        # The characters FG are followed by 2 more unmapped characters
```

All of the single letters above preceding the "control" section must be in one of these formats:

```
56      # decimal
045     # octal
0xfa    # hexadecimal
'b'     # quoted char
'\076'  # quoted octal
'\x4a'  # quoted hex
```

All of the above formats are translated to single byte values.

The *control* sections (which must be the last in the file) contain specifications of character sequences which should be passed through to or from the terminal device without going through the normal **mapchan** processing. These specifications consist of two parts: a fixed sequence of one or more defined characters indicating the start of a no-map sequence, followed by a number of characters of which the actual values are unspecified.

To illustrate this, consider a cursor-control sequence which should be passed directly to the terminal without being mapped. Such a sequence would typically begin with a fixed escape sequence instructing the terminal to interpret the following two characters as a cursor position; the values of the following two characters are variable, and depend on the cursor position requested. Such a control sequence would be specified as:

```
\E=     2       # Cursor control: escape = <x> <y>
```

There are two subsections under *control*: the *input* section is used to filter data sent from the terminal to UNIX, and the *output* section is used to filter data sent from UNIX to the terminal. The two fields in each control sequence are separated by white space, that is the Space or Tab characters. Also the "#" (number sign) character introduces a comment, causing the remainder of the line to be ignored. Therefore, if any of these three characters are required in the specification itself, they should be entered using one of alternative means of entering characters, as follows:

^x	The character produced by the terminal on pressing the ⟨Ctrl⟩ and ⟨x⟩ keys together.
\E or \e	The ⟨Esc⟩ character, octal 033.
\c	Where *c* is one of b, f, l, n, r or t, produces Backspace, Formfeed, Linefeed, Newline, Carriage Return, or Tab characters respectively.
\0	Since the NULL character can not be represented, this sequence is stored as the character with octal value 0200, which behaves as a NULL on most terminals.
\nn or **\nnn**	Specifies the octal value of the character directly.
\	followed by any other character is interpreted as that character. This can be used to enter Space, Tab, or Hash characters.

Diagnostics

mapchan performs these error checks when processing the mapfile:

- More than one compose key.
- Characters mapped to more than one value.
- Syntax errors in the byte values.
- Missing input or output keywords.
- Dead or compose keys also occurring in the input section.
- Extra information on a line.
- Mapping a character to null.
- Starting an output control sequence with a character that is already mapped.

If characters are displayed as the 7-bit value instead of the 8-bit value, use **stty -a** to verify that **-istrip** is set. Make sure *input* is mapping to the 8859 character set, *output* is mapping from the 8859 to the device display character set. *dead* and *compose* sequences are *input* mapping and should be going to 8859.

Files

/etc/default/mapchan
*/usr/lib/mapchan/**

See also

ascii(M), **keyboard**(HW), **lp**(C), **lpadmin**(ADM), **mapchan**(M), **mapkey**(M), **parallel**(HW), **screen**(HW), **serial**(HW), **setkey**(C), **trchan**(M), **tty**(M)

Notes

Some non-U.S. keyboards and display devices do not support characters commonly used by UNIX command shells and the C programming language. Do not attempt to use such devices for system administration tasks.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

Warnings

Use of mapping files that specify a different "internal" character set per-channel, or a set other than the 8-bit ISO 8859 set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set (see **ascii**(M)). UNIX utilities and applications assume these values. Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. **trchan** can be used to "translate" from one internal character set to another.

Do not set **ISTRIP** (see **stty**(C)) on channels that have mapping that includes 8-bit characters.

Value added

mapchan is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

maxuuscheds

UUCP uusched(ADM) limit file

Description

The *Maxuuscheds* (*/usr/lib/uucp/Maxuuscheds*) file contains a numeric string to limit the number of simultaneous **uusched** programs running. Each **uusched** running will have one **uucico** associated with it; limiting the number will directly affect the load on the system. The limit should be less than the number of outgoing lines used by UUCP (a smaller number is often desirable). This file is delivered with a default entry of 2: this may be changed to meet the needs of the local system. However, keep in mind that the load on the system increases with the number of **uusched** programs running.

See also

uucico(ADM), **uucp**(C), **uusched**(ADM), **uux**(C), **uuxqt**(C)

maxuuxqts

UUCP uuxqt(C) limit file

Description

The *Maxuuxqts* (*/usr/lib/uucp/Maxuuxqts*) file contains an ASCII number to limit the number of simultaneous **uuxqt** programs running. This file has a default entry of 2. If there is a lot of traffic from **mail**, you can increase this number to reduce the time it takes for the mail to leave your system. Keep in mind that the load on the system increases with the number of **uuxqt** programs running.

See also

uucico(ADM), **uucp**(C), **uux**(C), **uuxqt**(ADM)

mcconfig

Irwin tape driver parameters

Description

mcdaemon - tape driver daemon

/etc/default/mcconfig contains information on Irwin tape driver parameters. **mcconfig** entries are in the following format:

variable=parameterlist

variable is a case insensitive character string that names a configuration parameter. ***parameterlist*** is a string of one or more parameter values, the format of which are dependent on the variable used.

The following variables are defined:

IROPT	driver options
IRDBG	debugging aids
SYSFDC	system floppy controller parameters
ALTFDC	alternate controller parameters
4100	Irwin 4100 PC bus controller parameters
4100B	second 4100 PC bus controller parameters
IRDRV	drive searching sequence (old method 2.00)
IRSRCH	drive searching sequence (new method 2.02)
4251	4151 address

When configuring parameters, space and tab characters cannot be used. For example,

irdrv=3	is correct, while
irdrv = 3	is incorrect and will be ignored.

Parameters are passed to the tape driver by the daemon program **/etc/mcdaemon**. Configuration parameters are given on separate lines. The number sign character (#) may be used to open a comment. Comments are terminated by a newline. For example, the *mcconfig* file might contain:

```
# this is a comment in the mcconfig file
iropt=F
4251=31f
```

Whenever changes are made to the *mcconfig* file, you need to reboot the system for the changes to take effect.

IROPT: configuration option string

The tape driver configuration variable **IROPT** may be used override certain default or automatically determined configuration parameters.

The values for **IROPT** are as follows:

B/b: 64K DMA Boundary Present/Absent

B This computer's hardware architecture has a 64K DMA memory boundary. Tape data transfer buffers may not cross a 64K physical boundary. This is the case for most PC and AT compatible machines.

b This computer's hardware architecture does not have a 64K DMA physical memory boundary. Tape data transfer buffers may be allocated anywhere in memory. This is true for PS/2s with the Micro Channel Architecture.

When neither "B" nor "b" is set, configuration is based upon the result of Micro Channel presence determination (see the M/m option). In a Micro Channel machine, "b" is assumed, otherwise "B" is used.

D/d: Use Demand/Single Byte DMA with Controllers Having a FIFO

D When running in PC or AT class machine an controller which has a first-in-first-out (FIFO) buffer, use demand mode DMA transfers. Both the Intel 82072 and 82077 floppy controller chips (the latter is used in the 4100PC) have a 16 byte FIFO.

d When running in a PC or AT class machine, use the standard single byte DMA transfer mode regardless of the floppy controller type.

When neither "D" nor "d" is set, automatic configuration determines whether a floppy controller chip with a FIFO is present on a per controller basis. When a controller having a FIFO is found (for example, Intel 82072/82077 parts return a positive response to the **CONFIGURE** command), DMA transfers with respect to that controller are set up using the demand mode. Using demand mode decreases the portion of the bus bandwidth consumed by tape read/write transfers.

F/f: Floating/Pulled-Up Drive Search

F When searching for drives on the system controller, use a special "floating track 0" drive search. The "floating" drive search assumes the track 0 floppy interface line floats (can be high or low) when no drive is attached. This algorithm works in all machines but cannot locate a drive which is executing a load-point operation. The floating search is required on certain Adaptec controllers.

f When searching for drives on the system controller, use the standard "pulled-up track 0" drive search. The standard algorithm assumes the floppy interface's track 0 line is pulled up (is high) when no tape drive is attached. When the standard search is employed on a controller which "floats" the track 0 line, a drive may be erroneously detected at a line where none is present. To deal with this condition either the **IRDRV** environment variable may be set to specify the drive line (preferred) or the "floating track 0" drive search "F" may be specified.

When neither "F" nor "f" is set, automatic configuration of this option is performed by examining the model information returned from the BIOS "Get Machine Configuration" service (int 15, AH = C0). The following model uses the "floating" drive search:

Model	Type	Sub-type PS/2 Model
F8	0D	24 MHz Model 70

H/h: Do/Don't Test for 4100 PC Bus Controller Signature

H Test for Irwin 4100 PC Bus controller (default).

h No 4100 PC controller present.

In the PC or AT (not Micro Channel) hardware environment (see the M/m option), when testing for the presence of a 4100 PC controller, the driver reads a byte from a signature port on the controller and compares this against the value 45 hexadecimal. The I/O port address of the signature port is found by adding six to the board's base port address (see the controller configuration section). For a 4100 PC Bus controller with switches set to "as shipped from the factory" positions, the signature port address is 0370 (hexadecimal) + 6. If the byte compares, the 4100 PC is present: otherwise it is not. This option is intended to be used when peeking at the factory set (0376 hexadecimal) signature port causes the disruption of some other adapter which is present at this address.

I/i: Do/Don't Wait-for-Index

I Wait-for-index before data transfer of each tape block.

i No need to wait-for-index before data transfer.

When neither "I" nor "i" is set, wait-for-index is enabled by default only when an Olivetti Micro Channel machine is present: otherwise wait-for-index is disabled.

If the following symptoms are experienced, after installing the MC driver in certain Micro Channel machines, the wait-for-index algorithm may need to be enabled:

- On the first backup this message is seen:

```
mc tape write error: Defect list has unrecoverable error
```

- If **tape format** gives the error:

```
Formatting failed: Block 0 medium error :
phase: CERTIFICATION,  track:  0,  cylinder:  0
```

- Extremely poor performance is experienced while listing the content of or restoring a previously written tape.

A condition exists in some Micro Channel computers which causes errors reading the first sector of each tape block. Included are the IBM models 50, 60, and 80, and the Olivetti P-500.

These machines employ 72065 (except for the Olivetti which has a 765) floppy controllers and data separators with certain characteristics. The 72065 differs from other controllers in that it does not inhibit **VCO SYNC** when an **INDEX** signal is received. Characteristically, the data separator circuit will:

1) have a phase lock loop (PLL) which totally looses synchronization when confronted with a 50/50 duty cycle read data signal; and

2) be slow to re-synchronize while in the "data following mode".

Most Irwin drives generate a read data signal with the 50/50 duty cycle when transiting servo headers.

When these factors are combined, and a tape is read, the following sequence of events occurs during a tape read operation: a servo header crosses the head. The drive sends a 50/50 duty cycle 250 KHz signal on the read data line. The PLL looses sync (that is, the loop control voltage goes to a rail). The end of the servo header crosses the head and the drive gives an **INDEX** pulse. No corresponding **VCO SYNC** inhibit is generated by the 72065 (this would normally put the PLL back on track). Sector 1 crosses the head but the PLL is still too far off to read the sector. The 72065 generates a record-not-found error.

Some Irwin drives are fitted with a data compensator board. This board has a circuit which alters the 50/50 duty cycle to a value which allows most of these controllers to maintain PLL synchronization. One exception is certain Model 80s.

For Micro Channel systems which do not have the compensator (and certain Model 80s which do), this problem can be circumvented by software. The technique relies on a feature of the 72065 (and other controllers in the 765 class): a **VCO SYNC** inhibit is generated just after the last byte of a **READ** command is sent to the controller. Inhibiting the **VCO SYNC** pin (which is normally telling the PLL to lock on incoming read data) causes the **VCO**'s input to be switched to a reference. This results in quickly returning the PLL to a state in which it will be nearly synchronized with the "real" read data. **VCO SYNC** inhibition results from programming the floppy controller using a "wait-for-index" algorithm.

The wait-for-index algorithm sends all but the last byte of the data transfer command to the 72065. It then waits for a logical high to low transition of the floppy **INDEX** signal. The wait is accomplished by polling a special I/O port (at address 03F0h) provided by the Micro Channel floppy controller. The wait is used to delay the writing of the last byte of the 72065 transfer command until after the **INDEX** transition. As a result, the 72065 generates an inhibit pulse on **VCO SYNC** after **INDEX**, but with sufficient lead time to allow the PLL to achieve synchronization. Thus, sector one's ID can be correctly read.

As no index interrupt is available, wait-for-index polls to accomplish its task. The sought **INDEX** event is time critical. Under DOS, control for polling comes from either a task time startup thread, or the controller

completion or timer tick interrupt. Under OS/2 and UNIX/XENIX a high priority daemon thread is awakened to poll for the index transition. Using the wait-for-index algorithm has the following drawback: all other system task time processing is stopped until index polling is complete. This means that the user will see sluggish system performance at certain times: typically, a 3 or 4 second dead period at tape track switch time. This may prove unacceptable in certain installations.

M/m: Micro-Channel-Architecture/PC-Bus

M This computer has a Micro Channel Architecture bus.

m This machine does not have a Micro Channel Architecture.

When neither "M" nor "m" is set, automatic configuration determines if Micro Channel Architecture hardware is present. The algorithm used depends upon the operating system and driver version. The M/m option is used for automatic configuration of the B/b, I/i, and P/p options.

If the string "EISA" is found at physical memory location 0xfffd9, (BIOS ROM location F000:FFD9) this is not a Micro Channel Architecture. Otherwise, if all 8 bits of the I/O port at address 0x0080 (DMA page register 0 in an AT compatibles) can be modified this is an AT 286/386 compatible. Otherwise this is a Micro Channel Architecture.

O/o: System Controller Does/Doesn't Support 1-Meg Transfers

O The system controller supports one Megabit data transfers.

o One Megabit transfers are not supported by the system controller.

When neither "O" nor "o" is set, automatic configuration determines whether the system controller supports 1-Megabit transfer rates. This is important when a 2120 is attached to the system controller. If the controller does not support 1- Megabit transfers, 500-Kilobit transfers are used for 80 and 120 MegaByte tapes. The driver detects the presence of the following 1-Megabit controllers: Intel 82072 and 82077.

P/p: 4251 Is/Isn't Present

P A 4251 board is present in the system and has its jumpers configured to address the 4251's digital output register (DOR) at 0372h. When present, the tape driver echoes commands sent to the system floppy controller's DOR (at I/O port address 03F2h) to the 4251's DOR.

p No 4251 board present.

When neither "P" nor "p" is set, and when running in a PC-us (non-Micro Channel) machine (see the M/m option), automatic configuration determines the presence of a 4251 board by reading I/O port 0372h and comparing the input byte with the signature of the 4251. The 4251 signature byte is 42h. See also 4151=***port***.

Q/q: Compaq Portable III Piggy Back Tape Unit Is/Isn't Present

Q A Compaq Portable III piggy back tape unit is present.

q No Compaq Portable III piggy back tape unit is present.

When neither "Q" nor "q" is set, the algorithm used to test for presence of an alternate (Compaq Portable III piggy back) controller does the following: first the model byte is checked to see if the machine is other than an 8086 class machine (that is, the model byte must be less than FE). If this test passes, the BIOS address F000:FFEA is checked for the string "COMPAQ". When a match is found, the I/O port at the 0374 (that is, the floppy controller chip status port) is read and the three low order bits are tested. If all three bits are zero, the alternate controller is present.

When an alternate floppy controller is present, the following port addresses are used by default:

Base	DOR	765 Stat	765 Data	Clock	
03F0	03F2	03F4	03F5	03F7	Primary FLOPPY controller
0370	0372	0374	0375	0377	Alternate TAPE controller

See the Controller Parameter Configuration section for information on reconfiguration of the default base address.

X/x: One Megabit Transfers Are/Aren't Allowed

X Allow 1 Megabit transfers when conditions permit.

x Never allow 1 Megabit transfers.

By default, 1 Megabit transfers "X" are allowed. If 1 Megabit transfers overload the system bus, the "x" option should be configured.

IRDBG: debugging options

Several debugging flags are available:

s Drive search debug

When "s" is set, the result of the tape drive search (presence test) is shown. The following shows an example:

```
4100MC:3=CTLRNOTFND :2=CTLRNOTFND :1=CTLRNOTFND :0=CTLRNOTFND
4100MCB:3=CTLRNOTFND :2=CTLRNOTFND :1=CTLRNOTFND :0=CTLRNOTFND
4100:3=DRVNOTFND :2=tapedrive :1=DRVNOTFND :0=DRVNOTFND
4100B:3=CTLRNOTFND :2=CTLRNOTFND :1=CTLRNOTFND :0=CTLRNOTFND
ALTFDC:3=CTLRNOTFND :2=CTLRNOTFND :1=CTLRNOTFND :0=CTLRNOTFND
SYDFDC:3=DRVNOTFND :2=tapedrive :1=nottested
```

The order of drive presence testing is shown left to right and top to bottom. On a given line, the left most field has a symbol which represents a controller. Numeric fields preceded by a colon (:) give the unit select in the range 0 through 3. Fields preceded by an equals sign (=) have a symbol which represents result of tape drive presence testing for the controller and unit. These fields normally have an uppercase symbol which represents a driver error code. Two special strings are used: "tapedrive" if a drive was found, or "nottested" if drive presence was not tested.

i Initialization value debug

When "i" is set, certain initialization values are displayed. The following is an example:

```
hz=18 12_us_scaler=12 scaler_loops=27510 model=0x1FC
is64kdma=1 demanddma_ok=1
isuchannel=0 port_4251=3F0
timers=[ 0 1 2 1 2 7 19 37 181 235 ]
```

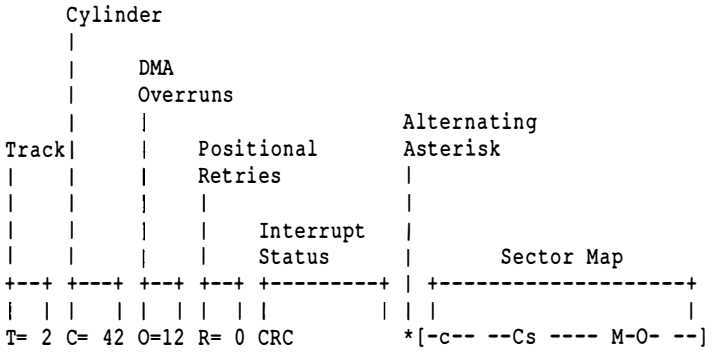
r Interrupt debug

When "r" is set, a character is displayed for each interrupt processed by the driver's finite state machine. In addition, reset cycles are shown. The following lists the characters and their meanings:

Character	Meaning
N	Floppy controller (NEC) interrupt
T	Timer Interrupt
R	Reset sent to floppy controller (start of reset)
r	Reset complete

x Data transfer debug

When "x" is set, the status of a transfer request is displayed at interrupt time. The display is similar to that shown below:



Track (T=*decimal number*) has the transfer request's track number.

Cylinder (C=*decimal number*) has the transfer request's cylinder number (tape block for the given track).

DMA Overruns (O=*decimal number*) has a count of DMA overruns (excluding, if indicated by in the Interrupt Status, the current **DMAOVERRUN**).

Positional Retry (R=*decimal number*) has the current positional retry number for the request. Note that a "free" retry is allowed under the following conditions:

1) A track switch was performed.

2) The tape is moving logically forward: this transfer request's target head, cylinder, and sector addresses match current values, but there is some positional uncertainty because this transfer request was not started on the completion thread of the previous request (That is, the period of time the tape has been moving between requests is not known).

3) A DMA overrun has occurred during the previous pass for a given read/write/verify request.

Interrupt Error has the current reason for the interrupt displayed symbolically.

Alternating Asterisk (*) This one character field is alternately set with an asterisk (*) and a space " " character so that screen updates may be distinguished.

Sector Map (`[-c-- --Cs ---- M-O- --]`) gives a visual indication of the status of each sector when an error occurs. Each printing character in the sector map represents the status of a sector. Before the start of a transfer, each entry is set to "s". On successful transfer of a sector, the corresponding entry is set to a hyphen(-). The following is a list of characters which appear in the sector map and their meanings:

Character	Interrupt Number	Symbol	Error Description
-	0	**IE_NOERR**	No error
C	12	**IE_CRC**	Data CRC error
c	13	**IE_IDCRC**	ID CRC error
s	14	**IE_RECNOTFND**	Record not found
M	16	**IE_DATAMARK**	No data address mark
O	17	**IE_DMAOVERRUN**	DMA overrun
?	other	unexpected	Unexpected value

IRDRV, IRSRCH: drive search control

IRDRV drive searching sequence (old method)
IRSRCH drive searching sequence (new method)

The tape driver uses a default drive searching sequence to test for the presence of tape drives. The default sequence may be replaced with a user configured sequence using either the **IRDRV** or **IRSRCH** variables. This is useful in situations where tape drives are erroneously detected by the default sequence, or where multiple tape drives are supported and a different mapping of logical to physical drives is desired. For example,

```
IRSRCH=SYSFDC:3,4100:2
```

says search for tape drives at unit select 3 on the system floppy controller, and unit select 2 on an Irwin 4100 PC bus controller.

The equivalent **IRDRV** specification is:

```
IRDRV=04,43
```

or alternately:

```
IRDRV=4,43
```

IRDRV specifications use a 2-digit number to specify a controller and unit select. The high-order digit gives the controller, and the low-order digit the unit select. If the high-order digit is missing, 0 (for the system floppy controller) is assumed. Note that the unit select used by **IRDRV** is in the range 1-4 while the unit select used by **IRSRCH** is in the range 0-3.

The following is a list of controllers supported by **IRSRCH** and **IRDRV**:

IRSRCH, IRDRV Name	High Digit	Controller
SYSFDC	0	System floppy (2.00)
ALTFDC	1	Alternate floppy (2.00)
4100MC	2	Irwin 4100 Micro Channel (2.01)
4100MCB	3	Second 4100 Micro Channel (2.01)
4100	4	Irwin 4100 PC Bus (2.01)
4100B	5	Second 4100 PC Bus (2.02)

The syntax of an **IRSRCH** drive search sequence specification is:

IRSRCH =	***searchlist***	
searchlist =	***searchspec***	
	searchspec,searchlist	
searchspec =	***controller:unitlist***	
controller =	**SYSFDC**	(System floppy controller)
	ALTFDC	(Alternate controller)
	4100MC	(Irwin 4100 Micro Channel tape controller)
	4100MCB	(Second 4100 Micro Channel controller)
	4100	(Irwin 4100 PC Bus tape controller)
	4100B	(Second 4100 PC Bus controller)
unitlist =	***unit***	
	unit:unitlist	
unit =	0, 1, 2, 3	

The syntax of an **IRDRV** drive search sequence specification is:

IRDRV =	***searchlist***	
searchlist =	***searchspec***	
	searchspec,searchlist	
searchspec =	***controllerdigit:unitdigit***	
controllerdigit =	0	(System floppy controller, may be omitted)
	1	(Alternate controller)
	2	(Irwin 4100 Micro Channel tape controller)
	3	(Second 4100 Micro Channel controller)
	4	(Irwin 4100 PC Bus tape controller)
	5	(Second 4100 PC Bus controller)
unitlist =	***unit***	
	unit:unitlist	
unitdigit =	0, 1, 2, 3	

SYSFDC, ALTFDC, 4100, 4100B: controller parameter configuration

Certain variables may be set to specify tape controller specific parameters. For example:

```
4100=P:370,I:6,D:2,T:2,T:0
```

says an Irwin 4100 PC bus controller is installed and configured with a base I/O Port address (P) 0370 hexadecimal, using IRQ (I) 6, DMA channel (D) 2, and has two tape units (T), one wired for physical unit select number 2, and the other 0.

The general form for controller parameter specifications is:

controller =	***paramlist***	
paramlist =	***parameter***	
	parameter,paramlist	
parameter =	***name:value***	
controller =	**SYSFDC**	(System floppy controller)
	ALTFDC	(Alternate controller)
	4100	(Irwin 4100 PC Bus controller)
	4100B	(Second 4100 PC Bus controller)
name =	P	(Base I/O Port address)
	I	(Interrupt Request line (IRQ))
	D	(DMA channel)
	T	(Tape unit number [0-3])
value =	[0123456789abcefABCDEF]+ (Hexadecimal number)	

4100 PC configuration switch settings

The following tables contain the 4100 switch settings.

Base Address	SW1	SW2	SW3	SW4
300	ON	ON	ON	ON
310	off	ON	ON	ON
320	ON	off	ON	ON
330	off	off	ON	ON
340	ON	ON	off	ON
350	off	ON	off	ON
360	ON	off	off	ON
* 370	off	off	off	ON
380	ON	ON	ON	off
390	off	ON	ON	off
3a0	ON	off	ON	off
3b0	off	off	ON	off
3c0	ON	ON	off	off
3d0	off	ON	off	off
3e0	ON	off	off	off
3f0	off	off	off	off

DMA Channel	SW5	SW6	SW7	SW8
1	ON	off	ON	off
* 2	off	ON	off	ON

IRQ	SW9	SW10
3	ON	off
* 6	off	ON

* factory setting

4251: floppy extender address configuration

The Irwin 4251 adapter board augments the system floppy controller. It extends the total number of drives which may be attached from 2 to 4, and allows for the attachment of an external drive. The 4251 uses a single drive select I/O port. By design, the 4251 I/O port partially mimics the functionality of the system floppy controller's drive select port. The system controller's drive select port is called the Digital Output Register (DOR). When written with certain values, both the system controller's DOR and the 4251's drive select port activates a drive select line at the floppy interface. In the standard "as shipped from the factory" configuration, the 4251's port is addressed at 03F2 hexadecimal. The same address is used by system floppy controller's DOR. Thus, in the standard configuration, the 4251 monitors (that is, listens to and uses) bytes written to the system's DOR to select a drive. The 4251 uses unit selects 2 and 3. Unit selects are used by the software and should not be confused with the DRIVE SELECT jumpers on the tape drive which are almost always set to DRIVE SELECT 2. In certain hardware environments, the standard 4251 configuration either does not detect the presence of or fails to write tapes in a tape drive.

When a 4251 is configured for the standard address and is connected to:

- a DTC controller, data is never written to tape. The reason: DTC controllers disable the floppy interface **WRITE GATE** signal when unit selects 2 or 3 (the third and fourth) selects are activated. This means the tape drive's write circuitry is never enabled.
- an Adaptec suffix "B" controller (for example, ACB-2xxxB or 1542B SCSI controllers), driver software never detects the presence of a tape drive. The reason: Adaptec suffix "B" controllers drive the TRACK 0 line active for unit selects 2 or 3. The TRACK 0 line is the line used by the drive to return the results of status requests and motion commands issued by the driver software.

The conditions listed in the above three paragraphs can be overcome. Typically, reconfiguring the 4251 to use the recommended alternate address by installing the A7 jumper allows the tape drive to function correctly. When this is done, the 4251's I/O address moves from 3F2 to 372 hexadecimal.

When configuring the address of the Irwin 4251, the board address jumpers are changed from the "as shipped" A0, A2, A3 position. Normally, the change involves reinstalling a jumper stored on one pin of the A7 pin pair to connect the "A7" pin pair. This selects the address 372. However, when a secondary floppy controller (such as the Irwin 4100) or other adapter is present, the 372 address may be in conflict. In general, a secondary floppy controller uses addresses in the range 370 through 377, which includes the alternate "372" address of the 4251. To resolve this conflict, the 4251 can be re-addressed. In addition, the tape driver software must be informed of the new address.

The following information is given to aid in understanding of the relationship of the 4251 and tape driver software, the meaning of the 4251 jumpers A0 through A9, and an example of a non-standard configuration.

At initialization, the tape driver software tests for the presence of a 4251 at an alternate address. By default, the alternate address is 372 hexadecimal. (To select the 372 address on the 4251 install jumpers across the A0, A2, A3, and A7 pin pairs.) The test reads a byte from the alternate address and compares the byte with the signature. When the 4251's select port is read, a signature byte (42 hexadecimal) is returned. If the signature compares, the driver sends select bytes to both the system's DOR and the 4251's port. The default alternate address may overridden by using the variable named "4251". For example,

```
4251=31f
```

tells the driver to test and use, if present, the port at 030F hexadecimal.

The 4251's port uses a single 10-bit I/O port address. The address is set using the jumper pin pairs labeled A0 through A9. Each jumper pin pair corresponds directly with an I/O port address bit. When a jumper pin pair is connected, the corresponding address bit is set to a logical 0. When the pin pair is disconnected, the address bit is set to a logical 1.

For example, to address the 4251 at 31F (an address which is unlikely to conflict with standard adapters), connect jumper pin pairs A5, A6, and A7.

See also

tape(C), **tape**(HW)

mdevice

device driver module description file

Description

/etc/conf/cf.d/mdevice is a one-line description of each device driver and configurable software module in the system (except for file system types, see **mfsys**(FP)). Each line in *mdevice* represents the *Master* file component from a Driver Software Package (DSP) which is either delivered with the base system or installed later via **idinstall**(ADM).

Each line contains several fields separated by spaces or tabs; these are described below. Each field must be supplied with a value or a "-" (dash).

1. *Device name*: This field is the internal name of the device or module, and may be up to 8 characters long. The first character of the name must be an alphabetic character; the others may be letters, digits, or underscores.

2. *Function list*: This field is a string of characters that identify driver functions that are present. Using one of the characters below requires the driver to have an entry point (function) of the type indicated. When the kernel is relinked an entry is made in the kernel switch tables for each function in the list. If no functions in the following list are supplied, the field should contain a dash. Most drivers use some or all of the following functions:

   I **init** function. Called by the system during the system boot sequence to initialize the driver or module for use. Interupts are not enabled when this function is called.

   o **open** function. Called whenever a device node associated with this driver is opened.

   c **close** function. Called whenever the last open connection to a device node associated with this driver is closed.

   i **ioctl** function. I/O control function called to issue control commands to the driver. Used by character devices only.

   r **read** function. Called to read data from a device controlled by this driver. Used by character devices only.

   w **write** function. Called to write data to a device controlled by this driver. Used by character devices only.

The remaining functions in this list are likely to be used only by drivers with special requirements:

h **halt** function. Called during system shutdown. This might be provided if, for instance, the driver needs to reset hardware in preparation for a warm boot.

p **poll** function. Called once every system clock-tick. This might be required (for instance) for a device which loses interrupts and needs to be re-primed periodically.

E **kenter** function. Called by the system whenever kernel mode is entered. Provided for drivers which have specific actions to carry out whenever this happens.

X **kexit** function. Called on exit from kernel mode to user mode. Provided for drivers which have specific actions to carry out when this happens.

e **exec** function. Called when a process with an open (or previously open) connection to this device, using this driver, issues an **exec**(S) system call. Provided for devices which must perform specific actions when this situation arises.

s **start** function. Secondary initialization function called by the system late in the system boot sequence. Provided for drivers which have initialization functions which must be deferred until after system interrupts have been enabled.

P **pminit** function. Pre-main initialization function called by the system very early in the system initialization process. Provided for drivers which need to be initialized early. Normally this function is used by a kernel debugger so that it is available to the user as early as possible.

x **exit** function. Called when a process with a previously open connection to this device exits. Provided for devices which must perform specific actions when this situation arises.

S **swtch** function. Called by the system whenever a process context switch occurs. This is provided for drivers which have specific actions to carry out whenever a context switch occurs.

Note that if the device is a 'block' type device (see field 3 below), a **strategy** routine and a **print** routine are mandatory.

3. *Characteristics of driver*: This field contains a set of characters that indicate the characteristics of the driver. If none of the characters below apply, the field should contain a dash. The legal characters for this field are:

   a The driver is installed automatically.

   b The device is a 'block' device.

   c The device is a 'character' device.

   d The device can accept 32-bit addresses for DMA transfer, but cannot directly access addresses above 16Mb (see "x" below).

   h The device is an SCSI host adapter.

   i The device driver is installable.

   n The driver is not installable.

   o This device may have only one *sdevice*(F) entry.

   p The device is an SCSI peripheral.

   r This device is required in all configurations of the kernel. This option is intended for drivers delivered with the base system only. Device nodes (special files in the */dev* directory), once made for this device, are never removed. See **idmknod**(ADM).

   s Suppress device count field.

   t The device is a *ttym* and has a **_tty** table.

   x The device can perform DMA to addresses above 16Mb without the need for memory windowing support from dedicated hardware and the kernel. This is only valid if the "d" characteristic is also set.

   C The device is a scatter/gather device which performs cluster I/O requests. (Block devices only.)

   D This option indicates that the device driver can share its DMA channel.

   G The interrupt handler specified in the *sdevice*(F) entry is not installed, but is checked for conflicts with other devices. This is used when you wish to associate a device with a specific device group.

   H This device driver controls hardware. This option distinguishes drivers that support hardware from those that are entirely software (pseudo-devices).

I Ignore *pack.d* directory. This option prevents the system from looking for driver components when relinking the kernel. This option is used when the functions associated with this driver name are actually included in another driver.

M This driver defines a range of extended minor device numbers for a driver already defined in a preceding entry. This also causes the Minumum and Maximum fields to be interpreted differently. (See fields 7 and 8 below.)

N No *Driver.o* or *space.c* file.

O This option indicates that the IOA range (*sdevice*(F) columns 7 and 8) of this device may overlap that of another device, which must also have the 'O' characteristic.

S This device driver is a STREAMS module.

Z This driver may have multiple entries in the *mdevice* file, with different major numbers.

4. *Handler prefix*: This field contains a character string which is used as a prefix for all the externally-known handler routines associated with this driver. The string may be up to 4 characters long.

5. *Block major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'block' type device, a value will be assigned by **idinstall** during installation.

6. *Character major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'character' type device (or 'STREAMS' type), a value will be assigned by **idinstall** during installation.

7. *Minimum units*: This field is an integer specifying the minimum number of these devices that can be specified in the *sdevice*(F) file. If "M" appears in the characteristics field, this is the base major number of the driver to which the extended minor numbers apply (i.e. BASE major).

8. *Maximum units*: This field specifies the maximum number of these devices that may be specified in the *sdevice*(F) file. It contains an integer. If "M" appears in the characteristics field, this is the offset at which the range of the extended minor numbers begin (i.e. OFFSET). Must be a multiple of 256.

9. *DMA channel*: This field contains an integer that specifies the DMA channel to be used by this device. If the device does not use DMA, place a "-1" in this field. Note that more than one device can share a DMA channel provided that each sharing device has the "D" characteristic set.

Specifying STREAMS modules and device

STREAMS modules and devices are treated in a slightly different way from other drivers in all UNIX systems, and their configuration reflects this difference. To specify a STREAMS device driver, its *mdevice* entry should contain both an "S" and a "c" in the "characteristics" field (see field 3. above). This indicates that it is a STREAMS device and that it requires an entry in the UNIX kernel's **cdevsw** table, where STREAMS devices are normally configured into the system.

A STREAMS module that is not a device driver, such as a line discipline module, requires an "S" in the "characteristics" field of its *mdevice* file entry, but should not include a "c", as a device driver does.

See also

configure(ADM), **idbuild**(ADM), **idinstall**(ADM), **mfsys**(FP), **sdevice**(F)

Device Driver Writer's Guide

mmdftailor

provide run-time tailoring for the MMDF mail router

Description

The MMDF mail router reads site-dependent information from the ASCII file */usr/mmdf/mmdftailor* each time it starts up.

Keywords in the tailor file are not case-sensitive; however, case is important for filenames and similar values. Use quotation marks to delimit strings to prevent them from being parsed into separate words accidentally.

The following alphabetical list describes most of the information you can set in the *mmdftailor* file. For information about additional channel-specific settings, refer to the documentation about the particular channel.

ALIAS defines an alias table. The following parameters can be used:

- **table** specifies the name of the table to be associated with this alias entry
- **trusted** allows the entries in the table to cause delivery to files and pipes
- **nobypass** does not allow the ~***address*** alias bypass mechanism to work on this file

Here is an example:

```
ALIAS table=sysaliases, trusted, nobypass
```

AUTHLOG controls authorization information. See **MCHANLOG** and **MLOGDIR**.

AUTHREQUEST is the address to which users should mail if they have questions about why a message was not authorized for delivery. It is also used as the sender of authorization warning messages. It is not used if authorization is not enabled on some channel. See the **auth** parameter under **MCHN**.

MADDID controls whether **submit** adds `Message-ID:` header lines if they are missing from messages. It takes a number as an argument. If the number is 0, no action is taken. If the number is non-zero, then **submit** adds `Message-ID:` header lines if they are missing from messages.

MADDRQ is the address files directory. If it is not a full pathname, it is taken relative to **MQUEDIR**.

MCHANLOG controls MMDF logging, except for authorization information and information produced by **deliver** and **submit**. See also **MMSGLOG**, **AUTHLOG**, and **MLOGDIR**.

Logging files and levels can also be specified in the channel descriptions. The logging file, if specified there, overrides the **MCHANLOG** definition. The logging level for the channel is set to the maximum of the **MCHANLOG** level and the channel description's level. The **MCHANLOG** level can therefore be used to increase logging on all channels at once.

Here is an example:

```
MCHANLOG        /tmp/mmdfchan.log, level=FST, size=40,
                stat=SOME
```

An argument without an equal sign is taken as the name of the log. Logging levels are:

FAT logs fatal errors only
TMP logs temporary errors and fatal errors
GEN saves the generally interesting diagnostics
BST shows some basic statistics
FST gives full statistics
PTR shows a program trace listing of what is happening
BTR shows more detailed tracing
FTR saves every possible diagnostic

The **size** parameter is the number of 25-block units you will allow your log file to grow to. When a log file reaches that size, that logging either stops or cycles around overwriting old data (see **cycle**).

The **stat** parameter sets up various status flags for logging:

close closes the log after each entry; this allows other processes to write to it as well

wait if the log is busy, waits a while for it to free

cycle if the log is at the maximum length specified with the **size** parameter, then cycles to the beginning

some sets the values **close** and **wait** (the most common setting)

timed opens the log and, after the timeout period (for example, 5 minutes), closes the log and reopens it; this option overrides all other options (used to reduce the overhead of re-opening the log for every entry while still retaining the ability to move the log file out from under a running process and have the process begin logging in the new log file soon thereafter)

Tailoring of the log files is generally performed at the end of the tailor file to prevent logging the tailoring action itself, thereby needlessly filling the log files when higher tracing levels are enabled. If you have bugs in the tailoring, you can move the log-file tailoring closer to the top of the tailor file.

MCHN defines a channel. The following parameters can be used:

name the name of the channel

show a descriptive name used by certain programs as a display line to explain the function of the channel

que the queue subdirectory of */usr/spool/mmdf/lock/home* in which to queue messages for this channel; MMDF prefixes the name with *q.* to form the subdirectory name.

tbl the abbreviated name (from **MTBL**) for the associated table that lists the hosts that are accessible on this channel. If the specified table has not been previously defined, it will be defined with the same **name**, **file**, and **show** parameters as for this channel (do not define two channels that process the same queue, but use different tables because it will cause queue structure problems).

pgm the channel program (in */usr/mmdf/chans*) to invoke for this channel. This program takes mail from **deliver**(ADM) and carries it to its destination on the local machine or across the network to a remote machine.

mod the delivery mode for the channel; if several values are selected, they are cumulative:

- **reg** regular mode (the default). This mode queues mail, but does not send it; you must run **deliver** (manually, with **cron**(C), or as a background program) to actually send mail through the regulated channel.
- **host** same as **reg**, but specifies that **deliver** sort by host after sorting by channel, which allows as many mail messages as possible to get sent to a particular host before the connection is broken.
- **bak** channel can be invoked only by the background deliver daemon
- **psv** channel is passive; it is a pickup-type channel (for example, pobox)
- **imm** channel can be invoked immediately; no need to batch up mail
- **pick** channel can pick up mail from the remote host
- **send** channel can send mail to the remote host

ap the type of address parsing to use for reformatting headers on messages going out on this channel; if several values are selected, they are cumulative:

- **same** does not reformat headers **822** converts to RFC822-style source routes (for example, @A:B@C) **733** converts to RFC733-style source routes (for example, B%C@A)
- **nodots** selects output of leftmost part of domain names (for example, A in A.B.C) for sites that cannot handle domains (usually used in conjunction with the **known=** parameter to hide domain names behind a smart relay)

lname a name overriding the default **MLNAME** value for this channel (used when the channel should have non-standard values for the local domain)

ldomain a name overriding the default **MLDOMAIN** value for this channel

host the name of the host that is being contacted by this channel, usually used in the phone and pobox channels, or the name of the relay host when all mail to hosts in this channel's table gets relayed to one host (this is required on the *badusers* and *badhosts* pseudo-channels; it must be set to the local host for the list channel)

poll the frequency of polling the remote machine (0 disables polling, -1 requests polling to be done every time the channel is started up, any other value is the number of 15-minute intervals to wait between polls); if any mail is waiting to be sent, this value is ignored because a connection is always attempted

insrc a table of hosts controlling message flow

outsrc see **insrc**

indest see **insrc**

outdest see **insrc**

known a table of hosts that are known on this channel; be sure that the table contains your own fully specified host name

confstr a channel-specific configuration string. See the individual manual pages for the channel for more information.

auth specifies the authorization tests that are made on this channel:

- **free** default, no checking takes place
- **inlog** log information for incoming messages
- **outlog** log information for outgoing messages
- **inwarn** warn sender of incoming message if authorization is inadequate (the message still goes through)
- **outwarn** as **inwarn**, but for outgoing messages
- **inblock** reject incoming messages that have inadequate authorization
- **outblock** as **inblock**, but for outgoing messages
- **hau** host and user authorizations are required
- **dho** (direct host only) when applying host controls, the message must be associated with a user local to that host (that is, no source routes)

ttl (time-to-live) specifies the number of minutes for which retries to a host are blocked when **deliver** detects a connection failure to that host; this value can be overridden on the **deliver** command line (default is 2 hours)

log the name of the channel log file to be used instead of the default **MCHANLOG**

level the logging level for this channel (see also **MCHANLOG**)

Here is a simple example:

```
MCHN    name=local, que=local, tbl=local,
        show="Local Delivery", pgm=local,
        poll=0, mod=imm, ap=822, level=BST
```

If the first argument does not have an equal sign, the values of the **name**, **que**, **tbl**, **pgm**, and **show** parameters take on this value.

MCHNDIR is where the channel programs are to be found.

MCMDDIR is the default commands directory where the various MMDF commands are located. Any command that does not have a full pathname is taken relative to this directory.

MDBM tells MMDF where to find the database file containing the associative store. DBM-style databases get their speed and flexibility by performing dynamic hashing on an associative store. This can get quite large. By default, the file is located in the **MTBLDIR** directory, but it might need to be relocated due to its size.

MDFLCHAN sets the default channel to something other than local.

MDLV is the name of the file used for tailoring the delivery for each user.

MDLVRDIR is the directory in which to deliver mail. If this is null, then the user's home directory is used. See also **MMBXNAME** and **MMBXPROT**.

MDMN defines a domain. The following parameters can be used:

name an abbreviated name for the domain

show a display line, which is used for formatting purposes to explain what the domain is all about

dmn the full name (x.y.z...) of this domain

table the associated table entry of known sites in this domain; if the specified table has not been previously defined, it will be defined with the same **name**, **file**, and **show** parameters as for this domain

Here is an example:

```
MDMN    name="Root", dmn="", show="Root Domain",
        table=rootdomain
```

If the first argument does not have an equal sign, the values of the **name**, **dmn**, and **show** parameters take on this value. If no **table** parameter is specified, it defaults to the value of the **name** parameter.

MFAILTIME is the time a message can remain in a queue before a failed-mail message is sent to the sender and the message is purged from the queue. See also **MWARNTIME**.

MLCKDIR is the directory where the locking of files takes place: this is dependent on what style of locking you are doing.

MLCKTYPE specifies the locking protocol for MMDF to use when locking mailboxes. Use one or more of the following keywords with **MLCKTYPE**:

Keyword	Lock File
advisory	System V **fcntl()** kernel file locking
v7	Version 7 and System V Release 3, and earlier file locking
xenix	XENIX file locking
all	all known locking protocols

If you specify more than one locking keyword, all locks must be successful before MMDF considers the mailbox locked. Here is an example **MLCKTYPE** setting:

```
MLCKTYPE advisory, xenix
```

MLDOMAIN gives your full local domain (this, combined with the **MLNAME**, and possibly the **MLOCMACHINE**, gives the full network address).

MLISTSIZE specifies the maximum number of addresses in a message before it is considered to have a "big" list. If there are more than the maximum number of addresses, then MMDF does not send a warning message for waiting mail and only returns a "citation" for failed mail. A citation consists of the entire header plus the first few lines of the message body.

MLNAME is the name of your machine or site as you wish it to be known throughout the network, which can be a generic host name used to hide a number of local hosts. If it is a generic host name, internal hosts are differentiated by **MLOCMACHINE**. See also **MLDOMAIN**.

MLOCMACHINE is the local name of the machine. It is used by a site that has several machines, but wants the machines themselves to appear as one address. See also **MLNAME** and **MLDOMAIN**.

MLOGDIR is the default directory in which the log files are kept. See also **MMSGLOG**, **NAUTHLOG**, and **MCHANLOG**.

MLOGIN is the user who owns the MMDF transport system.

MMAXHOPS specifies the maximum number of `Received:` or `Via:` lines in a message before it is considered to be looping and is rejected.

MMAXSORT controls sorting of messages based on the number of messages in the queue. If the number of messages in the queue is less then **MMAXSORT**, the messages are sorted by arrival time and are dispatched in that order; otherwise, a message is dispatched as it is found during the directory search.

MMBXNAME is the name of the mailbox. If this is null, then the user's login name is used. See also **MDLVRDIR** and **MMBXPROT**.

MMBXPREF is a string written before the message is written into the mailbox. It is usually set to a sequence of ⟨Ctrl⟩**A** characters. The default **MMBXPREF** value looks like this:

```
MMBXPREF "\001\001\001\001\n"
```

See also **MMBXSUFF**.

The values of **MMBXPREF** and **MMBXSUFF** should consist of non-printable characters only and must end in a newline.

MMBXPROT gives the protection mode in octal for the user's mailbox. See also **MDLVRDIR** and **MMBXNAME**.

MMBXSUFF is a string written after the message is written into the mailbox. It is usually set to a sequence of ⟨Ctrl⟩**A** characters. The default **MMBXSUFF** value looks like this:

```
MMBXSUFF "\001\001\001\001\n"
```

See also **MMBXPREF**.

MMSGLOG controls the logging information produced by **deliver** and **submit**. See also **MCHANLOG**, **AUTHLOG**, and **MLOGDIR**.

MMSGQ is the directory for the files of message text. If it is not a full pathname, it is taken relative to **MQUEDIR**.

MPHSDIR is the directory in which the timestamps for the channels are made, showing what phase of activity they are in.

MQUEDIR is the parent directory for the various queues and address directories.

MQUEPROT gives the protection mode in octal that the parent of the **MQUEDIR** directory should have.

MSIG is the signature that MMDF uses when notifying senders of mail delivery problems.

MSLEEP is the length of time in seconds that the background delivery daemon sleeps between queue scans.

MSUPPORT is the address to which to send mail that MMDF cannot cope with (that is, that MMDF cannot deliver or return to its sender).

MTBL defines an alias, domain, or channel table. The following parameters can be used:

- **name** a short name by which the table can be referred to later in the file
- **file** the file from which the contents of the table are built
- **show** a display line, which is used for reporting purposes to explain what the table is all about
- **flags** indicates additional properties about the table being defined. Use this to specify the source type, nameserver lookup parameters, and control of partial lookups table options.

The following three source types that define the table:

- **file** comes from a sequentially read file (default).
- **dbm** is in the MMDF hashed database built with **dbmbuild**.
- **ns** the table data is obtained by making queries on a nameserver.

domain specifies to look up the given address in the domain specified by **dmn=** parameter of the domain definition.

channel specifies to look up the given fully-qualified domain name to determine the address(es) to use in delivering via SMTP.

alias specifies to look up the given alias name in alias tables.

abort specifies that if MMDF encounters a problem when searching an **ns**-type domain table, MMDF does not search any other domain tables (because the **ns**-type domain table is the most reliable).

route specifies to search for successive subdomains of the domain if no exact match exists.

partial specifies to search for the domain in other domain tables; this allows users to omit the full domain specification when referring to local machines.

Note that MMDF treats **flags=file** and **flags=dbm** the same. In the case of an **ns**-type table, the **flags** field specifies the type of nameserver lookup (either **domain**, **channel**, or **alias**).

A typical example might be:

```
MTBL    name=aliases, file=aliases,
        show="User & list aliases"
```

If the first argument does not have an equal sign, the values of the other parameters take on this value. The following example sets the **name**, **file**, and **show** parameters to the string "aliases", then resets the **show** parameter to the string "Alias table".

```
MTBL aliases, show="Alias table"
```

MTBLDIR is the default directory for the table files.

MTEMPT is the temporary files directory. If it is not a full pathname, it is taken relative to **MQUEDIR**.

MWARNTIME specifies the time in hours that a message can remain in a queue before a warning message about delayed delivery is sent to the sender. See also **MFAILTIME**.

UUname defines the UUCP sitename (short form, not full path) and is used only by the UUCP channel. See also **MLNAME**.

See also

dbmbuild(ADM), **mmdf**(ADM), **queue**(F), **tables**(F)

"Setting Up Electronic Mail" in the *System Administrator's Guide*

Value added

mmdftailor is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

mnttab

format of mounted filesystem table

Syntax

#include<*stdio.h*>
#include<*mnttab.h*>

Description

The */etc/mnttab* file contains a table of devices mounted by the **mount**(ADM) command.

Each table entry contains the pathname of the directory on which the device is mounted, the name of the device special file, the read/write permissions of the special file, and the date on which the device was mounted.

The maximum number of entries in **mnttab** is based on the system parameter **NMOUNT**, which defines the number of mounted special files which are allowed.

See also

mount(ADM)

mtune

tunable parameter file

Description

/etc/conf/cf.d/mtune contains information about all the system tunable parameters. Each tunable parameter is specified by a single line in the file, and each line contains the following whitespace-separated set of fields:

1. *parameter name*: a character string no more than 20 characters long. It is used to construct the preprocessor "#defines" that pass the value to the system when it is built.

2. *default value*: this is the default value of the tunable parameter. If the value is not specified in the *stune* file, this value will be used when the system is built.

3. *minimum value*: this is the minimum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will verify that the new value is equal to or greater than this value.

4. *maximum value*: this is the maximum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will check that the new value is equal to or less than this value.

The file *mtune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *mtune* file to change the setting of a system tunable parameter. Instead the **idtune**(ADM) or **configure**(ADM) commands should be used to modify or append the tunable parameter to the *stune* file.

In order for the new values to become effective, the UNIX system kernel must be rebuilt and the system must then be rebooted.

See also

configure(ADM), **idbuild**(ADM), **idtune**(ADM) **stune**(F)

mvdevice

video driver back end configuration file

Description

The */etc/conf/cf.d/mvdevice* file accomplishes configurability of video hardware by permitting the linking of back ends to the console video driver. This linking scheme includes a C library of video back ends for use with the link kit and separate driver entries for each of the back ends.

The configuration program uses the **mvdevice** file to produce a *space.c* for the console driver. This *space.c* includes the appropriate include files and extern references to the appropriate video back ends. In addition, the configuration program builds the console display switch within the *space.c*.

The *mvdevice* file has the following entries:

prefix — name of driver from 1 to 4 characters long (for example "mono"). This is the name of the video back end.

routine_mask — This mask tells which routines were supported by the particular back end. These routines are: **vvvinit()**, **vvvcmos()**, **vvvinitscreen()**, **vvvscroll()**, **vvvcopy()**, **vvvclear()**, **vvvpchar()**, **vvvscurs()**, **vvvsgr()**, **vvvioctl()**, **vvvadapctl()**.

type — This is placed in the file as a literal. For example, if the word **MONO** was put into the file, it would include the word **MONO** as the type entry of the adapter structure.

oem — OEM information treated exactly the same as the type (i.e. a literal).

paddr — The physical address at which the video RAM is located. This would allow a user to configure a future driver. Also included as a literal field.

size — The size of the video RAM. Also included as a literal field.

This information provides all the basic information needed for the program to generate an appropriate *space.c* and build the the correct adapter switch.

The routine mask uses the following bits to signify the following routines:

0x0001	**vvvinit()**
0x0002	**vvvcmos()**
0x0004	**vvvinitscreen()**
0x0008	**vvvscroll()**
0x0010	**vvvcopy()**
0x0020	**vvvclear()**
0x0040	**vvvpchar()**
0x0080	**vvvscurs()**
0x0100	**vvvsgr()**
0x0200	**vvvioctl()**
0x0400	**vvvadapctl()**

The default mvdevice file looks like this:

```
#
#       mvdevice:       video configuration master file.
#
#
#prefix name     routines        type    oem     paddr   size
mono    MONO    0x07fd          MONO    0       0       0
cga     CGA     0x07fd          CGA     0       0       0
ega     EGA     0x07ff          EGA     0       0       0
vga     VGA     0x07ff          VGA     0       0       0
```

See also

sdevice(F)

permissions

format of UUCP Permissions file

Description

The *Permissions* file (*/usr/lib/uucp/Permissions*) specifies the permissions for remote computers concerning login, file access, and command execution. In the *Permissions* file, you can specify the commands that a remote computer can execute and restrict its ability to request or receive files queued by the local site.

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name-value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a hash sign (#) and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of *Permissions* file entries:

LOGNAME specifies the permissions that take effect when a remote computer calls your computer.

MACHINE specifies permissions that take effect when your computer calls a remote computer.

Options

This section describes each option, specifies how they are used, and lists their default values.

REQUEST=yes | no

Specifies whether the remote computer can request to set up file transfers from your computer. When a remote computer calls your computer and requests to receive a file, this request can be granted or denied. **no** value is the default value. It will be used if the **REQUEST** option is not specified. The **REQUEST** option can appear in either a **LOGNAME** (remote calls you) entry or a **MACHINE** (you call remote) entry.

SENDFILES=yes | call
Specifies whether your computer can send the work queued for the remote computer. When a remote computer calls your computer and completes its work, it may attempt to take work your computer has queued for it. The **call** value is the default for the **SENDFILE** option. This option is only significant in **LOGNAME** entries since **MACHINE** entries apply when calls are made out to remote computers. If this option is used with a **MACHINE** entry, it will be ignored.

READ and **WRITE**
Specify the various parts of the file system that **uucico** can read from or write to. The **READ** and **WRITE** options can be used with either **MACHINE** or **LOGNAME** entries.

The default for both the **READ** and **WRITE** options is the *uucppublic* directory as shown in the following example:

```
READ=/usr/spool/uucppublic
WRITE=/usr/spool/uucppublic
```

Supplying " / " as a pathname gives permission to access any file that can be read by UUCP. Multiple entries must be separated by a colon. The **READ** option is for requesting files, and the **WRITE** option for depositing files. One of the values must be the prefix of any full path name of a file coming in or going out.

Note that the **READ** and **WRITE** options do not affect the actual permissions of a file or directory. You should be careful what directories you make accessible for reading and writing by remote systems.

NOREAD and **NOWRITE**
Specify exceptions to the **READ** and **WRITE** options or defaults. **NOWRITE** works in the same manner as the **NOREAD** option. **NOREAD** and **NOWRITE** can be used in both **LOGNAME** and **MACHINE** entries.

CALLBACK
Specifies in **LOGNAME** entries that no transaction will take place until the calling system is called back. There are two examples of when you would use **CALLBACK**. From a security standpoint, if you call back a machine you can be sure it is the machine it says it is. If you are doing long data transmissions, you can choose the machine that will be billed for the longer call. The default for the **COMMAND** option is **no**. The **CALLBACK** option is rarely used. If two sites have this option set for each other, a conversation will never get started.

COMMANDS
Specifies the commands in **MACHINE** entries that a remote computer can execute on your computer. This affects the security of your system; use it with extreme care.

The **uux** program will generate remote execution requests and queue them to be transferred to the remote computer. Files and a command are sent to the target computer for remote execution. Note that **COMMANDS** is not used in a **LOGNAME** entry; **COMMANDS** in **MACHINE** entries define command permissions whether you call the remote system or it calls you.

The default command that a remote computer can execute on your computer is **rmail**. If a command string is used in a **MACHINE** entry, the default commands are overridden. Full pathnames can also be used. Including the **ALL** value in the list means that any command from the remote computer specified in the entry will be executed. If you use this value, you give the remote computer full access to your computer. So, be careful; this allows far more access than normal users have. The **VALIDATE** option should be used with the **COMMANDS** option whenever potentially dangerous commands like **cat** and **uucp** are specified with the **COMMANDS** option. Any command that reads or writes files is potentially dangerous to local security when executed by the UUCP remote execution daemon (**uuxqt**).

VALIDATE
Used in conjunction with the **COMMANDS** option when specifying commands that are potentially dangerous to your computer's security. It provides a certain degree of verification of the caller's identity. The use of the **VALIDATE** option requires that privileged computers have a unique login/password for UUCP transactions. An important aspect of this validation is that the login/password associated with this entry be protected. If an outsider gets that information, that particular **VALIDATE** option can no longer be considered secure. (**VALIDATE** is merely an added level of security to the **COMMANDS** option, though it is a more secure way to open command access than **ALL**.)

Entries for OTHER systems

You may want to specify different option values for machines or logins that are not mentioned in specific **MACHINE** or **LOGNAME** entries. This may occur when there are many computers calling in that have the same set of permissions. The special name **OTHER** for the computer name can be used in a **MACHINE** or **LOGNAME** entry as follows:

```
MACHINE=OTHER \
COMMANDS=rmail:/usr/local/bin/lc

LOGNAME=OTHER \
REQUEST=yes SENDFILES=yes \
READ=/usr/spool/uucppublic \
WRITE=/usr/spool/uucppublic
```

All options that can be set for specific machines or logins can be used with the **OTHER** value, although the use of the **VALIDATE** option makes little sense.

Example

This entry is for public login. It provides the default permissions. Note that use of this type of anonymous login is not encouraged.

```
LOGNAME=nuucp \
MACHINE=OTHER \
READ=/usr/spool/uucppublic \
WRITE=/usr/spool/uucppublic \
SENDFILES=call REQUEST=no \
COMMANDS=/bin/rmail
```

See also

uucico(ADM), **uucp**(C), **uux**(C), **uuxqt**(C)

pkginfo

package characteristics file

Description

pkginfo is an ASCII file that describes the characteristics of the package along with information that helps control the flow of installation. It is created by the software package developer.

Each entry in the *pkginfo* file is a line that establishes the value of a parameter in the following form:

PARAM="*value*"

There is no required order in which the parameters must be specified within the file. Each parameter is described below. Only fields marked with an asterisk are mandatory.

PKG* Abbreviation for the package being installed, generally three characters in length (for example, **dir** or **pkg**). All characters in the abbreviation must be alphanumeric and the first may not be numeric. The abbreviation is limited to a maximum length of nine characters. **install**, **new**, and **all** are reserved abbreviations.

NAME* Text that specifies the package name (maximum length of 256 ASCII characters).

ARCH* A comma-separated list of alphanumeric tokens that indicate the architecture (for example, **i386**) associated with the package. The **pkgmk** tool may be used to create or modify this value when actually building the package. The maximum length of a token is 16 characters and it cannot include a comma.

VERSION* Text that specifies the current version associated with the software package. The maximum length is 256 ASCII characters and the first character cannot be a left parenthesis. The **pkgmk** tool may be used to create or modify this value when actually building the package.

CATEGORY* A comma-separated list of categories under which a package may be displayed. A package must at least belong to the system or application category. Categories are case-insensitive and may contain only alphanumerics. Each category is limited in length to 16 characters.

DESC Text that describes the package (maximum length of 256 ASCII characters).

VENDOR Used to identify the vendor that holds the software copyright (maximum length of 256 ASCII characters).

HOTLINE Phone number and/or mailing address where further information may be received or bugs may be reported (maximum length of 256 ASCII characters).

EMAIL An electronic address where further information is available or bugs may be reported (maximum length of 256 ASCII characters).

VSTOCK The vendor stock number, if any, that identifies this product (maximum length of 256 ASCII characters).

CLASSES A space-separated list of classes defined for a package. The order of the list determines the order in which the classes are installed. Classes listed first will be installed first (on a media by media basis). This parameter may be modified by the request script.

ISTATES A list of allowable run states for package installation (for example, "S s 1").

RSTATES A list of allowable run states for package removal (for example, "S s 1").

BASEDIR The pathname to a default directory where "relocatable" files may be installed. If blank, the package is not relocatable and any files that have relative pathnames will not be installed. The system administrator can override the default directory.

ULIMIT If set, this parameter is passed as an argument to the **ulimit** command, which establishes the maximum size of a file during installation.

ORDER A list of classes defining the order in which they should be put on the medium. Used by **pkgmk** in creating the package. Classes not defined in this field are placed on the medium using the standard ordering procedures.

MAXINST The maximum number of package instances that should be allowed on a machine at the same time. By default, only one instance of a package is allowed. This parameter must be set in order to have multiple instances of a package.

PSTAMP Production stamp used to mark the *pkgmap* file on the output volumes. Provides a means for distinguishing between production copies of a version if more than one is in use at a time. If **PSTAMP** is not defined, the default is used. The default consists of the UNIX system machine name followed by the string "***yymmddhhmm***" (year, month, day, hour, minute).

INTONLY Indicates that the package should only be installed interactively when set to any non-NULL value.

PREDEPEND Used to maintain compatibility with **installpkg**(ADM) package dependency checking. Pre-OAM dependency checks were based on whether or not the name file for the required package existed in the */usr/options* directory. This directory is not maintained for OAM packages since the *depend* file is used for checking dependencies. However, entries can be created in this directory to maintain compatibility. Setting the **PREDEPEND** parameter string to ***filename*** creates a */usr/options/filename.name* entry for the package. (Packages using the OAM system do not need to use this parameter.)

Example

Here is a sample *pkginfo*:

```
PKG="oam"
NAME="OAM Installation Utilities"
VERSION="3"
VENDOR="FOOBAR PRODUCTS"
HOTLINE="1-800-FOO-BUGS"
EMAIL="fonunix!bonuser"
VSTOCK="0122c3f5566"
CATEGORY="system.essential"
ISTATES="S 2"
RSTATES="S 2"
```

Note

Developers may define their own installation parameters by adding a definition to this file. A developer-defined parameter must begin with a capital letter, followed by lowercase letters.

pkgmap

package contents description file

Description

pkgmap is an ASCII file that provides a complete listing of the package contents. It is automatically generated by **pkgmk**(ADM) using the information in the *prototype*(F) file.

Each entry in *pkgmap* describes a single "deliverable object file." A deliverable object file includes shell scripts, executable objects, data files, directories, etc. The entry consists of several fields of information, each field separated by a space. The fields are described below and must appear in the order shown.

part — An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose the criteria for grouping files into a part (for example, based on class). If no value is defined in this field, part 1 is assumed.

ftype — A one-character field that indicates the file type. Valid values are:

- **f** a standard executable or data file
- **e** a file to be edited upon installation or removal
- **v** volatile file (one whose contents are expected to change)
- **d** directory
- **x** an exclusive directory
- **l** linked file
- **p** named pipe
- **c** character special device
- **b** block special device
- **i** installation script or information file
- **s** symbolic link

class — The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. It is not specified if the ftype is **i** (information file).

pathname The pathname where the object will reside on the target machine, such as */usr/bin/mail*. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable.

For linked files (ftype is either l or s), pathname must be in the form of ***path1=path2***, with ***path1*** specifying the destination of the link and ***path2*** specifying the source of the link.

For symbolically linked files, ***path2*** can be a relative pathname, such as ./ or ../. For example, if you enter a line such as

s /foo/bar/etc/mount=../etc/mount

path2 (*/foo/bar/etc/mount*) will be a symbolic link to *../etc/mount*.

pathname may contain variables which support relocation of the file. A "$" parameter may be embedded in the pathname structure. **$BASEDIR** can be used to identify the parent directories of the path hierarchy, making the entire package easily relocatable. Default values for parameter and **BASEDIR** must be supplied in the *pkginfo* file and may be overidden at installation.

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark "?" indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files, packaging information files or non-installable files.

owner The owner of the file (for example, *bin* or *root*). The field is limited to 14 characters in length. A question mark "?" indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what owner an installation script will be executed.

Can be a variable specification in the form of $[A-Z]. Will be resolved at installation time.

group The group to which the file belongs (for example, "bin" or "sys"). The field is limited to 14 characters in length. A question mark "?" indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what group an installation script will be executed.

Can be a variable assignment in the form of **$[A-Z]**. Will be resolved at installation time.

size — The actual size of the file in bytes. This field is not specified for named pipes, special devices, directories or linked files.

cksum — The checksum of the file contents. This field is not specified for named pipes, special devices, directories or linked files.

modtime — The time of last modification, as reported by the **stat**(S) function call. This field is not specified for named pipes, special devices, directories or linked files.

Each *pkgmap* must have one line that provides information about the number and maximum size (in 512-byte blocks) of parts that make up the package. This line is in the following format:

:number_of_parts maximum_part_size

Lines that begin with "#" are comment lines and are ignored.

When files are saved during installation before they are overwritten, they are normally just copied to a temporary pathname. However, for files whose mode includes execute permission (but which are not editable), the existing version is linked to a temporary pathname and the original file is removed. This allows processes which are executing during installation to be overwritten.

Example

The following is an example of a *pkgmap* file.

```
:2 500
1 i pkginfo 237 1179 541296672
1 b class1 /dev/diskette 17 134 0644 root other
1 c class1 /dev/rdiskette 17 134 0644 root other
1 d none bin 0755 root bin
1 f none bin/INSTALL 0755 root bin 11103 17954 541295535
1 f none bin/REMOVE 0755 root bin 3214 50237 541295541
1 l none bin/UNINSTALL=bin/REMOVE
1 f none bin/cmda 0755 root bin 3580 60325 541295567
1 f none bin/cmdb 0755 root bin 49107 51255 541438368
1 f class1 bin/cmdc 0755 root bin 45599 26048 541295599
1 f class1 bin/cmdd 0755 root bin 4648 8473 541461238
1 f none bin/cmde 0755 root bin 40501 1264 541295622
1 f class2 bin/cmdf 0755 root bin 2345 35889 541295574
1 f none bin/cmdg 0755 root bin 41185 47653 541461242
2 d class2 data 0755 root bin
2 p class1 data/apipe 0755 root other
2 d none log 0755 root bin
2 v none log/logfile 0755 root bin 41815 47563 541461333
2 d none save 0755 root bin
2 d none spool 0755 root bin
2 d none tmp 0755 root bin
```

Note

The *pkgmap* file may contain only one entry per unique pathname.

poll: Poll, Poll.hour, Poll.day

format of UUCP Poll files

Description

The *Poll* file (*/usr/lib/uucp/Poll*) contains information for polling remote computers. Each entry in the *Poll* file contains the name of a remote computer to call, followed by a tab character, and the hours the computer should be called. The hours must be integers in the range 0-23.

Poll file entries have the following format:

sysname⟨Tab⟩***hour*** ...

The following entry provides polling of computer *gorgon* every four hours:

```
gorgon  0 4 8 12 16 20
```

The *uudemon.poll* (see **uudemon**(ADM)) script uses the *Poll* file to set up the polling. Alternatively, *uudemon.poll2* uses the files *Poll.hour* and *Poll.day* to perform similar, but more precise functions. The format of these files is identical to *Poll*.

See also

cron(C), **crontab**(C), **uucico**(ADM), **uucp**(C), **uudemon**(ADM)

Standards conformance

poll is conformant with:

AT&T SVID Issue 2.

Poll.hour and *Poll.day* are an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

prototype

package information file

Description

prototype is an ASCII file used to specify package information. Each entry in the file describes a single deliverable object. An object may be a data file, directory, source file, executable object, etc. This file is generated by the package developer.

Entries in a **prototype** file consist of several fields of information separated by white space. Comment lines begin with a "#" and are ignored. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose criteria for grouping files into a part (for example, based on class). If this field is not used, part 1 is assumed.

ftype A one-character field which indicates the file type. Valid values are:

- **f** a standard executable or data file
- **e** a file to be edited upon installation or removal
- **v** volatile file (one whose contents are expected to change)
- **d** directory
- **x** an exclusive directory
- **l** linked file
- **p** named pipe
- **c** character special device
- **b** block special device
- **i** installation script or information file
- **s** symbolic link

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. The field is not specified for installation scripts. (**admin** and all classes beginning with capital letters are reserved class names.)

pathname The pathname where the file will reside on the target machine, for example, */usr/bin/mail* or */bin/ras_proc*. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable. The form

path1=path2

may be used for two purposes: to define a link and to define local pathnames.

For linked files, ***path1*** indicates the destination of the link and ***path2*** indicates the source file. (This format is mandatory for linked files.)

For symbolically linked files, ***path2*** can be a relative pathname, such as ./ or ../. For example, if you enter a line such as

s /foo/bar/etc/mount=../etc/mount

path2 (*/foo/bar/etc/mount*) will be a symbolic link to *../etc/mount*.

For local pathnames, ***path1*** indicates the pathname an object should have on the machine where the entry is to be installed and ***path2*** indicates either a relative or fixed pathname to a file on the host machine which contains the actual contents.

A pathname may contain a variable specification, which will be resolved at the time of installation. This specification should have the form **$[A-Z]**.

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (**?**) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

owner The owner of the file (for example, *bin* or *root*). The field is limited to 14 characters in length. A question mark (**?**) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of **$[A-Z]**. Will be resolved at installation time.

group The group to which the file belongs (for example, *bin* or *sys*). The field is limited to 14 characters in length. A question mark (?) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of **$[A-Z]**. Will be resolved at installation time.

An exclamation point (!) at the beginning of a line indicates that the line contains a command. These commands are used to incorporate files in other directories, to locate objects on a host machine, and to set permanent defaults. The following commands are available:

search Specifies a list of directories (separated by white space) to search for when looking for file contents on the host machine. The basename of the "path" field is appended to each directory in the ordered list until the file is located.

include Specifies a pathname which points to another **prototype** file to include. Note that **search** requests do not span "include" files.

default Specifies a list of attributes (mode, owner, and group) to be used by default if attribute information is not provided for prototype entries which require the information. The defaults do not apply to entries in "include" **prototype** files.

param=value Places the indicated parameter in the current environment.

The above commands may have variable substitutions embedded within them, as demonstrated in the two example **prototype** files below.

Before files are overwritten during installation, they are copied to a temporary pathname. The exception to this rule is files whose mode includes execute permission, unless the file is editable (that is, "ftype" is **e**). For files which meet this rule, the existing version is linked to a temporary pathname, and the original file is removed. This allows processes which are executing during installation to be overwritten.

Examples

Example 1:

```
!PROJDIR=/usr/proj
!BIN=$PROJDIR/bin
!CFG=$PROJDIR/cfg
!LIB=$PROJDIR/lib
!HDRS=$PROJDIR/hdrs
!search /usr/myname/usr/bin /usr/myname/src /usr/myname/hdrs
i pkginfo=/usr/myname/wrap/pkginfo
i depend=/usr/myname/wrap/depend
i version=/usr/myname/wrap/version
d none /usr/wrap 0755 root bin
d none /usr/wrap/usr/bin 0755 root bin
! search $BIN
f none /usr/wrap/bin/INSTALL 0755 root bin
f none /usr/wrap/bin/REMOVE 0755 root bin
f none /usr/wrap/bin/addpkg 0755 root bin
!default 755 root bin
f none /usr/wrap/bin/audit
f none /usr/wrap/bin/listpkg
f none /usr/wrap/bin/pkgmk
# The logfile starts as a zero length file, since the source
# file has zero length. Later, the size of logfile grows.
v none /usr/wrap/logfile=/usr/wrap/log/zero_length 0644 root bin
# the following specifies a link (dest=src)
l none /usr/wrap/src/addpkg=/usr/wrap/bin/rmpkg
! search $SRC
!default 644 root other
f src /usr/wrap/src/INSTALL.sh
f src /usr/wrap/src/REMOVE.sh
f src /usr/wrap/src/addpkg.c
f src /usr/wrap/src/audit.c
f src /usr/wrap/src/listpkg.c
f src /usr/wrap/src/pkgmk.c
d none /usr/wrap/data 0755 root bin
d none /usr/wrap/save 0755 root bin
d none /usr/wrap/spool 0755 root bin
d none /usr/wrap/tmp 0755 root bin
d src /usr/wrap/src 0755 root bin
```

Example 2:

```
# this prototype is generated by 'pkgproto' to refer
# to all prototypes in my src directory
!PROJDIR=/usr/dew/projx
!include $PROJDIR/src/cmd/prototype
!include $PROJDIR/src/cmd/audmerg/protofile
!include $PROJDIR/src/lib/proto
```

See also

pkginfo(F), **pkgmk**(ADM)

Notes

Normally, if a file is defined in the **prototype** file but does not exist, that file is created at the time of package installation. However, if the file pathname includes a directory that does not exist, the file will not be created. For example, if the **prototype** file has the following entry:

```
f none /usr/dev/bin/command
```

and that file does not exist, it will be created if the directory */usr/dev/bin* already exists or if the **prototype** also has an entry defining the directory:

```
d none /usr/dev/bin
```

purge

the policy file of the sanitization utility purge

Description

/etc/default/purge is an ASCII file whose lines each designate a file, filesystem or device to be a member of a ***type***. The command:

purge -t *type*

would overwrite all the members of ***type***.

Blank lines and lines beginning with "#" are ignored. Entries are of the form:

file type [***count***]

This specifies that ***file*** is a member of ***type***. The optional field ***count*** is the number of times to overwrite ***file*** when it is purged. The default is one.

The two ***types*** *system* and *user* are hardwired into the **purge**(C) utility. These types can be overwritten with the **-s** and **-u** switches to **purge** respectively.

This file should be configured on site to reflect files and devices which are sensitive and need to be protected from unauthorized access.

The initial contents of the file are:

/tmp	system
/user	user

File

/etc/default/purge

See also

purge(C), **sysadmsh**(ADM)

Value added

purge is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

queue

MMDF queue files for storing mail in transit

Description

MMDF stores mail in */usr/spool/mmdf/lock/home*, an isolated part of the filesystem, so that only authorized software may access the mail. Mail is stored under the directory sub-tree.

It must specify a path with at least two sub-directories. The next-to-bottom one is used as a "locking" directory and the bottom one is the "home". For full protection, only authorized software can move through the locking directory. Hence, it is owned by MMDF and accessible only to it.

Queue entries

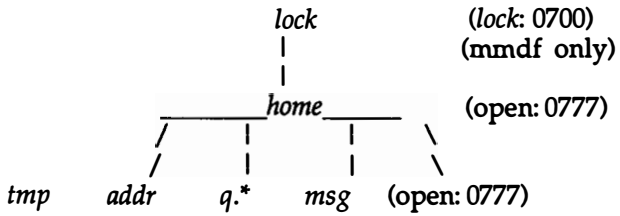
When mail is queued by **submit**, it is actually stored as two files. One contains the actual message text and the other contains some control information and the list of addressees.

The text file is stored in the *msg* directory. The other file is stored in the *addr* directory and is linked into one or more queue directories. The queue directories are selected based on the channels on which this message will be delivered. Each output channel typically has its own queue directory.

Another directory below *home* is a temporary area called *tmp*. It holds temporary address-lists as they are being built. Queuing of a message is completed by linking this address file into *addr* and the queue directories. The *msg* directory contains files with message text. *Addr* and *msg* files are synchronized by giving them the same unique name, which MMDF occasionally calls the message "name". The message name is derived by use of **mktemp**(S), using *msg* as the base directory. The files created in *addr* must have open read-write access; the ones in *msg* must have open read access.

When **submit** runs, it changes into *home* for its working directory. It then does a **setuid** to run as the caller. This is necessary to permit **submit** to access the caller's address-list files (specified as "< file"). **Deliver** changes into the queue directory to minimize the time spent searching for messages to deliver.

The following depicts the directory organization:



addresses ==> moved and linked message text
built here ==> into here into here put here

entries: 0666 0666 0644

Queue file formats

The *msg* portion of a queued message simply contains the message, which must conform to the Arpanet standard syntax, specified in RFC822. It is expected that the format of the message contents file eventually will be more structured, permitting storage of multi-media mail.

The following specifies the syntax of the *addr* (and queue directory) address-list portion of the queued message:

Address file contents

file ::= creation late flags '\ n' [rtrn-addr] '\ n' *(adr_queue '\ n')

creation ::= {long integer decimal representation of the time the message was created}

late ::= ADR_MAIL / ADR_DONE {from adr_queue.h}

flags ::= {decimal representation of 16-bits of flags}

rtrn-addr ::= {rfc822 return address}

adr_queue ::= temp_ok mode queue host local {conforms to structure specified in adr_queue.h}

temp_ok ::= {temporary flag indicating whether this address has been verified by the receiving host: "yes" is "+"; "not yet" is "-"}

mode ::= {send to mailbox(m), tty(t), both(b), either(e), or processing completed(*)}

queue ::= {name of the queue into which this message is linked for this address}

host ::= {official name (and domain) of recipient host}

local ::= {local address on receiving host}

Address file description

An address queue file contains a creation time-stamp, an indication if the sender has been notified of delayed delivery, some flags, an optional return-mail address, and an address list. Several <flags> are currently in use. 0004 indicates whether late warnings should be sent to the return-mail address if the entire address list has not been processed within the number of hours specified by "warntime". 0002 indicates whether mail should be returned to the sender if it hasn't been completely processed within the number of hours specified by "failtime". 0001 indicates whether warning and failure messages are to contain only a citation of the message. An " * " value, for the "late" flag, indicates that a warning notice has been sent.

The creation date is coded as a long ASCII decimal string, terminated by the "late" flag. This has to be inserted into the file because UNIX System V/386 doesn't maintain it. The date is used to sort the queue so that mail is delivered in the order submitted.

The return address is a line of text and may be any address acceptable to **submit.**

Each address entry is on a separate line, and conforms to the adr_struct format, defined in *adr_queue.h.* It contains several fields separated by spaces or commas. Fields containing spaces or commas must be enclosed in double quotes.

The temp_ok flag indicates whether the address has been accepted during an "address verification" dialog with the receiving host. When the message text is successfully accepted by the receiving host, then this flag no longer applies and the mode is set to ADR_DONE ("*"). Before final acceptance of message text, the mode flag indicates whether the mail is for a mailbox, terminal, both, or either. (Currently only mailbox delivery, ADR_MAIL, is used.)

The queue name is the name of the sub-queue in which the message is queued for this address. Each addressee's host may be on a separate queue or some hosts may share the same queue. When all addressees in the same queue have been delivered, the address file is removed from that queue directory. When all queues have been processed, the address file (in both *addr* and the queue directory) and the text file (in *msg*) are removed.

The host and local strings are used by the channel program. The host determines the type of connection the channel program makes. The local string is passed to the host; it should be something meaningful to that host. The local string must not contain newline or null and it must be a valid local address per RFC822.

See also

deliver(ADM), **cleanque**(ADM), **submit**(ADM)

Credit

MMDF was developed at the University of Delaware and is used with permission.

queuedefs

scheduling information for cron queues

Description

The **queuedefs** file is read by the clock daemon, **cron**, and controls how jobs submitted with **at**, **batch**, and **crontab** are executed. Every job submitted by one of these programs is placed in a certain queue, and the behavior of these queues is defined in */usr/lib/cron/queuedefs*. Queues are designated by a single, lower-case letter. The following queues have special significance:

a	**at** queue
b	**batch** queue
c	**cron** queue

For a given queue, the **queuedefs** file specifies the maximum number of jobs that may be executing at one time (**njobs**), the priority at which jobs will execute (**nice**), and the how long **cron** will wait between attempts to run a job (**wait**). If **njobs** jobs are already running in a given queue when a new job is scheduled to begin execution, **cron** will reschedule the job to execute **wait** seconds later. A typical file might look like this:

```
a.4j1n
b.2j2n90w
```

Each line gives parameters for one queue. The line must begin with a letter designating a queue, followed by a period (.). This is followed by the numeric values for **njobs**, **nice**, and **wait**, followed by the letters *j*, *n*, and *w* respectively. The values must appear in this order, although a value and its corresponding letter may be omitted entirely, in which case a default value is used. The default values are **njobs** = 100, **nice** = 2, and **wait** = 60.

The value for **nice** is added to the default priority of the job (a higher numerical priority results in a lower scheduling priority — see **nice**(C)). **wait** is given in seconds.

File

/usr/lib/cron/queuedefs	*queuedefs* file

sdevice

local device configuration file

Description

Files in the directory */etc/conf/sdevice.d* contain local system configuration information for each of the devices specified in the *mdevice* file. Each file contains one or more entries for a device specified in *mdevice*(F). Every time a kernel is built using **idbuild**(ADM), these files are coalesced into the single file */etc/conf/cf.d/sdevice*. Files in */etc/conf/sdevice.d* are the *System* file components either delivered with the base system or installed later via **idinstall**.

Each entry must contain the following whitespace-separated fields:

1. "Device name": this field contains the internal name of the driver. This must match one of the names in the first field of an *mdevice* file entry.

2. "Configure": this field must contain the character "Y" indicating that the device is to be installed in the kernel. For testing purposes, an "N" may be entered indicating that the device will not be installed.

3. "Unit": this field can be encoded with a device dependent numeric value. It is usually used to represent the number of subdevices on a controller or pseudo-device. Its value must be within the minimum and maximum values specified in fields 7 and 8 of the *mdevice* entry.

4. "IPL": the Interrupt Priority Level field specifies the System Priority Level (SPL) at which the driver's interrupt handler will run in the new system kernel. Legal values are 1 through 7. If the driver does not have an interrupt handling routine, put a 0 in this field. All of a driver's *sdevice* entries must specify the same IPL.

5. "Type": this field indicates the type of interrupt scheme required by the device. The permissible values are:

   0 The device does not require an interrupt vector.

   1 The device requires an interrupt vector. If the driver supports more than one hardware controller, each controller requires a separate vector.

   2 The device requires an interrupt vector. If the driver supports more than one hardware controller, each controller will share the same vector.

   3 The device requires an interrupt vector. If the driver supports more than one hardware controller, each controller will share the same interrupt vector. Multiple device drivers having the same IPL can share this interrupt.

4 The device requires an interrupt vector. If the driver supports more than one hardware controller, each of those controllers may either share an interrupt vector with one of the other controllers or use a separate interrupt vector. Multiple device drivers having the same IPL can share this interrupt.

5 The device requires an interrupt vector. If the device supports more than one hardware controller, each controller will share the same interrupt. This interrupt type is only used by hard disk drivers to prevent other devices from sharing the same interrupt vector.

6. "Vector": this field contains the interrupt vector number used by the device. If the "Type" field contains a 0 (that is, no interrupt required), this field is ignored. Note that more than one device can share an interrupt number provided that both devices are type 3, 4 or 5.

7. "SIOA": the Start I/O Address field contains the starting address on the I/O bus through which the device communicates. This field must be between 0x0 and 0xFFFF inclusive. (If this field is not used, it should be 0.)

8. "EIOA": the End I/O Address field contains the end address on the I/O bus through which the device communicates. This field must be within 0x0 and 0xFFFF inclusive. (If this field is not used, it should be 0.) The SIOA cannot be larger than the EIOA.

9. "SCMA": the Start Controller Memory Address field is used by controllers that have internal memory. It specifies the starting address of this memory. This field must be at least 0x10000. (If this field is not used, it should be 0.)

10. "ECMA": the End Controller Memory Address field specifies the end of the internal memory for the device. This field must be at least 0x1000. (If this field is not used, it should be 0.) The SCMA cannot be larger than the ECMA.

See also

idbuild(ADM), **idinstall**(ADM), **mdevice**(F)

Notes

Some I/O Address ranges are reserved and cannot be used, including:

SIOA	EIOA
0x00	0x0F
0x20	0x21
0x40	0x43
0x63	0x63
0x70	0x7F
0x80	0x83
0xA0	0xA7

space

disk space requirement file

Description

space is an ASCII file that gives information about disk space requirements for the target environment. It defines space needed beyond that which is used by objects defined in the **prototype** file—for example, files which will be installed with the **installf** command. It should define the maximum amount of additional space which a package will require.

The generic format of a line in this file is:

pathname blocks inodes

Definitions for fields are as follows:

pathname Specifies a directory name which may or may not be the mount point for a filesystem. Names that do not begin with a slash (/) indicate relocatable directories.

blocks Defines the number of disk blocks required for installation of the files and directory entries contained in the pathname (using a 512-byte block size).

inodes Defines the number of inodes required for installation of the files and directory entries contained in the pathname.

Example

```
# extra space required by config data which is
# dynamically loaded onto the system
data    500     1
```

See also

installf(ADM), **prototype**(F)

stune

local tunable parameter file

Description

/etc/conf/cf.d/stune contains local system settings for tunable parameters. The parameter settings in this file replace the default values specified in the *mtune* file, if the new values are within the legal range for the parameter specified in *mtune*. The file contains one line for each parameter to be reset. Each line contains two whitespace-separated fields:

external name — This is the external name of the tunable parameter used in the *mtune* file.

value — This field contains the new value for the tunable parameter.

The file *stune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *stune* or *mtune* files. Instead the **idtune**(ADM) or **configure**(ADM) commands should be used.

In order for the new values to become effective the UNIX kernel must be rebuilt and the system must then be rebooted.

See also

configure(ADM), **idbuild**(ADM), **idtune**(ADM), **mtune**(F)

sysadmcolor

colors

Description

The files **$HOME**/*.sysadmcolor* and */usr/lib/sysadm/sysadmcolor* describe what colors will be used within each window of **sysadmsh**(ADM). This feature will only come into effect on terminals which support color, and have the necessary information in their respective **terminfo**(M) definitions. When **sysadmsh**(ADM) is started, it will first try to read the file *.sysadmcolor* in the user's home directory. If that file does not exist, it will try to read the file */usr/lib/sysadm/sysadmcolor*. If neither file exists, a default color table stored within **sysadmsh**(ADM) is used.

Each line in a **sysadmcolor** file must be in one of the following formats:

Comment line
: The first character must be a "#" (number sign); the rest of the line is taken to be a comment and ignored.

Blank line
: Any lines containing just tabs and spaces will be ignored.

Window Color
: This line must have three fields, which can be separated by spaces or tabs;

  Sysadmsh.*window*: ***foreground*** ***background***

  The ***window*** field is a label for one of the various windows that **sysadmsh**(ADM) can display, chosen from the following;

  menu_window which contains the menu bar, located on the second line of the display.

  desc_window which contains the description of the currently available menu selections, located on the third line of the display.

  context_window which contains the name of the last selected menu entry, located in the top right of the display.

  date_window which contains the current working directory and date, located on the fourth line of the display.

  mode_window which displays those editing modes which are in effect, located in the middle of the top line of the display.

error_window in which error messages display, located on the bottom line of the display.

point_window which is used for pick-and-point lists.

form_window in which forms are displayed.

scan_window which is used to display scanned listings of data.

edit_window which is used to edit text, usually opened over a field the user has selected for editing.

The ***foreground*** and ***background*** fields contain the code numbers for the foreground and background colors used in that window. The numbers for each color are listed below;

Code	Color
0	Black
1	Red
2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White

Note that if the file does not contain a line for a particular window construct, the default colors from the internal color table will be used.

Examples

Below is an example file, which shows the default colors used by **sysadmsh**(ADM).

```
#
# The standard colors used by sysadmsh(ADM).
#
Sysadmsh.menu_window:          7       0
Sysadmsh.desc_window:          7       0
Sysadmsh.context_window:       7       0
Sysadmsh.date_window:          7       0
Sysadmsh.mode_window:          7       0
Sysadmsh.error_window:         7       0
Sysadmsh.point_window:         7       0
Sysadmsh.form_window:          7       0
Sysadmsh.prompt_window:        7       0
Sysadmsh.scan_window:          7       0
Sysadmsh.edit_window:          7       0
```

See also

sysadmsh(ADM), **terminfo**(F), **terminfo**(M)

Notes

If the environment variable **$SYSADM** is set to other than the directory */usr/lib/sysadm*, then this is where the system-wide version of the **sysadmcolor** file should be found by **sysadmsh**(ADM).

Value added

sysadmcolor is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

sysadmmenu

layout of extensible menus in sysadmsh(ADM)

Description

These files each describe the layout of one extensible menu area in **sysadmsh**(ADM), as shown in the following table;

File	Extensible Menu Area
$HOME/.sysadmmenu	User
/usr/lib/sysadm/.menu-execute	System ➪ Execute
/usr/lib/sysadm/.menu-hardware	System ➪ Hardware
/usr/lib/sysadm/.menu-kernel	System ➪ Configure ➪ Kernel
/usr/lib/sysadm/.menu-network	System ➪ Configure ➪ Network
/usr/lib/sysadm/.menu-other	System ➪ Configure ➪ Other

The file *.sysadmmenu* in the user's home directory entry need not exist, in which case the **User** menu selection in **sysadmsh**(ADM) will have no visible effect.

These files contain lines which describe the menu layout, and the name, description, and arguments of the program that should be run when each menu item is selected. The files should only be edited using the System ➪ Configure ➪ Menu selection in **sysadmsh**(ADM), and the **menumerge**(ADM) utility. Note that only the system adminstrator will have suitable permissions to configure the last five menu areas.

See also

menumerge(ADM), **sysadmsh**(ADM)

Notes

If the environment variable **$SYSADM** is set to other than the directory */usr/lib/sysadm*, then this is where the files *.menu-execute*, *.menu-hardware*, *.menu-kernel*, *.menu-network*, and *.menu-other* should be found.

Value added

sysadmmenu is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

sysfiles

format of UUCP Sysfiles file

Description

The */usr/lib/uucp/Sysfiles* file lets you assign different files to be used by **uucp**(C) and **cu**(C) as *Systems*, *Devices*, and *Dialers* files.

You can use different *Systems* files so that requests for login services can be made to other addresses than UUCP services.

With different *Dialers* files you can use different handshaking for **cu** and **uucp**. Multiple *Systems*, *Dialers*, and *Devices* files are useful if any one file becomes too large.

An active *Sysfiles* file is not included in the distribution. Instead a *Sysfiles.eg* file is included, which contains comments and commented examples of how such a file can be used. This is done because UUCP runs faster without reading this file.

The format of the *Sysfiles* file is

service=*w* systems=*x:x* dialers=*y:y* devices=*z:z*

where *w* is replaced by **uucico**(ADM), **cu**, or both separated by a colon; *x* is one or more files to be used as the *Systems* file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the *Dialers* file; and *z* is one or more files to be used as the *Devices* file. Each file is assumed to be relative to the */usr/lib/uucp* directory, unless a full path is given. A backslash-carriage return (\⟨Return⟩) can be used to continue an entry onto the next line.

An example of using a local *Systems* file in addition to the usual *Systems* file follows:

```
service=uucico:cu  systems=Systems:Local_Systems
```

If this is in */usr/lib/uucp/Sysfiles*, then both **uucico** and **cu** will first look in */usr/lib/uucp/Systems*. If the system they are trying to call does not have an entry in that file, or if the entries in the file fail, then they will look in */usr/lib/uucp/Local_Systems*.

When different *Systems* files are defined for **uucico** and **cu** services, your machine will store two different lists of Systems. You can print the *uucico* list using the **uuname** command or the **cu** list using the **uuname -c** command.

Examples

The following example uses different *Systems* and *Dialers* files to separate the **uucico** and **cu**-specific info, with information that they use in common still in the "usual" *Systems* and *Dialers* files.

```
service=uucico  systems=Systems.cico:Systems \
                dialers=Dialers.cico:Dialers
service=cu      systems=Systems.cu:Systems \
                dialers=Dialers.cu:Dialers
```

This next example uses the same systems files for **uucico** and **cu**, but has split the *Systems* file into local, company-wide, and global files.

```
service=uucico  systems=Systems.local:Systems.company:Systems
service=cu      systems=Systems.local:Systems.company:Systems
```

See also

systems(F), **uucico**(ADM), **uucp**(C)

systemid

the Micnet system identification file

Description

The *systemid* file contains the machine and site names for a system in a Micnet network. A machine name identifies a system and distinguishes it from other systems in the same network. A site name identifies the network to which a system belongs and distinguishes the network from other networks in the same chain.

The *systemid* file may contain a site name and up to four different machine names. The file has the form:

[*site-name*]
[*machine-name1*]
[*machine-name2*]
[*machine-name3*]
[*machine-name4*]

The file must contain at least one machine name. The other machine names are optional, serving as alternate names for the same machine. The file must contain a site name if more than one machine name is given or if the network is connected to another through a UUCP link. The site name, when given, must be on the first line.

Each name can have up to eight letters and numbers but must always begin with a letter. There is never more than one name to a line. A line beginning with a number sign (#) is considered a comment line and is ignored.

The Micnet network requires one *systemid* file on each system in a network with each file containing a unique set of machine names. If the network is connected to another network through a UUCP link, each file in the network must contain the same site name.

The *systemid* file is used primarily during resolution of aliases. When aliases contain site and/or machine names, the name is compared with the names in the file and removed if there is a match. If there is no match, the alias (and associated message, file, or command) is passed on to the specified site or machine for further processing.

File

/etc/systemid

See also

aliases(M), **netutil**(ADM), **top**(F)

Value added

systemid is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

systems

format of UUCP Systems file

Description

The *Systems* file (*/usr/lib/uucp/Systems*) contains the information needed by the **uucico** daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that your computer can call. You can configure the *Systems* file to prevent unauthorized computers from logging in on your computer. More than one entry may be present for a particular computer. These additional entries represent alternative communication paths which the computer tries in sequential order.

Each entry in the *Systems* file has the following format:

sitename schedule device speed phone login-script

sitename	the node name of the remote computer.
schedule	a string that indicates the day-of-week and time-of-day when the remote computer can be called.
device	the device type that should be used to establish the communication link to the remote computer.
speed	the transfer speed of the device used in establishing the communication link.
phone	the phone number of the remote computer for automatic dialers. If you wish to create a portable *Systems* file that can be used at a number of sites where the dialing prefixes differ, see the **dialcodes**(F) man page.
login-script	login information (also known as a "chat script").

See also

devices(F), **dialers**(F), **uucico**(ADM), **uucp**(C)

tables

MMDF name tables for aliases, domains, and hosts

Description

All of the MMDF name tables are encoded into a database which is built on top of the **dbm**(S) package. A number of tables are associated with MMDF, the exact set being specified by the tailor file, */usr/mmdf/mmdftailor*. Name tables all have the same format. Functionally, they permit a simple key/value pairing. The syntax for tables is specified here:

entries ::=	entries entry
entry ::=	comment / real-entry
comment ::=	'#' value eol
real-entry ::=	name separator value eol
name ::=	{string of chars not containing a <separator>}
separator ::=	{':' and whitespace}
value ::=	{string of chars not containing an <eol>}
eol ::=	{newline, nullch, DEL}

where:

name is	a key
value is	any relevant text.

Hosts and domains

Two basic types of table are host and domain tables. This section gives a brief discussion of these concepts in terms of the MMDF system. The domain namespace is treated as a logical global hierarchy, according to the model of RFC 819, with subdomains separated by (.) (for example, ISI.USC.ARPA is a three level hierarchy with ARPA at the top level). A host is a computer associated with a channel which may be directly connected or reached through a relay associated with the channel. The distinction between hosts as physical entities, and domains as logical entities should be noted. All hosts known to an MMDF system must have unique names. For this reason, the convention of labeling hosts by an associated domain name is adopted in many cases. This is a useful method to guarantee unique names, but is not required.

The domain and host table structures are devised with three basic aims in mind:

1. To map a string into a fully expanded domain name.
2. To map this domain into a source route starting with a host.
3. To obtain the transport address associated with the host.

Domain tables

Domains are split in a two-level manner, with the top part of the tree specified in the tailor file and the lower parts of the tree in tables. The two-level structure is intended as a balance between generality and efficiency. The order of searching is also specified in the tailor file. The structure of a domain table is to have **name** as the part of the domain not in the tailor file. Thus for ISI.USC.ARPA there might be a domain ARPA with name=isi.usc or domain USC.ARPA with name=isi. The structure of value is:

value ::= *(domain dm_separator) host

The possible values of **dm_separator** are given in **tai**(S), although typically (,) or ( ) would be used. This value is essentially a source route to be traversed from right to left. Consider an example table for the domain ARPA:

```
# Sample ARPA domain table
isi.usc:a.isi.usc.arpa
b.isi.usc:b.isi.usc.arpa
foobar.isi.usc:b.isi.usc.arpa
graphics.isi.usc:graphics.isi.usc.arpa z.mit.arpa
```

Thus, if the "isi.usc.arpa" is analyzed, domain table ARPA will be selected, and host "a.isi.usc.arpa" associated with domain "isi.usc.arpa." If only "isi.usc" were given, the domain tables would be searched in order, and if the ARPA table were the first one to give a match, then the same result would be reached. If "foobar.isi.usc" is given, it would be mapped to host "b.isi.usc.arpa" and (official) domain "b.isi.usc.arpa." If "graphics.isi.usc.arpa" is given, a source route to domain "graphics.isi.usc.arpa" through HOST "z.mit.arpa" will be identified. If "xy.isi.usc.arpa" (or "xy.isi.usc") is given, then it will not be found. However, a subdomain will be stripped from the left and the search repeated. Thus domain "xy.isi.usc.arpa" will be identified as reached by a source route through host "a.isi.usc.arpa."

As specified earlier, the order of searching is also specified in the tailor file. For example, a host in domain UCL-CS.AC.UK, might have a search order UCL-CS.AC.UK, AC.UK, UK, SWEDEN, ARPA, "". Thus, if there were a domain isi.usc.ac.uk, it would be the preferred mapping for isi.usc over isi.usc.arpa. The last domain searched is null. This could be used to contain random fully qualified domains or to identify gateways to other domains. An example file is:

```
#  Sample Top level domain table
#  Odd host
basservax.australia:basservax.australia scunthorpe.ac.uk
# UUCP Gateway
uucp:seismo.arpa
# Mailnet Gateway  (-> multics -> educom ->mailnet)
mailnet:educom.mailnet mit-multics.arpa
```

To specify the top domain in the tailor file, the name and dmn parameters of the domain should be set to "".

Host tables

For every host associated with the channel, there will be one or more entries. In each case, the key is the name identified from the domain tables. A host may have multiple entries if it has more than one transport address which the channel might utilize.

When a channel just sends all its mail to a relaying site, the address portion (value) of the entry is not needed, directly, during the transmission process. Hence, it need not be accurate. However, it is still used to logically collect together host names, that is, all table entries with the same value are regarded as being aliases for the same host.

P.O. box channels

POBox channels, for passive, telephone-based exchange, operate in two modes. In the first mode, a single login is authorized to pickup all mail for the channel. In this case, the host-table addresses are only used for the "collecting" function. For the second mode, different logins share the channel and are to receive only some of the mail queued for the channel. In this case, the login is treated as an "address", and the table entries should have the value fields contain the name of the login authorized to pickup mail for that "host".

Access control tables

Channels also have access control tables associated with them, to determine whether a message is allowed to use a given route. Each channel has four (optional) tables that determine the access controls used: insrc, outsrc, indest, and outdest.

Reformatting tables

There may also be a "known hosts" table associated with each channel. This is exactly the same format as a host table. If a message is being reformatted, and if an address does not have its host in this list, then it will be modified to appear as a percent route (RFC733 or JNT Mail route) address, with the local domain as the root.

Local aliases

The password file specifies the mailing names are login names of all local recipients. Since this is a rather restricted name space, and since it is useful to have some other kinds of locally-known names, there is a second file used to specify "aliases". The location of the aliases file is specified in the tailor file.

An alias entry may be used for one of five functions:

1. True aliasing, where the key value maps to a local user's login name, for example "dave:dcrocker;"

2. Forwarding, where the key value maps to a foreign address, such as "dcrocker:dcrocker@udel;" and

3. Address lists, where the key value maps to a set of addresses, such as "mother:cotton,dcrocker,farber."

4. Redirection of a message to a file: for example, "foobar:dpk/foobar" would cause user and group ids to be set to dpk and the text of the message to be appended to the file "foobar" in dpk's default login directory. Similarly, "foobar:dpk//tmp/foobar" would do the same for file */tmp/foobar*.

5. Redirection of a message to a pipe. For example, "news-inject:news | /usr/lib/news/uurec" would cause a message to be passed into a UNIX pipe (see **pipe**(S)) with userid and groupid set to **news**.

As a convenience, the value-part of an entry may specify a file name, so that the **actual** value is taken from the file. There are two possible notations for this:

1. By having left-angle bracket (<) precede the value specification. For example: "mother: < /etc/mmdf/mother_list@udel-relay.arpa."

2. By using a data type with value "include." For example: "mother: :include: /etc/mmdf/mother@udel-relay.arpa"

In both cases, the @HOST (not a domain) is optional. If specified, it should be the local host.

Recursive specification is permitted. For example, "crocker" may map to "dcrocker" and "dcrocker" may map to "dcrocker at udel," so that both "crocker" and "dcrocker" are locally-known names, but mail sent to either of them will be forwarded to "dcrocker@udel."

In practice, it is useful to organize alias files into the following order:

List aliases — which contain a value referring to a later address list. This constitutes a one-to-one mapping of a key to a value, where the value points into the "Lists" group.

Lists — which contain values referring to multiple addresses: this constitutes a one-to-many mapping, where some of the addresses may refer to other entries (address lists) in the Lists group, as well as other entries (individual addresses) later in the table.

Mailbox aliases
: which contain values referring to single addresses. These constitute one-to-one mappings, where the value refers to an entry in the password file or to an entry in the "Forwarding aliases" group.

Forwarding aliases
: which contain values referring to single addresses on other machines. These, also, are one-to-one mappings, where the value always refers to an off-machine address.

By organizing the file in this manner, only the "Lists" portion requires a topological sort. Since the other three sections will never point to entries within their section, they may be sorted more conveniently, such as alphabetically. Such a structure also tends to make changes easy. In particular, the handling of forwarding is easy, since **all** references to a user will get intercepted, at the end of the table.

Mail-ID tables

The Mail-ID tables are used only if the Mail-IDs feature is enabled. This can be done in the tailoring file, by defining MMAILID to be 1. Mail-IDs are used to disassociate mail addresses from login names. There are two tables that are used to map Mail-IDs to users' login names and login ids to Mail-IDs. The "users" file is used to map users (login ids) to Mail-IDs, and the "mailids" file is used to map Mail-IDs to users. The names of these files can be overridden, but it is not recommended. Each file has lines with two entries per line (user and Mail-ID, or Mail-ID and user).

A user can have more than one entry in the Mail-IDs file, but should have only one entry in the users file. This does not prevent them from sending mail with any of their Mail-IDs. The users file is just a source of default Mail-IDs.

Value added

tables is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Credit

MMDF was developed at the University of Delaware and is used with permission.

tar

archive format

Description

The command **tar**(C) dumps files to, and extracts files from, backup media or the hard disk.

Each file is archived in contiguous blocks. The first block is occupied by a header, whose format is given below, and the subsequent blocks of the files occupy the following blocks. All headers and file data start on 512-byte block boundaries and any extra unused space is padded with garbage. The format of a header block is as follows:

```
#define TBLOCK 512
#define NBLOCK 20
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
                char extno[4];
                char extotal[4];
                char efsize[12];
        } dbuf;
} dblock;
```

The `name` entry is the pathname of the file when archived. If the pathname starts with a zero word, the entry is empty. It is a maximum of 100 bytes long and ends in a null byte. `mode`, `uid`, `gid`, `size`, and `time` modified are the same as described under i-nodes (refer to **filesystem**(FP)). The checksum entry has a value such that the sum of the words of the directory entry is zero.

If the entry corresponds to a link, then `linkname` contains the pathname of the file to which this entry is linked; `linkflag` is set to 0 if there are no links, or 1 if there are links. No data is put in the archive file.

See also

filesystem(FP), **tar**(C)

Standards conformance

tar is conformant with:

AT&T SVID Issue 2.

term

terminal driving tables for nroff

Description

nroff(CT) uses driving tables to customize its output for various types of output devices, such as printing terminals, special word-processing printers (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as C programs, compiled, and installed in */usr/lib/term/tab**name***, where ***name*** is the name for that terminal type, as shown in **term**(CT).

The structure of the tables is as follows. Sizes are in 240ths of an inch.

```
#define INCH    240
#include /usr/lib/term/terms.h

struct termtable tlp ; { \* lp is the name of the term, *\
        int bset;        \* modify with new name, such as tnew *\
        int breset;
        int Hor;
        int Vert;
        int Newline;
        int Char;
        int Em;
        int Halfline;
        int Adj;
        char *twinit;
        char *twrest;
        char *twnl;
        char *hlr;
        char *hlf;
        char *flr;
        char *bdon;
        char *bdoff;
        char *iton;
        char *itoff;
        char *ploton;
        char *plotoff;
        char *up;
        char *down;
        char *right;
        char *left;
        char *codetab[256-32];
        char *zzz;
} ;
```

The meanings of the various fields are as follows:

`bset` bits to set in `termio.c_oflag` (see **tty**(M) and **termio**(M)) after output

`breset` bits to reset in `termio.c_oflag` before output

`Hor` horizontal resolution in fractions of an inch

`Vert` vertical resolution in fractions of an inch

`Newline` space moved by a newline (linefeed) character in fractions of an inch.

`Char` quantum of character sizes, in fractions of an inch. (that is, characters are multiples of `Char` units wide. See `codetab` below.)

`Em` size of an em in fractions of an inch

`Halfline` space moved by a half-linefeed (or half-reverse-linefeed) character in fractions of an inch

`Adj` quantum of white space for margin adjustment in the absence of the **-e** option, in fractions of an inch. (that is, white spaces are a multiple of `Adj` units wide)

Note: if this is less than the size of the space character (in units of `Char`; see below for how the sizes of characters are defined), **nroff** will output fractional spaces using plot mode. Also, if the **-e** switch to **nroff** is used, `Adj` is set equal to `Hor` by **nroff**

`twinit` set of characters used to initialize the terminal in a mode suitable for **nroff**

`twrest` set of characters used to restore the terminal to normal mode

`twnl` set of characters used to move down one line

`hlr` set of characters used to move up one-half line

`hlf` set of characters used to move down one-half line

`flr` set of characters used to move up one line

`bdon` set of characters used to turn on hardware boldface mode, if any. **nroff** assumes that boldface mode is reset automatically by the `twnl` string, because many letter-quality printers reset the boldface mode when they receive a carriage return; the `twnl` string should include whatever characters are necessary to reset the boldface mode

`bdoff` set of characters used to turn off hardware boldface mode, if any

`iton` set of characters used to turn on hardware italics mode, if any

`itoff` set of characters used to turn off hardware italics mode, if any

`ploton` set of characters used to turn on hardware plot mode (for Diablo-type mechanisms), if any

`plotoff` set of characters used to turn off hardware plot mode (for Diablo-type mechanisms), if any

`up` set of characters used to move up one resolution unit (`Vert`) in plot mode, if any

`down` set of characters used to move down one resolution unit (`Vert`) in plot mode, if any

`right` set of characters used to move right one resolution unit (`Hor`) in plot mode, if any

`left` set of characters used to move left one resolution unit (`Hor`) in plot mode, if any

`codetab` Array of sequences to print individual characters. Order is **nroff**'s internal ordering. See the file */usr/lib/term/tabuser.c* for the exact order.

`zzz` a zero terminator at the end.

Each `codetab` sequences begins with a flag byte. The top bit indicates whether the sequence should be underlined in the **.ul** font. The rest of the byte is the width of the sequence in units of `Char`.

The remainder of each `codetab` sequence is a sequence of characters to be output. Characters with the top bit off are output as given; characters with the top bit on indicate escape into plot mode. When such an escape character is encountered, **nroff** shifts into plot mode, emitting `ploton`, and skips to the next character if the escape character was "\200".

When in plot mode, characters with the top bit off are output as given. A character with the top bit on indicates a motion. The next bit indicates coordinate, with **1** as vertical and **0** as horizontal. The next bit indicates direction, with **1** meaning up or left. The remaining five bits give the amount of the motion. An amount of zero causes exit from plot mode.

When plot mode is exited, either at the end of the string or via the amount-zero exit, `plotoff` is emitted, followed by a blank.

All quantities that are in units of fractions of an inch should be expressed as INCH**num/denom*, where ***num*** and ***denom*** are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as "INCH/48".

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The Development System must be installed on the computer to create a new driving table. The source code for a generic output device is in the file */usr/lib/term/tabuser.c*. Copy this file and make the necessary modifications, including the name of the termtable struct. Refer to the hardware manual for the codes needed for the output device (terminal, printer, etc.). Name the file according to the convention explained in **term**(CT). The makefile, */usr/lib/term/makefile*, should be updated to include the source file to the new driving table. To perform the modification, enter the command:

cc -M3e -O -c tabuser.c maketerm.o -o maketerm

When the files are prepared, enter the command :

make

See **make**(CP). The source to the new driving table is linked with the object file *mkterm.o*, and the new driving table is created and installed in the proper directory.

Files

/usr/lib/term/tabname	driving tables
/usr/lib/term/tabuser.c	generic source for driving tables
/usr/lib/term/makefile	makefile for creating driving tables
/usr/lib/term/mkterm.o	linkable object file for creating driving tables
/usr/lib/term/terms.h	used to create nroff driving tables

See also

nroff(CT,) **term**(CT)

Notes

The Development System and text processing software must be installed on the computer to create new driving tables.

Not all UNIX facilities support all of these options.

termcap

terminal capability database

Description

The file */etc/termcap* is a database describing terminals. This database is used by packages such as **vi**(C), Lyrix®, Multiplan™, and sub-routine packages such as **curses**(S). Terminals are described in **termcap** by giving a set of capabilities and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap.**

Entries in **termcap** consist of a number of fields separated by colons (:). The first entry for each terminal gives the names that are known for the terminal, separated by vertical bars (|). For compatibility with older systems the first name is always 2 characters long. The second name given is the most common abbreviation for the terminal and the name used by **vi**(C) and **ex**(C). The last name given should be a long name fully identifying the terminal. Only the last name can contain blanks for readability.

Capabilities (including XENIX extensions)

The following is a list of the capabilities that can be defined for a given terminal. In this list, (P) indicates that padding can be specified, and (P*) indicates that padding can be based on the number of lines affected. The capability type and padding fields are described in detail in the following section "Types of capabilities."

The codes beginning with uppercase letters (except for CC) indicate XENIX extensions. They are included in addition to the standard entries and are used by one or more application programs. As with the standard entries, not all modes are supported by all applications or terminals. Some of these entries refer to specific terminal output capabilities (such as GS for "graphics start"). Others describe character sequences sent by keys that appear on a keyboard (such as PU for PageUp key). There are also entries that are used to attribute special meanings to other keys (or combinations of keys) for use in a particular software program. Some of the XENIX extension capabilities have a similar function to standard capabilities. They are used to redefine specific keys (such as using function keys as arrow keys). The extension capabilities are included in the */etc/termcap* file, as they are required for some utilities. The more commonly used extension capabilities are described in more detail in the section "XENIX extensions."

Name	*Type*	*Pad?*	*Description*
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not ^**H**
bs	bool		Terminal can backspace with ^**H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
CO	str		Cursor on
cr	str	(P*)	Carriage return, (default ^**M**)
cs	str	(P)	Change scrolling region (vt100), like **cm**
cv	str	(P)	Like **ch** but vertical only.
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
DA	bool		Delete attribute string
db	bool		Display may be retained below
dB	num		Number of millisec of **bs** delay needed
dC	num		Number of millisec of **cr** delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of **ff** delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of **nl** delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give `:ei=:´ if **ic**
EN	str		Sent by END key
eo	bool		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default ^**L**)
G1	str		Upper-right (1st quadrant) corner character
G2	str		Upper-left (2nd quadrant) corner character

Name	*Type*	*Pad?*	*Description*
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GC	str		Center graphics character (similar to "+")
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and GE
GH	str		Horizontal bar character
GL	str		Left-tick character
GR	str		Right-tick character
GS	str		Graphics mode start
GU	str		Up-tick character
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward ½ linefeed)
HM	str		Sent by HOME key (if not **kh**)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse ½ linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	str		Insert mode (enter); give ':im=' if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key
LF	str		Sent by line feed key
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by **vi** version 2 only
mb	bool		Turn on flash
me	bool		Turn off flash
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
MP	str		Multiplan initialization string

Name	*Type*	*Pad?*	*Description*
MR	str		Multiplan reset string
ms	bool		Will scroll in stand-out mode
mu	str		Memory unlock (turn off memory lock)
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
PD	str		Sent by PAGE DOWN key
PN	str		Start local printing
PS	str		End local printing
pt	bool		Has hardware tabs (may need to be set with **is**)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RT	str		Sent by RETURN key
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by **so** or **se**
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than **^I** or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by **us** or **ue**
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
UP	str		Sent by up-arrow key (alternate to ku)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WL	str		Sent by WORD LEFT key
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce** \r \n (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 264)
xt	bool		Tabs are destructive, magic **so** char (Teleray 1061)

A sample entry

The following entry describes the Concept-100, and is among the more complex entries in the *termcap* file. (This particular Concept entry is outdated, and is used as an example only.)

```
c1 | c100 | concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
        :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:\
        :ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue over to multiple lines by giving a backslash (\) as the last character of a line. Empty fields can be included for readability between the last field on a line and the first field on the next. Capabilities in **termcap** are of three types: Boolean capabilities, which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has 'automatic margins' (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. The description of the Concept includes **am**. Numeric capabilities are followed by the character "**#**" and then the value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an "**=**", and then a string ending at the next following "**:**". A delay in milliseconds may appear after the "**=**" in such a capability, and padding characters are supplied by the editor after the rest of the string is sent to provide this delay. The delay can be either an integer, for example, '20', or an integer followed by an "*****", such as '3*'. A "*****" indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a "*****" is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, ^*x* maps to a Control *x* for any appropriate *x*, and the sequences **\n, \r, \t, \b, \f** give a Newline, Return, Tab, Backspace and Formfeed. Finally, characters may be given as three octal digits after a "****", and the characters "**^**" and "****" may be given as "**\^**" and "****". If it is necessary to place a colon (:) in a capability, it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability, it must be encoded as **\200**. The routines that deal with **termcap** use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and building up a description gradually, using partial descriptions with **ex** to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it. To test a new terminal description, you can set the environment variable **TERMCAP** to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap*. **TERMCAP** can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability. If the terminal can clear its screen, this is given by the **cl** string capability. If the terminal can backspace, it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over), it should have the **os** capability.

A very important point here is that the local cursor motions encoded in **termcap** are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the **termcap** file usually assumes that this is on (that is, **am**).

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the Model 33 Teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as:

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability. This capability uses **printf**(S) -like escapes (such as **%x**) in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, its arguments are the line and then the column to which motion is desired, and the "**%**" encodings have the following meanings:

%d	replaced by line/column position, 0 origin
%2	like %2d - 2 digit field
%3	like %3d - 3 digit field
%.	like **printf**(S) %c
%+x	adds x to value, then %.
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments line/column position (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output
%D	Reverse coding (x-2*(x%16)), no output (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, 'cm=^T%.%.'. Terminals that use "%." need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+ '.

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, it should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), this can be given as **ho**; similarly, a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, the sequence should be given as **ce**. If the terminal can clear from the current position to the end of the display, the sequence should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, the sequence should be given as **al**. Note that this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line on which the cursor rests, the sequence should be given as **dl**. This is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, the sequence can be given as

sb, but **al** can suffice. If the terminal can retain display memory above, the **da** capability should be given, and if display memory can be retained below, then **db** should be given. These let the editor know that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to the insert/delete character that can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and entering text separated by cursor motions. Enter 'abc def', using local cursor motions (not spaces) between the 'abc' and the 'def'. Then position the cursor before the 'abc' and put the terminal in insert mode. If entering characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the 'abc' shifts over to the 'def' which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for 'insert null'. No known terminals have an insert mode, not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Specify **im** as the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Specify **ei** as the sequence to leave insert mode (specify this with an empty value if you also gave **im** an empty value). Now specify **ic** as any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not support **ic**, terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode, these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as reverse video, blinking, or underlining - half bright is not usually an acceptable standout mode unless the terminal is in reverse video mode constantly), the preferred mode is reverse video by itself. It is acceptable, if the code to change into or out of standout mode leaves one, or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do. Although it may confuse some programs slightly, it cannot be helped.

Codes to begin and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, the sequence can be given as **uc**. (If the underline code does not move the cursor to the right, specify the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error silently (a bell replacement), the sequence can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of **ex**, the sequence can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed), even though it does not overstrike, you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by specifying **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not to transmit, enter these codes as **ks** and **ke**. Otherwise, the keypad is assumed always to transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh**. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the **termcap** 2 letter codes can be given in the *ko* capability, for example, ':ko=cl,ll,sf,sb:', which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete, but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding **vi** command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the Mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, the sequence can be given as **ta**.

Terminals that do not allow "~" characters to be displayed (such as Hazeltines), should indicate **hz**. Datamedia terminals that echo carriage-return-linefeed for carriage return, and then ignore a following linefeed, should indicate **nc**. Early Concept terminals, that ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **xx**.

If the leading character for commands to the terminal (normally the escape character) can be set by the software, specify the command character(s) with the capability **CC**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** is displayed before **if**. This is useful where **if** is */usr/lib/tabset/std*, but **is** clears the tabs first.

Similar terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability, **tc**, can be given with the name of the similar terminal. This capability must be **last** and the combined length of the two entries must not exceed 1024. Since **termlib** routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with ***xx*@** where ***xx*** is the capability. For example:

```
hn | 2621nl:ks@:ke@:tc=2621:
```

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

XENIX extensions

Capabilities

This table lists the (previously listed) XENIX extensions to the termcap capabilities. It shows which codes generate information input from the keyboard to the program reading the keyboard and which codes generate information output from the program to the screen.

Name	*Input/Output*	*Description*
CF	str	Cursor off
CL	str	Sent by CHAR LEFT key
CO	str	Cursor on
CW	str	Sent by CHANGE WINDOW key
DA	bool	Delete attribute string
EN	str	Sent by END key
G1	str	Upper-right (1st quadrant) corner character
G2	str	Upper-left (2nd quadrant) corner character
G3	str	Lower-left (3rd quadrant) corner character
G4	str	Lower-right (4th quadrant) corner character
G5	str	Upper right (1st quadrant) corner character (double)
G6	str	Upper left (2nd quadrant) corner character (double)
G7	str	Lower left (3rd quadrant) corner character (double)
G8	str	Lower right (4th quadrant) corner character (double)
GC	str	Center graphics character (similar to +)
Gc	str	Centre graphics character (double)
GD	str	Down-tick character
Gd	str	Down tick character (double)
GE	str	Graphics mode end
GG	num	Number of chars taken by GS and GE
GH	str	Horizontal bar character
Gh	str	Horizontal bar character (double)
GL	str	Left-tick character
Gl	str	left-tick character (double)
GR	str	Right-tick character
Gr	str	right-tick character (double)
GS	str	Graphics mode start
GU	str	Up-tick character
Gu	str	Up-tick character (double)
GV	str	Vertical bar character
Gv	str	Vertical bar character (double)
HM	str	Sent by HOME key (if not *kh*)
mb	str	blinking on
me	str	blinking off
MP	str	Multiplan initialization string
MR	str	Multiplan reset string
NU	str	Sent by NEXT UNLOCKED CELL key
PD	str	Sent by PAGE DOWN key
PU	str	Sent by PAGE UP key
RC	str	Sent by RECALC key

RF	str	Sent by TOGGLE REFERENCE key
RT	str	Sent by RETURN key
UP	str	Sent by up-arrow key (alternate to ku)
WL	str	Sent by WORD LEFT key
WR	str	Sent by WORD RIGHT key

Cursor motion

Some application programs make use of special editing codes. **CR** and **CL** move the cursor one character right and left respectively. **WR** and **WL** move the cursor one word right and left respectively. **CW** changes windows, when they are used in the program.

Some application programs turn off the cursor. This is accomplished using **CF** for cursor off and **CO** to turn it back on.

Graphic mode

If the terminal has graphics capabilities, this mode can be turned on and off with the **GS** and **GE** codes. Some terminals generate graphics characters from all keys when in graphics mode (such as the Visual 50). The other **G** codes specify particular graphics characters accessed by escape sequences. These characters are available on some terminals as alternate graphics character sets (not as a bit-map graphic mode). The vt100 has access to this kind of alternate graphics character set, but not to a bit-map graphic mode.

File

/etc/termcap File containing terminal descriptions

See also

ex(C), **curses**(S), **more**(C), **screen**(HW), **termcap**(S), **tset**(C), **vi**(C)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

ex(C) allows only 256 characters for string capabilities, and the routines in **termcap**(S) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the **vi**(C) program.

Not all programs support all entries. There are entries that are not supported by any program.

XENIX termcap extensions are explained in detail in the software application documentation.

Refer to the **screen**(HW) manual page, for a description of the character sequences used by the monitor device on your specific system.

terminfo

format of compiled terminfo file

Description

Compiled *terminfo* descriptions are placed under the directory */usr/lib/terminfo*. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: */usr/lib/terminfo/c/name* where ***name*** is the name of the terminal, and ***c*** is the first character of ***name***. Thus, *act4* can be found in the file */usr/lib/terminfo/a/act4*. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8- or more-bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the **tic**(C) program, and read by the routine **setupterm** in **terminfo**(S). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value -1 is represented by 0377, 0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines in which this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the *terminfo* description, listing the various names for the terminal, separated by the " | " character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1, as the flag is present or absent. The capabilities are in the same order as the file *<term.h>*.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short-word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of -1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information $<nn> and parameter information %x are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null-terminated.

Note that it is possible for **setupterm** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm** was recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm** must be prepared for both possibilities; this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
    cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
    ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
    cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],

3000 032 001     \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020   o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040   t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100  \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120 377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140  \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160 377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: the total size of a compiled description cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

File

/usr/lib/terminfo//** compiled terminal capability database

See also

terminfo(M), **terminfo**(S), **tic**(C)

timezone

set default system time zone

Description

The **/etc/TIMEZONE** file sets and exports the time zone environmental variable **TZ**.

This file is "dotted" into other files that must know the time zone, including */etc/cshrc*, */etc/profile*, and */etc/rc2*.

TZ contains the following information:

(*sss*) One to nine letters designating the standard time zone.

(*n*) Number of hours past Greenwich mean time for the standard time (partial hours are valid, for example 12:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich.

(*ddd*) One to nine letters designating the local daylight savings time (summer time) zone. If not present, summer time is assumed not to apply.

(*m*) Number of hours past Greenwich mean time for the summer time (partial hours are valid, for example 11:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich. If *m* is not given, the distance to GMT during summer time is assumed to be one hour less than during standard time.

(*start*) The rule defining the day summer time begins. In the southern hemisphere, the ending day will be earlier in the year than the starting day.

(*end*) The rule defining the day summer time ends.

(*time*) The time of day the change to and from summer time occurs. The default is 02:00:00 local time.

The rules for defining the ***start*** and ***end*** of summer time are as follows:

J*n*	1 based Julian day *n* ($1 \le n \le 365$) *
n	0 based Julian day *n* ($0 \le n \le 364$) *
W*n.d*	day *d* ($0 \le d \le 6$)** of week *n* ($1 \le n \le 53$) †
M*m.n.d*	day *d* of week *n* ($1 \le n \le 5$) ‡ of month *m* ($1 \le m \le 12$)

* Leap days (February 29) are never counted; that is, February 28 (J59) is immediately followed by March 1 (J60) even in leap years.

** Sunday is the first day of the week (0). If *d* is omitted, Sunday is assumed. Note that *d* is optional.

† The 5th week of the month is always the last week containing day *d*, whether there are actually 4 or 5 weeks containing day *d*.

‡ The 53rd week of the year is always the last week containing day *d*, whether there are actually 52 or 53 weeks containing day *d*.

If ***start*** and ***end*** are omitted, current U.S. law is assumed.

Examples

A simple setting for New Jersey could be

```
TZ='EST5EDT'
```

where "EST" is the abbreviation for the main time zone, "5" is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and "EDT" is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is

```
TZ='EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00'
```

where "EST" is the abbreviation for the main time zone, "5:00:00" is the difference, in hours, minutes, and seconds between GMT and the main time zone, "EDT" is the abbreviation for the alternate time zone, "4:00:00" is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, "117" is the number of the day of the year (Julian day) when the alternate time zone will take effect, "2:00:00" is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, "299" is the number of the day of the year when the alternate time zone will end, and "2:00:00" is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

```
TZ='KDT9:30KST10:00;64/5:00,303/20:00'
```

This setting means that "KDT" is the abbreviation for the main time zone, "KST" is the abbreviation for the alternate time zone, KST is *9* hours and *30* minutes later than GMT, KDT is *10* hours later than GMT, the starting date of KDT is the *64*th day at *5* AM, and the ending date of KDT is the *303*rd day at *8* PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, via the local */etc/profile* file (see **profile**(M)).

See also

ctime(S), **environ**(M), **profile**(M), **rc2**(ADM), **tz**(M), **initscript**(F).

Notes

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

Standards conformance

timezone is conformant with:

X/Open Portability Guide, Issue 3, 1989.

top, top.next

the Micnet topology files

Description

These files contain the topology information for a Micnet network. The topology information describes how the individual systems in the network are connected, and what path a message must take from one system to reach another. Each file contains one or more lines of text. Each line of text defines a connection or a communication path.

The *top* file defines connections between systems. Each line lists the machine names of the connected systems, the serial lines used to make the connection, and the speed (baud rate) of transmission between the systems. Each line has the following format:

machine1 `tty1a` ***machine2*** `tty2a`***speed***

machine1 and ***machine2*** are the machine names of the respective systems (as given in the *systemid* files). The *ttys* are the device names (for example, tty1a) of the connecting serial lines. The speed must be an acceptable baud rate (for example, 110, 300, ..., 19200).

The *top.next* file contains information about how to reach a particular system from a given system. There may be several lines for each system in the network. Each line lists the machine name of a system, followed by the machine name of a system connected to it, followed by the machine names of all the systems that may be reached by going through the second system. Such a line has the form:

machine1 machine2 machine3 [machine4]...

The machine names must be the names of the respective systems (as given by the first machine name in the *systemid* files).

The *top.next* file must be present even if there are only two computers in the network. In such a case, the file must be empty.

In the *top* and *top.next* files, any line beginning with a number sign (#) is considered a comment, and is ignored.

Files

/usr/lib/mail/top
/usr/lib/mail/top.next

See also

netutil(ADM), **systemid**(F)

ttytype

set terminal types automatically at login

Description

The file */etc/ttytype* will automatically set up a user's terminal type when they log in. The user's *.login* or *.profile* must contain the correct **tset**(C) command for this to work.

In a *.profile*:

```
eval `tset -s`
```

In a *.login*:

```
tset -s -Q > /tmp/tset$$; source /tmp/tset$$; /bin/rm /tmp/tset$$
```

Each line in the */etc/ttytype* file specifies a terminal type for a particular tty line.

The file has the format:

terminal type tty

You can use "unknown" as the terminal type if you want to put an entry in for a particular line, but you don't know its terminal type.

See also

tset(C)

Value added

/etc/ttype is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

utmp, wtmp

format of utmp and wtmp entries

Syntax

```
#include <sys/types.h>
#include <utmp.h>
```

Description

These files, which hold user and accounting information for such commands as **who**(C), **write**(C), and **login**(M), have the following structure as defined by *<utmp.h>*:

```
#define UTMP_FILE      "/etc/utmp"
#define WTMP_FILE      "/etc/wtmp"
#define ut_name        ut_user

struct  utmp {
        char    ut_user[8];             /* User login name */
        char    ut_id[4];               /* usually line # */
        char    ut_line[12];            /* device name (console, lnxx) */
        short   ut_pid;                 /* process id */
        short   ut_type;                /* type of entry */
        struct  exit_status {
            short       e_termination;  /* Process termination status */
            short       e_exit;         /* Process exit status */
          } ut_exit;                      /* The exit status of a process
                                          marked as DEAD_PROCESS. */
        time_t  ut_time;                /* time entry was made */
};
/* Definitions for ut_type */

#define EMPTY           0
#define RUN_LVL         1
#define BOOT_TIME       2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5           /* Process spawned by "init" */
#define LOGIN_PROCESS   6           /* A "getty" process waiting
                                       for login */
#define USER_PROCESS    7           /* A user process */
#define DEAD_PROCESS    8
#define ACCOUNTING      9
#define UTMAXTYPE       ACCOUNTING  /* Largest legal value of ut_type */

/* Special strings or formats used in the "ut_line" field when  */
/* accounting for something other than a process  */
/* No string for the ut_line field can be more than 11 chars +  */
/* a NULL in length  */
#define    RUNLVL_MSG      "run-level %c"
#define    BOOT_MSG        "system boot"
#define    OTIME_MSG       "old time"
#define    NTIME_MSG       "new time"
```

Files

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

See also

getut(S), **login**(M), **who**(C), **write**(C)

Standards conformance

utmp and *wtmp* are conformant with:

X/Open Portability Guide, Issue 3, 1989.

xbackup

XENIX incremental dump tape format

Description

The **xbackup** and **xrestore** commands are used to write and read incremental dump magnetic tapes.

The backup tape consists of a header record, some bit mask records, a group of records describing file system directories, a group of records describing file system files, and some records describing a second bit mask.

The header record and the first record of each description have the format described by the structure included by:

```
#include <dumprestor.h>
```

Fields in the **dumprestor** structure are described below.

NTREC is the number of 512 byte blocks in a physical tape record. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The **TS_** entries are used in the `c_type` field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label.

TS_INODE A file or directory follows. The `c_dinode` field is a copy of the disk inode and contains bits describing what sort of file this is.

TS_BITS A bit mask follows. This bit mask has one bit for each inode that was backed up.

TS_ADDR A subblock to a file (**TS_INODE**). See the description of `c_count` below.

TS_END End of tape record.

TS_CLRI A bit mask follows. This bit mask contains one bit for all inodes that were empty on the file system when backed up.

MAGIC All header blocks have this number in `c_magic`.

CHECKSUM Header blocks checksum to this value.

The fields of the header structure are as follows:

`c_type` The type of the header.

`c_date` The date the backup was taken.

`c_ddate` The date the file system was backed up.

`c_volume` The current volume number of the backup.

`c_tapea` The current block number of this record. This is counting 512 byte blocks.

`c_inumber` The number of the inode being backed up if this is of type **TS_INODE**.

`c_magic` This contains the value **MAGIC** above, truncated as needed.

`c_checksum` This contains whatever value is needed to make the block sum to **CHECKSUM**.

`c_dinode` This is a copy of the inode as it appears on the file system.

`c_count` The following count of characters describes the file. A character is zero if the block associated with that character was not present on the file system; otherwise, the character is nonzero. If the block was not present on the file system no block was backed up and it is replaced as a hole in the file. If there is not sufficient space in this block to describe all of the blocks in a file, **TS_ADDR** blocks will be scattered through the file, each one picking up where the last left off.

`c_addr` This is the array of characters that is used as described above.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a **TS_END** block and then the tapemark.

The structure **idates** describes an entry of the file where backup history is kept.

See also

xbackup(ADM), **xrestore**(ADM), **filesystem**(F)

Value added

xbackup is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

Hardware Dependent (HW)

Intro

introduction to machine related miscellaneous features and files

Description

The hardware-dependent section (HW) contains information useful in maintaining the system. Included are descriptions of files, devices, tables and programs that are directly related to the kind of computer on which the system runs. This section is intended for use with 80386- and i486-based computers.

80387

math coprocessor

Description

The 80387 is the Intel math co-processor for the 80386. The kernel tests for the presence of an 80387 at startup.

If your system has an 80387, you may have to set a switch on the main system board in order to enable 80387 interrupts. Check your hardware manual to determine the proper switch and setting. If your system does not have an 80387, or the switch is on, the kernel will run a set of emulator routines which are much slower.

The C compiler available with the development system generates the appropriate 80387 opcodes. C routines compiled with this compiler have run as much as 200 times as fast as the emulated code. In particular, the standard math library routines run considerably faster if you have an 80387.

The overflow, division by zero, and invalid operand exceptions return a **SIGFPE** signal. This signal can be caught. The rest of the 80387 floating point exceptions (underflow, denormalized operand, and precision error) are masked.

Notes

The emulator returns meaningless information on divide by zero.

There is no obvious way to tell which 80387 exception generated the **SIGFPE**.

Because of design defects in Intel's 80386 chip (B1 stepping), the Intel 80387 math co-processor may not operate correctly in some computers. The problem causes the CPU to hang when DMA/paging/coprocessor accesses are occurring. A workaround for this problem has been engineered that is engaged by using a special string at boot time:

```
Boot
: unix a31
```

This workaround may not work on all machines; some hardware is designed such that it will not work. If it is successful, the following message is displayed:

```
A31 CPU bug workaround in effect
```

If unsuccessful, the following is displayed:

```
A31 CPU bug workaround not possible for this machine
```

The bootstring may also be added to the end of the default bootstring (**DEFBOOTSTR**) found in */etc/default/boot*.

If you cannot use this workaround, you have two options. You may replace the 386 chip with a newer release of the 386 chip (a D-step part), or you can bypass the 387 chip by adding the **ignorefpu** keyword in your boot command as follows:

```
Boot
: unix ignorefpu
```

This means that the operating system will not use the 387 chip, but you need not remove it physically; the coprocessor is still usable from DOS. To automatically bypass the 387 chip every time you boot your system, add the **ignorefpu** keyword to the */etc/default/boot* file. See **boot**(HW) for more information.

For further information, see the Intel publication: *Intel 80387 Programmer's Reference Manual.*

audit

audit subsystem interface device

Description

The audit subsystem provides two minor devices for interfacing to the audit subsystem. One device, */dev/auditr* (audit read), is used exclusively by the audit daemon, **auditd**(ADM), for the purpose of reading the subsystem audit collection file records. The other device, */dev/auditw* (audit write), is used by application programs that are privileged to write audit records to the audit subsystem. This device may be opened by as many applications as necessary but may only be opened for writing. The device also supports a host of **ioctl**(S) functions to perform audit subsystem control.

The audit read device provides the usual character device driver **open**(S), **read**(S), and **close**(S) routines. Writing to this device is not permitted. Read requests are satisfied by the subsystem and optimize the efficiency of the daemon and the performance of the system. Read requests are satisfied when sufficient data has accumulated to meet an administrator-specified threshold. Until the data is available, the read request will block. In this manner, the daemon will receive sufficiently large blocks of data on each read to allow sufficient compaction. Also, the frequency of context switching is greatly reduced since the reads will not be satisfied on small blocks or when no data is available.

The audit write device provides an interface to the audit subsystem for applications that have the *writeaudit* authorization. The device supports the **open**, **close**, **write**, and **ioctl** entry points. Once opened, an application may compose an audit record and **write** it to the device for inclusion in the collection file. The entire audit record must be presented to the subsystem with a single write. It is incumbent on the application to gather the record into a single buffer before writing it to the device.

The format of an audit record depends upon the type of event being audited. All audit records begin with a common audit record header defined by the audit_header structure in the file *sysaudit.h*.

```
struct audit_header {
     ushort     rec_length;     /* total record length */
     time_t     tstamp;         /* date/time of record */
     ulong      event_id;       /* event sequence id */
     ushort     event_type;     /* event classification */
     ushort     record_type;    /* record format */
     ushort     obj_type;       /* object type */
     ushort     pid;            /* process_id */
};
```

The `event_type`, `record_type`, and `pid` fields must be filled by the application; all other fields are filled by the audit subsystem. The event types are defined in the header file and provide a method of categorizing audit records into groups such as login events or system administrator events. The record type informs the subsystem of the record template type. This information is also retained with the record when it is written to the collection file by the subsystem, since it is required at data reduction time.

Some of the record types have variable-length string areas that follow the fixed portion of the audit record. Each text string that is part of the record has its size recorded in a count field. Each string is null-terminated and the count must include the null character. When the record is written to the device, the amount of data written includes the fixed portion plus all text strings. The supported record types for application programs are:

RT_LOGIN	login/logout events
RT_PASSWORD	password modifications
RT_DATABASE	protected database modifications
RT_SUBSYSTEM	privileged subsystem events
RT_LOCK	terminal and account locking
RT_AUDIT	audit subsystem events

Each record type indicates a unique record structure definition. The **RT_LOGIN** record uses the **`login_audit`** structure. It contains the following fields, defined in *sys/audit.h*:

```
struct login_audit {
        struct audit_header aud_hdr;
        char    username[8];    /* login name */
        ushort  code;           /* function code */
        ushort  luid;           /* login userid */
        ushort  rgid;           /* real gid */
        dev_t   ttyd;           /* controlling terminal */
        ptr_t   cdir;           /* current directory */
        ptr_t   terminal;       /* stdin terminal name */
#ifdef B1
        ptr_t   sec_level;      /* login sensitivity level */
#endif
};
```

`username` is the login or logout user account name. The `luid` and `rgid` fields are those associated with the specified user account. The audit header, which precedes the login-specific portion of the record, must have the `record_type` field set to **RT_LOGIN**. The `event_type` used for login/logout is the **ET_LOGIN** event.

The `code` field is used to distinguish between specific actions that may fall into a common category. For instance, the **ET_LOGIN** event category includes both successful and unsuccessful logins, and also logoffs. The code values, defined in the header file, indicate which of these occurred.

The login audit record also contains two variable-length text strings. These are the login terminal and the process current directory. The string area begins immediately following the fixed portion of the record. The size of each text string field is indicated by the **ptr_t** `typedef` field which contains the length of the string including the null character. The null character is considered part of the string. Once the strings have been calculated and the record completed, the length field in the audit record header is set to the size of structure plus the total lengths of the strings. This is the amount of data to **write** to the audit device.

Modifications to user passwords are audited by the password management subsystem. Each attempt, whether successful or not, results in an audit record of type **RT_PASSWORD** being generated. The structure is defined in the *sys/sudit.h* header file:

```
struct passwd_audit {
        struct audit_header aud_hdr;
        char    username[8];    /* login user name */
        ushort  code;           /* function code */
};
```

The `code` value distinguishes between successful and unsuccessful attempts to change the password on the indicated user account.

The system maintains a number of protected database files to support the system security policy. Attempts to modify the databases are audited with the **RT_DATABASE** type records. These records have the following format, as defined in *<sys/audit.h>*:

```
struct database_activity {
    struct audit_header aud_hdr;
    ptr_t     command;       /* command name */
    ushort    code;          /* Type of database audit */
    ushort    object;        /* object type */
    long      expected_val;  /* Expected value of parameter */
    long      present_val;   /* Present value of parameter */
    ptr_t     action;        /* security action that failed */
    ptr_t     result;        /* result of failure */
};
```

The `dbase` and `code` values identify the database and the specific action, whether successful or not. A variable-length text string area is provided to identify precisely the database field along with the old and new database field values. The audit header `length` field includes the size of the string area and the fixed portion of the record.

Protected subsystems use the **RT_SUBSYSTEM** record type to record security related events that occur in subsystem components. `code` is used to identify the subsystem generating the record. Both the command and resulting action as well as the resulting failure are recorded in `command`, `action`, and `result`, respectively.

```
struct subsystem_activity {
    struct audit_header aud_hdr;
    ptr_t     command;     /* command name */
    ushort    code;        /* Subsystem type */
    ptr_t     action;      /* action that failed */
    ptr_t     result;      /* result of failure */
};
```

The **RT_LOCK** record type is used to audit user account and terminal locking events. The `username` identifies the user account which was locked or unlocked. `code` distinguishes between the several events that result in the generation of a lock audit record.

```
#include        <sys/audit.h>

struct lock_audit {
        struct audit_header aud_hdr;
        char    username[12];   /* login username */
        ushort  code;           /* lock function code */
        ushort  trys;           /* failed attempts */
};
```

Programs that interact with and control the audit subsystem are audited with the **RT_AUDIT** record type. The subsystem is enabled and disabled by an application program. The same is true of subsystem parameter initialization and modification. Events such as the initiation and termination of the audit daemon, the execution of the recovery mechanism, data reduction and report generation, and audit file archival are all audited.

The text string portion of the audit record is only applicable for the audit enable function since the initial subsystem collection file must be specified for the daemon log file. All other audit records do not use this field. The `code` indicates which of the above events took place.

```
struct audit_actions {
    struct audit_header aud_hdr;
    ushort    code;      /* audit function code */
    ptr_t     text1;     /* initial collection file */
};
```

The audit device supports a number of **ioctl**(S) functions to control the audit subsystem. The format of the **ioctl** calls is:

```
ioctl (fildes, command, arg)
int fildes, command;
struct audit_init *arg;
        -or-
struct audit_ioctl *arg;
        -or-
struct audit_stats *arg;
```

The **audit_init** structure is only used for the **AUDIT_ENABLE** command to perform subsystem initialization. The structure is defined as follows:

```
struct audit_init {
     uint      buf_length;      /* length of data including header */
     mask_t    audit_flags[1];  /* audit control flags */
     mask_t    event_mask[AUDIT_MASK_SIZE];    /* system event mask */
     uint      read_count;      /* daemon read count to satisfy */
     uint      write_count;     /* write count for coll. file flush */
     long      write_time;      /* write flush time in seconds */
     long      switch_count;    /* collection file size maximum */
     long      caf_maxsize;     /* compacted audit file max size */
     uint      dir_count;       /* directory count */
     uint      uid_count;       /* uid selection count */
     uint      gid_count;       /* gid selection count */
     ulong     dir_offset;      /* fseek of directory names */
     ulong     uid_offset;      /* fseek of uids to select */
     ulong     gid_offset;      /* fseek of gids to select */
     uint      buff_count;      /* number of collection file buffers */
     ulong     session;         /* system boot session number */
     short     audit_uid;       /* audit user uid */
     short     audit_gid;       /* audit group gid */
};
```

The subsystem initialization parameters are established through the **sysadmsh** interface and are written to a parameter file. This file is read and used to fill out the above structure to initialize the subsystem.

The `event_mask` is a bit mask of the selected events to audit during the session. Only events that are enabled will generate audit records. The `read_count` value is used by the subsystem to satisfy audit daemon reads. Only when the specified amount of data is available in the collection file will the read be satisfied.

The flushing of the internal subsystem buffers to the collection file is controlled by the `write_count` and `write_time` fields. When the specified amount of data has accumulated, the buffers will be flushed to disk. A time interval in seconds can also be set which will cause the flushing of data to disk after a certain period of elapsed time.

The `switch_count` controls the size to which subsystem collection files may grow until a file switch is performed. The size of the output compaction files written by the audit daemon are controlled by the `caf_maxsize` parameter. When these files reach this specified size, the daemon performs a switch to a new compaction file and records this fact in the audit session log file. `session` is the current session value that is used in filename generation. The `buff_count` value determines the number of file system blocksize buffers to be allocated by the subsystem for the purpose of internal buffering. At least 2 buffers are allocated, while 4-6 buffers are optimal.

`dir_count` is the number of collection file and compaction file directories that are available to both the subsystem and the audit daemon for the creation of their respective files. If a file write error occurs, both will attempt to use an alternate directory. Both will terminate only when all directories have been tried without success. The directory names are located in the variable-length directory area following the fixed portion of the initialization record. Each pathname is a null-terminated string. The `dir_offset` field points to the start of this variable-length text string area with respect to the start of the structure.

The audit subsystem is capable of selective audit record generation based on user and group IDs. These values may be specified to the subsystem at initialization time using the `uid_count` and `gid_count` values. The actual list of user and group IDs are located at the end of the structure in a variable length table of short integers. The offsets where the ID arrays may be found are located by the `uid_offset` and `gid_offset` values.

The `audit_uid` and `audit_gid` fields are used to communicate certain ID values to the subsystem since these are used to create files with specific owners and groups for security purposes.

All remaining **ioctl**(S) commands except **AUDIT_STATS** use the **audit_ioctl** structure. The **audit_ioctl** structure is defined by the following:

```
struct audit_ioctl {
     uint       read_count;     /* daemon read count */
     uint       write_count;    /* write count for file flush */
     long       write_time;     /* write flush time */
     mask_t     user_control[AUDIT_MASK_SIZE]; /* control mask */
     mask_t     user_disp[AUDIT_MASK_SIZE];    /* disposition mask */
     mask_t     system_mask[AUDIT_MASK_SIZE];  /* system event mask */
};
```

The **AUDIT_STATS ioctl** command uses the following structure for the retrieval and display of statistics.

```
struct audit_stats {
     uint      session;         /* current session number */
     uint      sequence;        /* current sequence number */
     ulong     total_bytes;     /* total bytes written */
     ulong     total_recs;      /* total records written */
     ulong     syscall_recs;    /* system call audit record count */
     ulong     syscall_norecs;  /* system call audit record count */
     ulong     appl_recs;       /* application audit record count */
     ulong     read_count;      /* number of device reads */
     ulong     write_count;     /* number of device writes */
     ulong     coll_files;      /* number of collection files */
     ulong     buffers_used;    /* maximum audit buffer usage */
     ulong     buffer_sleep;    /* number of audit write sleeps */
};
```

The commands supported by the audit device are:

ENABLE Initialize and enable the audit subsystem for the generation of audit records.

SHUTDOWN Notify the audit subsystem that a system shutdown is in progress.

DISABLE Terminate the audit subsystem and close all collection files. The audit daemon is also terminated after the last audit record has been read from the subsystem.

SYSMASK Modify the audit subsystem event mask that controls the generation of audit records based on certain event types.

USERMASK Modify the user event mask for a process. Each process has a mask which can be used to always or never audit certain event types regardless of the system event mask. The mask is a control mask which indicates for each bit set on that the generation of records for the corresponding event type is controlled by the second mask. The second mask is the enable/disable mask which determines whether the event is always or never audited. If a control mask bit is 0, the event is controlled by the system event mask.

FLUSH Modify the write count and time interval values.

DAEMON Modify the audit daemon read count value.

ACK Used by the daemon to acknowledge certain events such as recognition of system shutdown and the disabling of the audit subsystem. Provides a synchronization means between the subsystem and the daemon.

MOUNT The system has transitioned to multi-user state and alternate audit directories are now mounted and available.

STATS Retrieve the current audit subsystem statistics from the audit device.

IDS Specify the user and group IDs to use for selective audit generation.

ioctl calls will fail if any of the following are true:

[EPERM] The process required *SelfAudit* privilege but did not have it.

[EEXIST] An attempt is made to enable audit and it is already running.

[EACCES] An open attempt is made on the audit device and the calling process does not have the *configaudit* or *writeaudit* authorization.

[EBADF] ***fildes*** is not a valid open file descriptor.

[EFAULT] *arg* points to an illegal address.

[EINVAL] *command* is an illegal value.

Files

/dev/auditr
/dev/auditw

See also

auditcmd(ADM), **auditd**(ADM)

"Maintaining system security" chapter of the *System Administrator's Guide*

Diagnostics

Upon successful completion, the device returns a 0. Otherwise, a -1 is returned and **ERRNO** is set to indicate the error.

Value added

audit is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

boot

UNIX boot program

Description

boot is an interactive program used to load and execute stand-alone programs. It is used primarily for loading and executing the UNIX System kernel, but can load and execute any program linked for stand-alone execution. **boot** is a required part of the operating system and must be present in the root directory of the root filesystem to ensure successful loading of the UNIX System kernel.

The **boot** program is invoked by the system each time the computer is started. To restart the system without going through the lengthy **shutdown**(ADM) procedure, you can use the **reboot** command. This causes the system to reboot after shutting down without waiting for keyboard input. See **haltsys**(ADM) for more information.

For diskette boot, the procedure has four stages:

1. The ROMs load the first half of the boot block from sector 0 of the floppy.
2. The boot block loads its second half from sector 1 of the floppy.
3. The now-complete boot block loads **/boot** from the floppy filesystem.
4. **/boot** locates the console and prompts for a command.

For hard-disk boot, the procedure has five stages:

1. The ROMs load in the **masterboot** block from sector 0 on the hard disk.
2. The **masterboot** block then loads the **partition** boot block (**boot0**) from sector 0 of the active partition (see **fdisk**(ADM)).
3. Then, assuming the UNIX partition is active, **boot1** is loaded starting from 1K in the active partition. **boot1** spans 20 physically contiguous 1K blocks on the disk.
4. **boot1** loads **/boot** from the UNIX filesystem.
5. **boot** locates the console and prompts for a command.

/boot and */unix* can lie on tracks that have been mapped by **badtrk**(ADM). **masterboot**, **boot0**, and **boot1** cannot lie on bad tracks.

During installation, a custom **masterboot** is placed on the hard disk. If a non-standard disk is specified, its parameters are stored and enabled in this **masterboot**.

The hard disk boot procedure is invoked if the diskette drive is empty.

boot locates the console using the procedure described in the section "Selecting the system console" and then prompts for a command:

```
SCO UNIX System V/386 [ on i80486 ]

Boot
:
```

The command line you enter should be of the form:

standalone [***arguments***]...

where ***standalone*** is the name of command to execute (for example, **unix.old**), and the (optional) ***arguments*** are the "bootstring" passed to that command. The default ***standalone*** command (that is, the command assumed if only ⟨Return⟩ is pressed) is specified in the defaults file. It is normally the UNIX kernel.

Some ***standalone*** commands are built into **boot**, including:

? Lists the devices **boot** knows about; **boot** can only load stand-alone programs and **boot**-linked driver packages from these devices.

systty=*digit*
Specifies the system console used by the UNIX kernel. Please see the section "Selecting the system console" for details.

mem=*ranges*
Defines the default core location and size available for use by both **boot** and the UNIX kernel. Please see the section "Defining the available memory" for details.

dir [***directory***]
List the files in ***directory***; the default is the root of the filesystem from which **boot** was loaded.

link [***standalone*** [***arguments***]...]
boot prompts for a list of packages to link-edit to the **standalone** command. Please see the section "Boot-time loading of device drivers" for details.

If ***standalone*** is not recognized as one of the built-in commands, then the defaults file is searched for an alias of the form:

standalone [=] ***definition***

The initial defaults file is */etc/default/boot* on the booted media. If the file contains a definition of the form:

`ALTDEF [ = ]` ***defaults***

then the named ***defaults*** file (if it exists) is used instead. This process is repeated (up to 100 times) until the current defaults file does not contain an **ALTDEF** definition, or the specified ***defaults*** file cannot be opened.

The default ***standalone*** command is defined by **defbootstr** in the defaults file. Hence, entering:

```
defbootstr
```

is identical to just pressing ⟨Return⟩.

The ***arguments*** (if any) are appended to the ***standalone*** definition (if any) from the defaults file. The result should name a stand-alone program. **boot** loads that program into core and runs it, passing the booted program most of the specified ***arguments***.

boot uses an extension of the normal UNIX filename syntax; the device containing the file should also be specified. The question mark *?* ***standalone*** command lists the devices known to **boot**. The extended format for the device and pathname are any of the following:

xx(m,o)filename
xx(m)filename
filename
*

Name	Description	Example	
xx	Device name	hd	Hard disk
		fd	Floppy diskette
m	Minor device number	40	Primary hard disk *root* filesystem
		104	Secondary hard disk *root* filesystem
		64	Floppy diskette filesystem
o	Filesystem offset (optional; usually 0)		
filename	Standard UNIX pathname, relative to the filesystem's root directory.		
*	Depending on usage, the default device is typically either the media the computer booted or the media from which **boot** is loading the stand-alone program being booted.		

All numbers are in decimal. See the **hd**(HW) and **fd**(HW) manual pages for minor device numbers of those devices. The offset is optional. Not all filesystem types are recognized; those which are understood include XENIX, S51K, and AFS.

Not all ***arguments*** are passed by **boot** to the loaded ***standalone*** command. Some are processed by **boot** itself:

prompt [="*string*"]
boot prints the ***string*** and waits for Return after loading (but before starting) the **standalone** command.

The default ***string*** is:

```
Loaded, press ⟨Return⟩
```

The ***string*** should be enclosed in quotes (").

mem=*range*
Overrides the default core location and sizing established by the **mem=** ***standalone*** command (described above); please see the section "Defining the available memory" for details.

btld=*device*
The ***device***, in the form where offset ***o*** is optional, from which **boot**-linked driver packages are loaded. The default is the media from which **boot** was loaded. Please see the section "Boot-time loading of device drivers" for details. The alternative form **btlddev=*device*** is equivalent.

link="*pkg1 pkg2*"
Names of BTLD (Boot-Time Loaded device Driver) packages to be linked into the loaded UNIX kernel by **boot**. The list of package names should be enclosed in quotes ("), and the package names separated by spaces. Each package name may be optionally preceded by a ***device***; if no ***device*** is specified, the package is loaded from the above **btld** ***device***. The **link** **standalone** command overrides the **link=**bootstring argument (note that this is opposite of **mem=**). Please see the section "Boot-time loading of device drivers" for details.

All other arguments are passed by **boot** to the stand-alone program. Recent versions of the UNIX kernel make the bootstring arguments passed to it available via the special file */dev/string/boot*.

boot may add a few additional arguments to those entered or read from the defaults file:

kernel.auto
Nothing was typed in response to the **boot** prompt, **AUTOBOOT** is set to **YES** in the defaults file, no drivers are being **boot**-linked, and the **prompt** bootstring argument was not specified. The UNIX kernel and **init**(M) are to start the system assuming no human is present; **init** is passed an argument of **-a**.

If necessary, an appropriate **kernel.systty** bootstring argument is added to the bootstring by **boot**; please see the section "Selecting the system console" for details. For compatability with older systems, **boot** does not include the **kernel module** name; please see the section "Configuring the Kernel" for details.

If **AUTOBOOT=YES** is set in the defaults file, then **boot** will wait ***n*** seconds for a reply to its first prompt (***n*** is the value of **TIMEOUT** in the defaults file). If nothing is typed within those ***n*** seconds, **boot** assumes a reply of RETURN (that is, the **defbootstr** defined in the defaults file) and proceeds to **auto**-boot the system.

If you wish to install DOS on the hard disk, it is recommended that you do so before you install the Operating System. The **dos** stand-alone command boots your installed DOS.

Configuring the kernel

Except for the bootstring arguments described above, **boot** passes all of the arguments to the UNIX kernel. The kernel parses the bootstring to adjust its configuration; for example, to set the root, dump, and swap devices. Appropriate default values are assumed if the bootstring does not specify a value.

All bootstring arguments passed to the kernel have the form:

module.name=value

The ***module***, **=**, and ***value*** are all optional (obviously, if a ***value*** is specified, a **=** must also be given; a **=** without a ***value*** is the same as a ***value*** of **""** or the empty string). The default ***module*** is **kernel**; for safety, unknown **kernel** ***name*****s** or illegal ***value*****s** usually cause the UNIX kernel to halt (thereby forcing the system to be rebooted and a correct ***name*** or ***value*** to be entered).

Some ***values*** specify a ***device***. These ***device values***, which are parsed by the UNIX kernel and not by **boot**, are identical to what **boot** uses:

xnamex **(*m*)**
xnamex **(*m*,*o*)**

except that the device names (UNIX: ***xnamex***; **boot**: ***xx***) are not always identical, and the kernel always ignores the offset (***o***). The default ***device*****s** depend on the kernel's configuration and can be (but usually are not) changed; the default values of the initial configuration shipped with the system are listed below.

The recognized **kernel** ***name*****s** include:

kernel.root=*device*
: The root filesystem; default is usually **hd(40)**. The alternative form **kernel.rootdev=*device*** is equivalent.

kernel.swap=*device*
: The initial swap area; default is usually **hd(41)**. The alternative form **kernel.swapdev=*device*** is equivalent.

kernel.swplo=*s*
: First block (starting with **0**) in the swap area the system can use for swapping and paging. The default is usually **0**.

kernel.nswap=*n*
Number of blocks in the swap area the system can use for swapping and paging. The blocks used are *s* through *s* + *n* - *1* , inclusive (where *s* is the **kernel.swplo** value). Some drivers override the specified **kernel.nswap** and **kernel.swplo** values if they detect they are the swapping device; otherwise there is no default **kernel.nswap** value.

kernel.dump=*device*
The system image dump area. Should the kernel panic, an attempt is made to save the system's memory on this ***device*** for later analysis by **crash**(ADM). If a **kernel.swap** area is specified, it is the default dump area; otherwise the default is usually **hd(41)**. The special device **none** means the kernel should not attempt to save a core image if the system panics. The alternative form **kernel.dumpdev=*device*** is equivalent.

kernel.auto
The system is automatically booting; **init**(M) should be started with an argument of **-a**.

kernel.ronly
The root filesystem is read-only. Under normal conditions, use of this option will prevent any changes from being made to the root filesystem and in normal use will probably cause your system to malfunction. It is intended for use only by the installation procedure.

kernel.inboard
The motherboard is an Intel Inboard, which should be run in "turbo" mode with caching enabled. This bootstring argument does not need to be specified on Inboard-equipped systems (but it is useful for improving performance), and must not be specified on any other type of system.

kernel.ignorefpu
Do not use any 287 or 387 present for floating point computations; the (slower) floating point emulator will always be used instead. The default is to use any FPU found for floating point. This is useful if the FPU is suspected of of malfunctioning or if the kernel incorrectly recognises an FPU when one is not present.

kernel.hd=*xnamex*
The ***xnamex*** driver controls the primary hard disk; ***xnamex*** is the "internal name" of the driver specified in column 1 of its **mdevice**(F) entry. By default, the kernel searches for a hard disk (which it assumes is the primary unit) by querying a pre-configured list of drivers for a hard disk controller which is present.

kernel.disable=*adapter1, adapter2...*
De-configure the pre-configured SCSI host ***adapters***. The de-configured host ***adapters*** will not be used by the system.

kernel.systty=*xnamex* [(***m*** [,***params*** ...])]

The system console is controlled by driver ***xnamex***, and is minor device ***m*** (default **0**). Only the **sio** and **cn** drivers (***xnamex***) are supported. The optional list of ***params*** specify the baud rate, character size, parity, and number of stop bits. The known baud rates include **B50** (50 baud), **B75, B110, B134, B150, B300, B600, B1200, B2400, B4800,** and **B9600** (9600 baud), plus the two undefined speeds **EXTA** (typically 19,200 baud) and **EXTB**. Only two character sizes are recognized: **CS7** (7 bits) and **CS8**. By default, parity is neither checked nor generated. If **PARENB** is specified, parity generation and checking is enabled with a parity bit added to each character. Even parity is used unless **PARODD** (odd parity) is also specified. If **CSTOPB** is specified, two stop bits are used; normally, only one stop bit is used. For example:

```
kernel.systty=sio(1,B1200,PARENB)
```

would specify that the system console is */dev/tty1b* (minor device **1** of the **sio** driver) at 1200 baud with even parity and one stop bit. See the **termio**(M) manual page for additional information. Normally, an alternative system console is specified by use of the **systty=standalone** command or the **SYSTTY** setting in the defaults file. Please see the section "Selecting the system console" for details. Note that the **systty=standalone** command and **SYSTTY** defaults setting use a different syntax than the **kernel.systty** bootstring argument.

kernel.ct=*type*(*base,irq,dma*)

Define the primary tape system connected to the system; by default, the ct driver searches a pre-configured list of tape controllers for one which is present. The ***type*** specifies the controller. The known controllers include: **mountain, ibm6157, everex/archive, tecmar/wangtek, archive, compaq,** and **emerald**. Not all ***type***s are recognized on all systems. The special ***type*** **none** means there is no primary tape (so if there actually is one, it is ignored and cannot be used). The ***base*** is an I/O port address; ***irq*** is the interrupt request (interrupt vector) number; and ***dma*** is the DMA channel. The ***base***, ***irq***, and ***dma*** must be specified unless the ***type*** is **none**. All drivers are interpreted as decimal unless preceded by "0x" for hexadecimal or "0" for octal.

kernel.adapter=*host*(*base,irq,dma*)

Overrides the default hardware parameters for SCSI host adapter ***host1***.

kernel.*xnamex*=*host*(*num, id, lun*)

Specifies the SCSI host adapter used by the generic SCSI interface driver (Sdsk, Srom or Stp).

Specific drivers may use additional bootstring arguments; they do not have a ***module*** of **kernel** but usually use their internal name as the ***module***:

xnamex.name [=[***value*** ...]]
The ***name*** bootstring argument with optional ***value*** applies only to driver ***xnamex***.

Consult the individual driver manual pages for a description of the recognized bootstring arguments.

Selecting the system console

You can select the system console at boot time either by entering **systty=*digit*** ***standalone*** command at **boot's** prompt, or by placing the keyword SYSTTY=***digit*** in the defaults file. The ***digit*** is either **0** or **1**:

Digit	Console
0	Primary display adapter
1	Serial adapter at COM1

boot follows this procedure to determine the system console:

1. If there is a SYSTTY=***digit*** in the defaults file, that defines the console.
2. If SYSTTY is not found or the defaults file is unreadable, **boot** checks for a display adapter and (if one is found) assigns it as the system console.
3. If no display adapter is found, **boot** looks for COM1.

If the console is not a display adapter (that is, it is a serial port such as COM1), **boot** sets the serial port to the parameters specified by the **SERIAL** setting in the defaults file. See the description of the **systty** bootstring argument passed by **boot** to the UNIX kernel for a description of the possible ***params***. The default ***params*** are **B9600** (9600 baud), **CS8** (8 data bits), one stop bit no parity.

After determing the system console, **boot** prompts. You can then change the console recognized by the kernel by entering the **systty=*digit*** **standalone** command. For example, to assign the system console to the serial port at COM1, enter this **standalone** command at the boot prompt:

```
systty=1
```

To have **boot** automatically set the system console to the serial port at COM1 (on every boot), the line:

```
SYSTTY=1
```

should be in */etc/default/boot*.

If the console is not the primary display adapter and the bootstring arguments passed to the **standalone** command do not include a **kernel.systty** bootstring argument as described in "Configuring the kernel", **boot** generates an appropriate bootstring argument.

Defining the available memory

boot attempts to automatically determine how much RAM is in the machine, and where it is located. On most systems this automatic search succeeds in finding the memory. However, on some machines the search does not find all the available memory, or "finds" memory which does not actually exist (or which should not or cannot actually be used). It is possible to control where, and how, **boot** searches for RAM by using either the **mem=standalone** command or bootstring argument:

mem= [***range***] [***/flag***] ... [,...]

The (optional) ***range*** is either:

start - end The memory begins at address ***start*** and ends prior to address ***end***. If the ***end*** is prior to the ***start***, the two addresses are exchanged and the ***/d flag*** is assumed.

start + size The memory begins at address ***start*** and continues for at most ***size*** bytes. (So ***end*** is equivalent to ***start + size***.)

The ***start*** and ***end*** addresses, and the ***size***, are specified in either kilobytes (1024 bytes per kilobyte) or megabytes (1024 kilobytes per megabyte); a suffix of **k** is kilobytes, and **m** is megabytes.

For example, all of the following mean the two megabytes starting at address one megabyte:

```
1m-3m
1m+2m
1024k-3m
1m+2048k
```

The **k** or **m** suffix is required.

The (optional) flags are:

/n This memory range is not DMAable. All memory above **16m** is automatically marked non-DMAable. Only the **standalone** program's text is placed in non-DMAable memory by **boot**.

/d This memory should be scanned downwards (from the ***end*** towards the ***start*** address); normally **boot** verifies the memory ***range*** by scanning from the ***start*** towards the ***end***. Specifying **/d** is equivalent to giving an ***end*** address lower than the ***start*** address.

/r This memory range is reserved; **boot** must not load any part of the stand-alone program into it. However, the memory is available for use by the UNIX system. (To hide memory from both **boot** and the UNIX system, simply do not include it in any ***range***.)

/s This memory is "special" and is best used to load the stand-alone program's text (not data) section. "Shadow RAM" usually should be so marked, but shadowing RAM can only be used if the shadowing feature is disabled. That is, if the shadow RAM is hidden by a "shadowed" copy of the machine's startup ROM, then the underlying RAM is inaccessible to both **boot** and the UNIX kernel and must not be used. Only the stand-alone program's text is placed in "special" memory by **boot**.

/L The UNIX kernel's text must be loaded below **16m**; this flag is recommended when booting earlier releases (that is, versions earlier than Release 3.2 Version 4.0) of .

/p Print a concise summary of the RAM found. This summary can be written down and used as the **mem=**input on a later to force **boot** to find the same memory it found on this **boot** (provided none of the described RAM was removed in the meantime).

All memory below **640k** is automatically determined by the machine, and there is never any memory in the range **640k-1m**. All **mem=** descriptions of this "base memory" below **1m** are silently ignored. Hence, **mem=** should only be used to specify the extended memory configuration at and above **1m**.

The default **mem=** specification is equivalent to:

```
mem=1m-16m,16m-256m/n
```

on most machines (plus up to 640 kilobytes of base memory). **boot** scans each ***range*** and **16m-256m** by default, stopping its scan as soon as no memory can be found. Thus, if there is a "hole" in the specified memory ***range***, all RAM above the first hole in the range will not be found by **boot**.

Consider a machine with:

1. 512 kilobytes of base memory,
2. extended RAM from one to three megabytes,
3. some additional RAM from ten to twenty megabytes, and
4. 50 kilobytes of shadow RAM ending at 32 megabytes (shadowing disabled).

By default, **boot** would only find:

a. 512 kilobytes of base memory,

b. extended RAM from one to three megabytes, and

c. additional RAM from sixteen (16) to twenty megabytes.

Most of the RAM — the six megabytes starting at address ten megabytes — is not found (due to the hole starting at address three megabytes). Clearly then, on such a machine, it's advisable to define the memory which really exists.

This machine can be specified as:

```
mem=1m-3m,10m-20m,32718k-32m/d/s
```

Note that **32718k** is the address starting **50k** before **32m**. Since all memory above **16m** is automatically marked as non-DMAable, the above is equivalent to:

```
mem=1m-3m,10m-16m,16m-20m/n,32718k-32m/d/s
```

A simple:

```
mem=/p
```

will print out the automatically found memory (which is:

```
mem=0k-512k,1m-3m,16m-20m/n
```

on the example machine); to print out the memory found for a specific configuration, **/p** must be included in the **mem=** specification. Thus, adding **/p** to the first specification above:

```
mem=1m-3m,10m-20m,32718k-32m/d/s,/p
```

would print out the equivalent of the second definition (on the described machine):

```
mem=0k-512k,1m-3m,10m-16m,16m-20m/n,32718k-32m/d/s
```

On most machines **boot** scans the defined memory ranges and deletes from the definition any RAM not actually found. Thus, if the following is specified on the described machine:

```
mem=1m-5m,12m-25m,31m+1m/d/s,/p
```

the printed result would be:

```
mem=0k-512k,1m-3m,12m-16m,16m-20m/n,32718k-32m/d/s
```

Note that the RAM from ten to twelve megabytes, despite existing, was not found by **boot**, because it was excluded from the **mem=** definition.

On most machines, shadowing must be disabled using the machine's setup procedures. **boot** automatically disables startup ROM shadowing only on a few machines. The RAM hidden by shadowing the startup ROM cannot be used by **boot** or the UNIX kernel unless the shadowing is disabled.

Boot-Time loading of device drivers

Some device drivers can be added to the loaded UNIX kernel by **boot** prior to the kernel starting to run. Such a driver is called a BTLD: Boot-Time Loaded device Driver.

The **boot**-linker (which does the boot-time loading) is invoked by either entering the **link standalone** command or by giving the **link=**bootstring argument.

BTLDs are distributed in "packages". The **link standalone** command prompts for the names of the packages to boot-load:

```
What packages do you need linked into the system,
or q to quit?: pkg1 pkg2
```

and the **link=**bootstring argument takes the names of the packages as its value:

```
link="pkg1 pk2"
```

The **link standalone** command causes any **link=**bootstring argument to be ignored.

After the UNIX kernel is loaded, each package ***pkg*** is prompted for. The appropriate diskette should be inserted into the drive. If the package name (***pkg***) does not include a ***device***, the default device defined by the **bltd** (or **btlddev**) bootstring argument device is used. That is usually the same media as **boot** was loaded from.

The diskette should contain the files described in **btld**(F).

For each driver listed in */pkg/install/btld* the same process is repeated (an error in a preceding step aborts the link and subsequent steps are not done):

1. The driver's *Master*, *System* and *Bootload* files (if any) are read.
2. The appropriate object modules (as defined in *Bootload*) are loaded; the default is **Driver.o** and **Space.o**.
3. Any **tune**-able parameters (defined in *Bootload*) are prompted for.
4. The function dispatch tables defined in *Master*, and any **tune**d or **patch**ed symbols are adjusted. If any conflicts occur (for example, if the interrupt vector that the boot-loaded driver wants to use is already occupied by another driver), **boot** explains the problem, lists the possible resolutions, and prompts for what to do about the conflict. Please see the section on "Resolving BTLD conflicts" for details.
5. Unresolved references in the object modules are resolved.

The **boot**-linker queues the changes to the UNIX kernel (step 4), so if there is a problem in resolving references (step 5) the linking can be aborted without forcing a reboot: the loaded kernel has not been changed. The queued changes to the kernel are not applied until the relinking (step 5) has completed successfully.

If every driver in the package was successfully linked by the **boot**-linker, the first line of the file */pkg/install/bootstring* is appended to the bootstring and the name of the package is added to the "package string" passed to the UNIX kernel. The package string can be read from the special file */dev/string/pkg*.

Hardware usually has jumpers, switches, or a DOS (or stand-alone) setup program to configure the board. When **boot**-loading the drivers for such boards, the **tune** directive in the *Bootload* file (see **btld**(F)) causes **boot** to prompt for the settings of these jumpers or switches. The information **boot** may prompt for includes:

vector — The interrupt vector or IRQ used by the hardware.

DMAchan — The channel used by the hardware to directly access memory.

SIOA — Base or start I/O register address of the board.

EIOA — Ending (or extra) I/O register address of the board.

SCMA — Start or base bus, controller, or dual-ported memory address used by the board.

ECMA — Ending bus, controller, or dual-ported memory address used by the board.

units — Number or size of the peripherals attached to the board.

The manufacturer's instructions should say how to set the jumpers or switches, and what to reply in response to any prompts.

Resolving BTLD conflicts

The *Master* and *System* files on the BTLD diskette instruct the **boot**-linker how to configure the **boot**-loaded driver into the loaded kernel. Three types of conflicts with drivers already present in the kernel may occur:

1. Another driver is already using the major device number the **boot**-loaded driver wants to use,
2. Another driver already occupies the interrupt vector the **boot**-loaded driver wants to occupy,
3. A function dispatch table is full and so the appropriate **boot**-loaded driver's function cannot be added to the table.

When a conflict is detected, **boot** explains the problem and offers several possible ways of dealing with the issue. If there is a comparatively safe alternative, that is the default resolution.

There are several answers which may always be given:

q Quit: The **boot**-linking of this driver is stopped; the kernel has not been changed.

RETURN Use the default answer (if any).

M List the major device numbers and associated drivers.

I[***vecno*** **]** Display interrupt vector ***vecno*** (0-255).

T***funcs*** Prints the function dispatch table ***funcs***; some of the dispatch tables include **io_init**, **io_start**, and **io_halt**.

The major device number is how the UNIX kernel refers to a specific driver. Each driver is identified by its unique major number. When **boot**-loading a driver, major numbers may present a problem because:

- The *Master* file specifies a major number which is already in use.
- The *Master* file does not specify a particular major number, but there are no available (free) major numbers.
- The driver is both a block and character device, but the block and character major numbers specified in the *Master* file differ.
- The major number specified in the *Master* file is out-of-range.
- Another driver has the same internal name (column 1 **mdevice**(F)). This should never happen unless you are **boot**-linking a driver to a kernel which already contains that driver; doing so (called "driver replacement") is unreliable.

The possible resolutions include:

a Add the driver using an available (free) major number.

number Use major ***number***; if that major is in use, **boot** prompts again to confirm the choice.

r Replace the "other" driver (which has either the same internal name or is using the desired major number). This is inherently unreliable, not supported, and is never recommended.

Major device number conflicts are unlikely to occur. Should one occur, **boot**'s default (if possible) is **a** — add the driver using an available major number. The initial system installation procedure does not rely on **boot**-loaded drivers using any specific majors, even if the *Master* file gave a specific value. (However, there may be conflicts later when the driver is added to the kernel using the Link Kit.)

Most devices use interrupts. When they complete, or are ready to do some I/O, they asynchronously notify the CPU and wait for a response. Drivers for devices which use interrupts usually have an interrupt-handling procedure and priority defined in the *System* file (columns 6 and 7 in **sdevice**(F)). Some drivers want the **boot**-linker to install the handler at the specified priority; these drivers have a non-"0" type (column 5 in **sdevice**(F)) file and no **G** characteristic in *Master*. Other drivers only want the **boot**-linker to check that the driver itself should be able to later install its handler at that priority; these drivers also have a non-"0" type but include the **G** characteristic. When **boot**-linking a driver, interrupt handlers (or "vectors") and priorities (or IPL — "interrupt priority level") can be a problem because:

- There is another handler of the same name at this vector. This should never happen unless you are **boot**-linking a driver to a kernel which already contains that driver; such driver replacement is unreliable.
- The **boot**-loaded driver wants exclusive use of the vector, but the vector is already in use. (The driver's type is **1** or **2**.)
- The **boot**-loaded driver is willing to share use of the vector, but the vector is in use by another driver that wants exclusive use. (The driver's type is **3** or more, but the vector is occupied by a type **1** or **2** handler.)
- The vector is full and cannot be shared with any more handlers.
- The vector is in use and sharable but the priority (IPL) is wrong.

The possible resolutions depend on whether or not the **boot**-linker should actually configure the handler or just check the configuration. If the **boot**-linker is only checking the configuration (the driver has the **G** characteristic), the possible answers include:

d Delete all handlers from the vector. The corresponding devices probably cannot be used, or the system may not function properly.

number Delete a specific handler from the vector. That device probably cannot be used or the system may not function properly. A **?** lists the handlers using the vector.

s Convert an unsharable vector into one which can be shared. This may result in a handler not designed to share a vector nonetheless sharing a vector; the system may not function properly.

If the **boot**-linker should configure the handler into the system, the possible answers include:

number Which handler already installed in the vector should be replaced by the **boot**-loaded driver's handler. The device controlled by the replaced driver probably cannot be used and the system may not function properly.

a Add the **boot**-linked driver's handler to the vector. The system should work unless the vector was unshareable (and hence had to be converted to a sharable vector); if such a conversion is done, the system may not function properly.

r Replace the entire list of handlers sharing this vector with the **boot**-linked driver's handler. Probably none of the replaced devices can be used or the system may not function properly.

An additional resolution is always possible:

n Do nothing. The **boot**-loaded device probably cannot be used, or the system may not function properly.

If a vector is already in-use, the interrupt priority level (IPL) may need to be adjusted:

l The **boot**-loaded driver's IPL is less than that of the vector; lower the vector's IPL to that of the driver.

r The **boot**-loaded driver's IPL is greater than that of the vector; raise the vector's IPL to that of the driver.

c Keep the vector's current IPL ignoring the driver's setting.

number Set the vector's IPL to ***number*** (1 - 7 inclusive).

Since the system will probably malfunction if the driver's interrupts occur with a higher priority that the driver was designed for, **r** (raise) is never recommended. Whichever of **l** (lower) or **c** (current) that results in the least IPL (lowest priority) is always recommended.

Interrupt vector and priority conflicts are the most common problems encountered when **boot**-linking a driver to a kernel. The best resolution is to shut down the machine, change the IRQ (interrupt vector) setting on board using the jumpers, switches or software setup as per the manufacturer's instructions, and try the **boot**-link again. Hardware which has jumper or switch selectable configuration parameters is preferable to "hard-wired" (fixed) equipment. The **boot**-loadable drivers for jumper or switch selectable boards should use the **tune** directive in the *Bootload* file to prompt for the settings of the switches or jumpers. The new values should be entered in response to the prompts, as per the manufacturer's instructions.

If the hardware does not have a switch-, software- or jumper-selectable IRQ, or the **boot**-linker fails to prompt for values, there is no ideal resolution. If possible, **boot**'s default action is either **s** (convert an unshareable vector into a shareable one) or **a** (add the handler to the vector, converting the vector if necessary). Either may result in an uninstallable system.

Function dispatch tables list driver routines called on certain events (such as system startup or shutdown). Problems which may arise when adding a **boot**-linked driver to these tables include:

- The dispatch table is full.
- There already is another routine of the same name in the table. This should never happen unless you are **boot**-linking a driver to a kernel which already contains that driver; such driver replacement is unreliable.

The possible resolutions include:

number Replace a routine already in the dispatch table with the **boot**-linked driver's routine.

a Add the driver's routine to the table.

n Do nothing; the driver's routine is not added. The **boot**-linked device may not work.

Dispatch table conflicts are unlikely to occur. Should one happen, **boot**'s default (if possible) is **a** — add the driver's function to the end of the table.

Default file settings

The defaults file is used by several programs involved in the startup process, including **boot** itself and **init**(M). The default defaults file is */etc/default/boot* on the filesystem from which **boot** was loaded; however, this can be changed for **boot** only by use of **ALTDEF**, described below.

All keywords and aliases **boot** recognizes are n the form:

name [=] ***value***

If no = is used, one or more tabs or spaces should separate the ***name*** from the ***value***. An = separating ***name*** from ***value*** may be preceded or followed by spaces and tabs. Programs other than **boot** which read the defaults file require the = and do not allow spaces or tabs.

The keywords **boot** recognizes include:

ALTDEF=*defaults*
The file ***defaults***, if it exists, is used instead as the default file (for **boot**). This process is repeated up to 100 times.

AUTOBOOT=YES or **NO**
If **YES**, **boot** automatically loads the UNIX kernel as per the **defbootstr** setting after waiting for input for the time specified by **TIMEOUT**. The default is **NO**.

RONLYROOT=YES or **NO**
Whether or not the root filesystem is mounted **readonly**. This must be used only during installation, and not for a normal boot. It will effectively prevent writing to the filesystem.

TIMEOUT=*n*
How many seconds to wait for input after the first prompt before assuming an answer of RETURN (thus causing the **defbootstr** to be used). Applies only when **AUTOBOOT** is **YES**; default is **60** seconds.

SYSTTY=*digit*
If ***digit*** is **0**, the system console is the display adapter; if **1**, the system console is COM1. Please see the section "Selecting the system console" for details, including the default console search procedure.

SERIAL=*param1, param2...*
List of parameters specifying the baud rate, character size, parity, and number of stop bits of the serial console (**SYSTTY=1**). The ***params*** are the same as for the **kernel.systty** bootstring argument. Please see the section "Selecting the system console" for details, including the default values.

PWRCHECK=Y or **N**
Applies only to some machines equipped with a built-in uninterrupible power supply (UPS): if **Y**, checks the mains and battery condition; if the battery is low or the mains power has failed, warns that the system probably should not be booted. Default is **N**; note that this option applies only to a few machines and almost all available systems cannot use this facility.

defbootstr=standalone *arguments...*
The **standalone** program to run (with optional ***arguments***) when only a RETURN is entered or when **AUTOBOOT=YES** and nothing is entered within **TIMEOUT** seconds.

standalone=definition
boot checks to see if the entered **standalone** command is defined (or "aliased") in the defaults file; if it is, its ***definition*** is substituted for the entered **standalone** command, with any entered ***arguments*** appended to the end.

For example, if **boot** was loaded from the hard disk (**hd(40)**) and the entered command was:

```
test mem=1m-12m
```

and **test** is aliased as:

```
test  /etc/conf/cf.d/unix prompt="Ready? "
```

then the result bootstring is:

```
hd(40)/etc/conf/cf.d/unix prompt="Ready? " mem=1m-12m
```

By convention, ***standalone*** aliases (including **defbootstr**) are lower-case and keywords are UPPER-CASE, but **boot** will recognize both aliases and keywords in either case. (However, the case of the values, such as **YES** and **NO**, must be UPPER-CASE as shown above.)

Several other commands besides **boot** involved in the bootup procedure also use the */etc/default/boot* file. **init**(M) recognizes the keywords:

MAPKEY=YES or **NO**

SERIAL8=YES or **NO**

PANICBOOT=YES or **NO**
Whether or not the system reboots after a panic; default is **NO**.

MULTIUSER=YES or **NO**

SLEEPTIME=*n*
Seconds between **sync**(S); that is, seconds between periodically forcing all cached-but-unwritten dirty (new or changed) blocks to be written to the disk. Default is **60** seconds (one minute).

SPAWN_INTERVAL=*n*
If an **inittab** entry is respawned **SPAWN_LIMIT** times within this amount of time (seconds), **init** will not try to respawn that entry for **INHIBIT** seconds (unless a **"telinit q"** is done). The default value is **120** seconds.

SPAWN_LIMIT=*n*
If an **inittab** entry is respawned this many times within **SPAWN_INTERVAL** seconds, **init** will not try to respawn that entry for **INHIBIT** seconds (unless a **"telinit q"** is done). The default value is **10** attempts.

INHIBIT=*n*
If an **inittab** entry is respawned **SPAWN_LIMIT** times within **SPAWN_INTERVAL** seconds, **init** will not try to respawn that entry for this many seconds (unless a **"telinit q"** is done). The default value is **300** seconds (five minutes).

DEFAULT_LEVEL=*n*
The run-level to enter when leaving single-user mode.

Diagnostics

If an error occurs, **masterboot** displays an error message, and locks the system. The following is a list of the most common messages and their meanings:

IO ERR An error occurred when **masterboot** tried to read in the partition boot of the active operating system.

BAD TBL The bootable partition indicator of at least one of the operating systems in the **fdisk** table contains an unrecognizable code.

NO OS There was an unrecoverable error that prevented the active operating system's partition boot from executing, or there was no active partition.

When **boot** displays error messages, it returns to the "Boot" prompt. The following is a list of the most common messages and their meanings:

`bad magic number`
: The given file is not an executable program.

`can't open <`***pathname***`>`
: The supplied pathname does not correspond to an existing file, or the device is unknown.

`Stage 1 boot failure`
: The bootstrap loader cannot find or read the *boot* file. You must restart the computer and supply a filesystem disk with the *boot* file in the root directory.

`not a directory`
: The specified area on the device does not contain a valid UNIX filesystem, or the given pathname tries to pass through a file (e.g. */bin/sh/foo*.

`zero length directory`
: Although an otherwise valid filesystem was found, it contains a directory of apparently zero length. This most often occurs when a pre- System V UNIX filesystem (with incorrect, or incompatible word ordering) is in the specified area.

`fload:read(`***x***`)=`***y***
: An attempted read of ***x*** bytes of the file returned only ***y*** bytes. This is probably due to a premature end-of-file. It could also be caused by a corrupted file, or incorrect word ordering in the header.

`Error: request outside range of BIOS (1023 cylinders)`
: The low level hard disk transfer routine was unable to load a file because it lay on or beyond the 1024th cylinder of the boot device. This message is usually followed by another error message indicating the file which which was being read when the failure occured. The boot prompt is then displayed. The problem occurs when the kernel or */etc/default/boot* are innaccessible to the BIOS disk transfer routine which is used to read the hard disk. This problem only occurs on large disk systems. The solution is to move the file so that it lies within the first 1024 cylinders on the boot device.

Files

/boot
/etc/default/boot
/etc/masterboot
/etc/hdboot0
/etc/hdboot1
/etc/fd135ds18boot0
/etc/fd135ds9boot0
/etc/fd48ds9boot0
/etc/fd96ds15boot0
/etc/fd96ds18boot0
/etc/fd96ds9boot0
/dev/string/boot
/dev/string/pkg
/unix

See also

autoboot(ADM), **badtrk**(ADM), **crash**(ADM), **fd**(HW), **fdisk**(ADM), **fsck**(ADM), **haltsys**(ADM), **hd**(HW), **idbuild**(ADM), **idconfig**(ADM), **init**(M), **ld**(CP), **screen**(HW), **serial**(HW), **shutdown**(ADM), **string**(M), **sulogin**(ADM), **systty**(M), **btld**(F)

Device Driver Writer's Guide

Notes

The computer tries to boot off any diskette in the drive. If the diskette does not contain a valid bootstrap program, errors occur.

The **boot** program cannot be used to load programs that have not been linked for stand-alone execution. To create stand-alone programs, the **-A** option of the UNIX linker and special stand-alone libraries must be used.

Stand-alone programs can operate in real or protected mode, but they must not be large or huge models. Programs in real mode can use the input/output routines of the computer's startup ROM.

Some of the UNIX kernel's bootstring arguments should not be using a ***module*** of **kernel**; these historical exceptions include **kernel.ct**. All new driver-specific keywords should use a module name of ***xnamex***, where ***xnamex*** is the internal name of the driver.

Older systems do not recognize module name prefixes, and so **boot** omits **kernel** from any keywords it automatically adds, such as **auto** and **systty**.

The **boot**-linker cannot and does not detect all the errors detected by the **ld**(CP) linker and the **idconfig**(ADM) and **idbuild**(ADM) configuration tools. It is possible to **boot**-link to a kernel a driver which could not otherwise exist in that kernel. This behavior should not be relied upon.

Value added

boot is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

cdrom

compact disk devices

Description

The **cdrom** devices implement the interface with compact disk drives.

The character special CD devices (*/dev/rcd0* and so forth) support raw I/O in multiples of the physical sector size of the CD-ROM (typically 2048 bytes).

The block special CD devices (*/dev/cd0* and so forth) support buffered I/O.

The minor device number determines which compact disk unit will be accessed. The correspondence beween the unit number and the SCSI host adaptor, controller and LUN is defined in the SCSI configuration file */etc/conf/cf.d/mscsi*.

Files

/dev/cd[0-n]
/dev/rcd[0-n]
/usr/lib/mkdev/cdrom

See also

mkdev(ADM), **scsi**(HW)

Notes

Because the CD-ROM is a read-only device it is only possible to open it for input.

The command **mkdev cdrom** can be used to interactively configure the CD-ROM driver.

clone

open any minor device on a STREAMS driver

Description

clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to **clone** during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate stream to a previously unused minor device.

The **clone** driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls (including **close**(S)) require no further involvement of **clone**.

clone will generate an **ENXIO** error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

Warnings

Multiple opens of the same minor device cannot be done through the **clone** interface. Executing **stat**(S) on the filesystem node for a cloned device yields a different result from executing **fstat**(S) using a file descriptor obtained from opening the node.

See also

log(HW), **log**(M), **pipe**(ADM), **pipe**(S).

cmos

display and set the configuration data base

Syntax

cmos [***address*** [***value***]]

Description

The **cmos** command displays and/or sets the values in the CMOS configuration database. This battery-powered database stores configuration information about the computer that is used at power up to define the system hardware configuration and to direct boot procedures. The database is 64 bytes long and is reserved for system operation. Refer to your hardware manual for more information.

The **cmos** command is typically used to alter the current hardware configuration when new devices are added to the system. When only ***address*** is given, the command displays the value at that address. If both ***address*** and a ***value*** are given, the command assigns the value to that address. If no arguments are given, the command displays the entire contents of the data base.

The CMOS configuration database may also be examined and modified by reading from and writing to */dev/cmos* file. Because successful system operation depends on correct configuration information, the database should be modified by experienced system administrators only.

The computer manufacturer's diagnostic diskette should be run before setting the CMOS database.

Files

/etc/cmos
/dev/cmos

dat

digital audio tape device

Description

Digital Audio Tape (DAT) devices, such as the HP DAT device, are cassette tape drives based on DAT technology. These devices use a Digital Data Storage (DDS) recording format developed for computer applications. Digital Audio Tapes are also referred to as either *4-mm tapes* or *DAT tapes*.

The DDS media requires no pre-formatting unless you want to partition the tape. See the section "Partitions" later in this manual page. A typical 60-meter cassette holds approximately 1300 megabytes of data.

The DAT drive operates similarly to other SCSI cartridge tape drives (and traditional nine-track drives). You can use typical UNIX system utilities (such as **tar**(C), **cpio**(C), and **tape**(C) with DAT drives in the same way that you use these utilities with other SCSI tape drives.

Several DDS features are explained in later sections: setmarks, fast search, and partition support. These features allow applications greater speed and flexibility in archiving and accessing data.

Setmarks

Data on tapes is usually organized as a sequence of one or more tape records, forming a file. A tape can contain a sequence of tape records and files. Filemarks indicate the end of a file and mark separation between files. Positioning a tape to a filemark occurs much faster than the typical read/write speed.

The DDS format introduces an additional type of mark, called the *setmark*. Logically, it is a higher organizational unit than a filemark; a search to a setmark ignores filemarks (whereas the reverse is not true). Together, filemarks and setmarks are called *tapemarks*.

The DDS format still uses records and filemarks in the standard manner. If you, or an application, choose to ignore setmarks, then the DDS drive responds the same way as other SCSI tape devices.

One way to use setmarks is by grouping together sets of files that are logically connected. This is especially valuable on larger capacity tapes (such as the HP DAT tape) that can contain numerous files. By using setmarks in conjunction with filemarks, access to a specified file can be improved dramatically.

You can write setmarks by using the **tape** command. See **tape**(C).

Fast search

This is a function of the drive firmware. It cannot be evoked by a command from the host system. The DDS mechanism enables a tapemark search at speeds in excess of 100 times the normal read/write speed. At these speeds, a search for a setmark on a 1300 megabyte tape typically takes only 22.5 seconds.

Partitions

The DDS format allows the tape to be formatted into two entirely separate and independent partitions, each with its own distinct data area. Each partition can have a minimum size (this value is dependent on the tape type; for example, HP DAT specifies a minimum value of 1 megabyte) and a maximum size of the whole tape. Select the desired device file to access individual partitions.

You do not need to partition or format a DDS tape before use. If you want to use the simplest mode of operation, you can take any DAT cassette, insert it into the drive, and start to use it immediately. In this case, the tape effectively has one partition that spans the entire tape.

If you format the tape into two partitions, the tape behaves as two independent units. Each partition acts as a logically distinct tape, and tape operations are specific to the partition selected (via the device file).

Several examples illustrate this. This example shows a partitioned tape with device driver files:

```
ls -l /dev/nurStp1 /dev/nrStp1.1
crw-rw-rw-   2 root   other   46,  4  /dev/nurStp1.0
crw-rw-rw-   1 root   other   46, 68  /dev/nurStp1.1
```

To simplify references to the two partitions, you can link the device filenames to mnemonic filenames:

```
ln  /dev/nurStp1.0 /dev/part1
ln  /dev/nurStp1.1 /dev/part2
```

In the following example, the first command archives */dir_A* onto partition 1, and the second command archives */dir_B* onto partition 2:

```
find /dir_A  -print | cpio -oB > /dev/part1
find /dir_B  -print | cpio -oB > /dev/part2
```

Note that both partitions are on the same tape. If an application rewinds the tape, selecting */dev/part2*, then the tape head is positioned at the beginning of the tape for partition 2.

Use the **tape** command to format a tape into two partitions. For example, to create a 500 megabyte partition on a DDS tape, enter:

```
tape -a 500 partition
```

Partition 2 is 500 megabytes, while partition 1 is the remainder of the tape. For a 1300 megabyte tape, this implies that partition 1 is approximately 800 megabytes. To reformat two partitions into a single partition, execute the **tape** command with a partition size of 0.

Files

The device files created for the DAT are:

/dev/urStp1.0	*/dev/urStp1.1*
/dev/nurStp1.0	*/dev/nurStp1.1*
/dev/nrStp1.0	*/dev/nrStp1.1*
/dev/xStp1.0	*/dev/xStp1.1*

The DAT partition 1 is linked to the default SCSI tape device locations:

/dev/rStp1	linked to	*/dev/nurStp1.0*
/dev/rStp1.0	linked to	*/dev/nurStp1.0*
/dev/nrStp1	linked to	*/dev/nrStp1.0*
/dev/xStp1	linked to	*/dev/xStp1.0*
/dev/urStp1	linked to	*/dev/urStp1.0*
/dev/rStp1.1	linked to	*/dev/nurStp1.1*

The following tables define the filename prefixes and suffixes:

Prefix	Definition
nu	no unload on close
u	unload on close
r	rewind on close
nr	no rewind on close
x	control override (same as SCSI tapes)

Suffix	Definition
.0	partition 1
.1	partition 2

Standard SCSI tape devices */dev/rStp1*, */dev/nrStp1*, and */dev/xStp1* are linked into the *.0* device nodes.

/dev/rStp1 is linked to */dev/nurStp1.0*. This creates a minor number for *rStp1* that is different than the minor number for a standard SCSI tape.

Notes

The HP DAT drive supports Immediate Response. (Immediate Response is the ability of the drive to return good status on write requests after the data has been written to the buffer, but before the actual data is written to tape.) Immediate Response is automatically enabled. The device performs all error detection and error correction.

Although the HP DAT drive can operate in both variable mode and fixed-block mode, only fixed-block mode is supported for higher performance.

See also

tape(C), **tape**(HW)

fd

floppy devices

Description

The **fd** devices implement the interface with floppy disk drives. Each device name corresponds to a specific major and minor device. Typically, the **tar**(C), **cpio**(C) or **dd**(C) commands are used to read or write floppy disks. For instance,

tar tvf /dev/fd0

tabulates the contents of the floppy disk in drive 0 (zero).

The block special **fd** devices are also block-buffered. The floppy driver can read or write 1K bytes at a time using raw I/O. Note that block transfers are always a multiple of the 1K disk block size.

XENIX devices

XENIX diskette device file names use the following format:

/dev/[r]fd[0|1][48ss8|48ss9|96ds9|96ds15|135ds9|135ds18]

(See notes below for more information about the device naming procedure.) The corresponding character special (raw) devices allow direct, unbuffered transmission between the floppy and the user's read or write transfer address in the user's program.

For information about formatting, see **format**(C).

The minor device number determines what kind of physical device is attached to each device file. When accessing the character special floppy devices, the user's buffer must begin on a word boundary. The count in a **read**(S), **write**(S), or **lseek**(S) call to a character special floppy device must be a multiple of 1K bytes.

Device names determine the particular drive and media configuration. The device names have the form:

fd048ds9

Where:

fd0	= drive number (0, 1, 2 or 3)
48	= number of disk tracks per inch (48 or 96)
ds	= single or double sided floppy (ss or ds)
9	= number of sectors per track (8,9,15 or 18)

For instance, */dev/fd048ss9* indicates a 48 track per inch, single sided, 9 sector floppy disk device in drive 0.

The minor device numbers for floppy drives depend on the drive and media configuration. The most common are:

Drive	Minor Device Number							
	48tpi				96tpi		135tpi	
	ds/8	ds/9	ss/8	ss/9	ds/15	ds/8	ds/9	ds/18
0	12	4	8	0	52	44	36	60
1	13	5	9	1	53	45	37	61
2	14	6	10	2	54	46	38	62
3*								

* reserved for special, non-floppy devices connected to the floppy controller as unit #3.

The scheme for creating minor device numbers is as follows: when interpreted as a binary number, each bit of the minor device number represents some aspect of the device/media configuration.

For example, the minor device number for */dev/fd048ss8* is "8". Interpreted as a binary number, 8 is:

00001000

Only bits 1 through 6 are used in minor device identification:

Tracks per inch	Sectors per track		ss or ds	Drive number	
32	16	8	4	2	1
0	0	1	0	0	0

Drive number

Bits 1 and 2 describe the drive number:

Bits		Drive number
2	1	
0	0	0
0	1	1
1	0	2
1	1	3*

* reserved for special, non-floppy devices connected to the floppy controller as unit #3.

Single sided or double sided

Bit 3 describes whether the disk is single sided or double sided. If this bit is 0, the disk is single sided; if this bit is 1, the disk is double sided.

Sectors per track

Bits 4 and 5 describe sectors per track:

Bits		Sectors per track
16	8	
0	0	9
0	1	8
1	0	15
1	1	18

Tracks per inch

Bit 6 describes tracks per inch. If this bit is 0, the floppy has 46 tracks per inch; if this bit is 1, the floppy has either 96 tracks per inch or 135 tracks per inch. 96tpi disks can be distinguished from 135tpi disks by the number of sectors per track. 96tpi disks will have either 15 or 8 sectors per track, while 135tpi disks will have either 9 or 18 sectors per track, so 95tpi disks and 135tpi disks will always have different minor numbers.

Using this information, you can construct any minor device numbers you need.

UNIX devices

UNIX diskette device file names use the following format:

/dev/[r]dsk/f[0 | 1][5h | 5d9 | 5d8 | 5d4 | 5d16 | 5q | 3h | 3d][t | u]

where *r* indicates a raw (character) interface to the diskette, *rdsk* selects the raw device interface and *dsk* selects the block device interface. *0* or *1* selects the drive to be accessed: *f0* selects floppy drive 0, while *f1* selects drive 1. The following list describes the possible formats:

5h	5.25" high density diskette (1.2MB)
5d9	5.25" double density diskette, 9 sectors per track (360KB)
5d8	5.25" double density diskette, 8 sectors per track (320KB)
5d4	5.25" double density diskette, 4 sectors per track (320KB)
5d16	5.25" double density diskette, 16 sectors per track (320KB)
5q	5.25" high density diskette (720KB)
3h	3.50" high density diskette (1.44MB)
3d	3.50" double density diskette (720KB)

Format specification is mandatory when opening the device for formatting. However, when accessing a floppy disk for other operations (read and write), the format specification field can be omitted. In this case, the floppy disk driver will automatically determine the format previously established on the diskette and then perform the requested operation; for example,

cpio -itv</dev/rsdk

The last parameter, **t** or **u**, selects the partition to be accessed. **t** represents the whole diskette. Without **t** or **u** specified, the whole diskette except cylinder 0 will be selected. **u** represents the whole diskette except track 0 of cylinder 0.

Besides the device file naming convention described above, some of the formats have alias names that correlate to previous releases. The following list describes the formats that have an alias:

format	**alias**
5h	q15d
5d8	d8d
5d9	d9d

For example, the device file */dev/rdsk/f0q15dt* is equivalent to */dev/rdsk/f05ht*.

Files

XENIX Devices:

/dev/[r]fd0	*/dev/[r]fd048ss8*	*/dev/[r]fd096*	*/dev/[r]fd0135ds9*
/dev/[r]fd1	*/dev/[r]fd148ss8*	*/dev/[r]fd196*	*/dev/[r]fd1135ds9*
/dev/[r]fd048	*/dev/[r]fd048ds9*	*/dev/[r]fd096ds9*	*/dev/[r]fd0135ds18*
/dev/[r]fd148	*/dev/[r]fd148ds9*	*/dev/[r]fd196ds9*	*/dev/[r]fd1135ds18*
/dev/[r]fd048ds8	*/dev/[r]fd048ss9*	*/dev/[r]fd096ds15*	
/dev/[r]fd148ds8	*/dev/[r]fd148ss9*	*/dev/[r]fd196ds15*	

UNIX Devices:

/dev/[r]dsk/f0	*/dev/[r]dsk/f05d8*	*/dev/[r]dsk/f05q*
/dev/[r]dks/f0t	*/dev/[r]dsk/f05d8t*	*/dev/[r]dsk/f05qt*
/dev/[r]dsk/f05h	*/dev/[r]dsk/f05d4*	*/dev/[r]dsk/f03h*
/dev/[r]dsk/f05ht	*/dev/[r]dsk/f05d4t*	*/dev/[r]dsk/f03ht*
/dev/[r]dsk/f05d9	*/dev/[r]dsk/f05d16*	*/dev/[r]dsk/f03d*
/dev/[r]dsk/f005d9t	*/dev/[r]dsk/f05d16t*	*/dev/[r]dsk/f03dt*

Notes

It is not advisable to format a low density (48tpi) diskette on a high density (96tpi or 135tpi) floppy drive. Low density diskettes written on a high density drive should be read on high density drives. They may or may not be readable on a low density drive.

Use error-free floppy disks for best results on reading and writing.

hd

internal hard disk drive

Description

Block-buffered access to the **primary** hard disk is provided through the following block special files: *hd00*, *hd01* through *hd04*, *hd0a* and *hd0d*, *root*, and *swap*. Block-buffered access to the **secondary** hard disk is provided through the following block special files: *hd10*, *hd11* through *hd14*, *hd1a*.

hd00 refers to the entire physical disk; *hd01* through *hd04* refer to the fdisk partitions. *root* refers to the root file system; *swap* refers to the swap area; the block special files access the disks via the system's normal buffering mechanism and may be read and written without regard to the size of physical disk records.

Character special files follow the same naming convention as the block special files except that the character special file is prefaced with an "r" (for "raw" or character device). For example, the character special file referring to the entire physical disk is */dev/rhd00*.

The following are example names of the fixed disk partitions. Each partition can be accessed through a block interface, for example */dev/hd01*, or through a character (raw) interface, for example */dev/rhd01*.

The above devices follow the XENIX naming convention. Equivalent UNIX devices are found in the */dev/dsk* (character) and */dev/rdsk* (raw) directories. In the table that follows, both XENIX and UNIX devices are shown. XENIX devices extend only to disks located on the first controller; beyond this, the UNIX devices shown must be used.

Device File Names for Fixed Disks

First Controller		
Disk 1	Disk 2	Partition
/dev/hd00 /dev/rhd00	/dev/hd10 /dev/rhd10	entire disk
/dev/hd01 /dev/rhd01	/dev/hd11 /dev/rhd11	first partition
/dev/hd02 /dev/rhd02	/dev/hd12 /dev/rhd12	second partition
/dev/hd03 /dev/rhd03	/dev/hd13 /dev/rhd13	third partition
/dev/hd04 /dev/rhd04	/dev/hd14 /dev/rhd14	fourth partition
/dev/hd0a /dev/rhd0a	/dev/hd1a /dev/rhd1a	active partition
/dev/root /dev/rroot		root file system
/dev/swap /dev/rswap		swap area

Note that the *root* and *swap* files exist only for the root disk.

The following table lists the minor device number definitions for the hard disk special files, along with examples. Note that the block and character special devices share the same minor device definition. The minor device number definition is as follows: bits 7 and 6 denote physical drive, bits 5-3 denote virtual (**fdisk**) partition and bits 2-0 denote **divvy** partition.

Because some SCSI controllers support large numbers of physical drives, an extended minor device numbering scheme is provided for those devices requiring more than 256 minor numbers.

Each driver that uses extended minor numbers has a *base major number* which is used to refer to the driver through the extended minor numbers. The driver has a normal entry in the *mdevice* table for the base major number, plus one additional entry for each group of 256 minor devices that the driver uses.

Additional entries are identified by the "M" flag in the driver's device characteristics field. The "M" flag indicates that the entry corresponds to a *virtual major number* that points to a device driver that uses extended minor numbers. Therefore a very large disk may sometimes have two (or more) major numbers; the first will correspond to minor numbers 0-255, the second to minor devices 256-512, and so on.

The minor device bits for DOS disk special files are listed in a separate table.

Minor Device Bits

Phys. 7 6	Virtual 5 4 3	divvy 2 1 0	Device special file name	Description
0 0	0 0 0	0 0 0	/dev/hd00	whole PD 0
0 1	0 0 0	0 0 0	/dev/hd10	whole PD 1
1 0	0 0 0	0 0 0	/dev/dsk/4s0	whole PD 2
1 1	0 0 0	0 0 0	/dev/dsk/5s0	whole PD 3
0 0	0 0 1	1 1 1	/dev/hd01	PD 0, whole VD 1
0 0	0 1 0	1 1 1	/dev/hd02	PD 0, whole VD 2
0 0	0 1 1	1 1 1	/dev/hd03	PD 0, whole VD 3
0 0	1 0 0	1 1 1	/dev/hd04	PD 0, whole VD 4
0 0	1 0 1	1 1 1	/dev/hd0a	PD 0, whole active VD
0 0	1 0 1	0 0 0	/dev/root	PD 0, active virtual, DP 0
0 0	1 0 1	0 0 1	/dev/swap	PD 0, active virtual, DP 1
0 0	1 0 1	0 1 0	/dev/usr	PD 0, active virtual, DP 2
0 0	1 0 1	1 1 0	/dev/recover	PD 0, active virtual, DP 6
0 1	0 0 1	1 1 1	/dev/hd11	PD 1, whole VD 1
0 1	0 1 0	1 1 1	/dev/hd12	PD 1, whole VD 2
0 1	0 1 1	1 1 1	/dev/hd13	PD 1, whole VD 3
0 1	1 0 0	1 1 1	/dev/hd14	PD 1, whole VD 4
0 1	1 0 1	1 1 1	/dev/hd1a	PD 1, whole active VD
0 1	1 0 1	0 0 0	/dev/u0	PD 1, active virtual, DP 0*
0 1	1 0 1	0 0 1	/dev/u1	PD 1, active virtual, DP 1*
0 1	1 0 1	0 1 0	/dev/u2	PD 1, active virtual, DP 2*
1 0	0 0 1	1 1 1	/dev/dsk/2s1	PD 2, whole VD 1
1 0	0 1 0	1 1 1	/dev/dsk/2s2	PD 2, whole VD 2
1 0	0 1 1	1 1 1	/dev/dsk/2s3	PD 2, whole VD 3
1 0	1 0 0	1 1 1	/dev/dsk/2s4	PD 2, whole VD 4
1 0	1 0 1	1 1 1	/dev/dsk/2sa	PD 2, whole active VD
1 1	0 0 1	1 1 1	/dev/dsk/3s1	PD 3, whole VD 1
1 1	0 1 0	1 1 1	/dev/dsk/3s2	PD 3, whole VD 2
1 1	0 1 1	1 1 1	/dev/dsk/3s3	PD 3, whole VD 3
1 1	1 0 0	1 1 1	/dev/dsk/3s4	PD 3, whole VD 4
1 1	1 0 1	1 1 1	/dev/dsk/3sa	PD 3, whole active VD

Key VD = virtual drive PD = physical drive
DP = divvy partition * = user-defined name

The device files *usr* and *u[0-2]* are optional filesystem names; these nodes are not present unless created by the system administrator.

DOS Minor Device Bits

Phys. 7 6	Virtual 5 4 3	divvy 2 1 0	Device special file name	Description
0 0	1 1 0	0 0 0	/dev/dsk/0sC	PD 0, PDP, logical drive C
0 0	1 1 0	0 0 1	/dev/dsk/0sD	PD 0, EDP, logical drive D
0 0	1 1 0	0 1 0	/dev/dsk/0sE	PD 0, EDP, logical drive E
0 0	1 1 0	0 1 1	/dev/dsk/0sF	PD 0, EDP, logical drive F
0 0	1 1 0	1 0 0	/dev/dsk/0sG	PD 0, EDP, logical drive G
0 0	1 1 0	1 0 1	/dev/dsk/0sH	PD 0, EDP, logical drive H
0 0	1 1 0	1 1 0	/dev/dsk/0sI	PD 0, EDP, logical drive I
0 0	1 1 0	1 1 1	/dev/dsk/0sJ	PD 0, EDP, logical drive J
0 1	1 1 0	0 0 0	/dev/dsk/1sC	PD 1, PDP, logical drive C
0 1	1 1 0	0 0 1	/dev/dsk/1sD	PD 1, EDP, logical drive D
0 1	1 1 0	0 1 0	/dev/dsk/1sE	PD 1, EDP, logical drive E
0 1	1 1 0	0 1 1	/dev/dsk/1sF	PD 1, EDP, logical drive F
0 1	1 1 0	1 0 0	/dev/dsk/1sG	PD 1, EDP, logical drive G
0 1	1 1 0	1 0 1	/dev/dsk/1sH	PD 1, EDP, logical drive H
0 1	1 1 0	1 1 0	/dev/dsk/1sI	PD 1, EDP, logical drive I
0 1	1 1 0	1 1 1	/dev/dsk/1sJ	PD 1, EDP, logical drive J
1 0	1 1 0	0 0 0	/dev/dsk/2sC	PD 2, PDP, logical drive C
1 0	1 1 0	0 0 1	/dev/dsk/2sD	PD 2, EDP, logical drive D
1 0	1 1 0	0 1 0	/dev/dsk/2sE	PD 2, EDP, logical drive E
1 0	1 1 0	0 1 1	/dev/dsk/2sF	PD 2, EDP, logical drive F
1 0	1 1 0	1 0 0	/dev/dsk/2sG	PD 2, EDP, logical drive G
1 0	1 1 0	1 0 1	/dev/dsk/2sH	PD 2, EDP, logical drive H
1 0	1 1 0	1 1 0	/dev/dsk/2sI	PD 2, EDP, logical drive I
1 0	1 1 0	1 1 1	/dev/dsk/2sJ	PD 2, EDP, logical drive J
1 1	1 1 0	0 0 0	/dev/dsk/3sC	PD 3, PDP, logical drive C
1 1	1 1 0	0 0 1	/dev/dsk/3sD	PD 3, EDP, logical drive D
1 1	1 1 0	0 1 0	/dev/dsk/3sE	PD 3, EDP, logical drive E
1 1	1 1 0	0 1 1	/dev/dsk/3sF	PD 3, EDP, logical drive F
1 1	1 1 0	1 0 0	/dev/dsk/3sG	PD 3, EDP, logical drive G
1 1	1 1 0	1 0 1	/dev/dsk/3sH	PD 3, EDP, logical drive H
1 1	1 1 0	1 1 0	/dev/dsk/3sI	PD 3, EDP, logical drive I
1 1	1 1 0	1 1 1	/dev/dsk/3sJ	PD 3, EDP, logical drive J

Key PDP = primary DOS partition EDP = extended DOS partition

The device file */dev/hd0d* (first DOS partition) is linked to */dev/dsk/0sC* for backwards compatibility.

To access DOS partitions, specify letters such as "C:" or "D:" to indicate first or second partitions. The file */etc/default/msdos* contains lines that assign a letter abbreviation for the DOS device name. Refer to **dos**(C).

See also

badtrk(ADM), **divvy**(ADM), **dos**(C), **fdisk**(ADM), **mkdev**(ADM), **mknod**(C)

Diagnostics

The following messages are among those that may be printed on the console:

```
invalid fixed disk parameter table
```

and:

```
error on fixed disk (minor n), block = nnnnn,
cmd=nnnnn, status=nnnn,
Sector = nnnnn, Cylinder/head = nnnnn
```

Possible reasons for the first error include:

- The kernel is unable to get drive specifications, such as number of heads, cylinders, and sectors per track, from the disk controller ROM.
- Improper configuration.
- The disk is not turned on.
- The disk is not supported.

The second error specifies the following information:

- **block**: The UNIX block number within the device.
- **cmd**: The last command sent to the disk controller.
- **status**: The error status from the disk controller.
- **Sector** and **Cylinder/head** specify the location of a possible flaw. This information is used with **badtrk**(ADM).

Notes

On the first disk, *hd00* denotes the entire disk and is used to access the master boot block which includes the fdisk partition table. For the second disk, *hd10* denotes the entire disk and is used to access its fdisk partition table. Do not write to *hd10* and *hd00*.

keyboard

the PC keyboard

Description

The PC keyboard is used to enter data, switch screens, and send certain control signals to the computer. The operating system performs terminal emulation on the PC screen and keyboard, and, in doing so, makes use of several particular keys and key combinations. These keys and key combinations have special names that are unique to UNIX systems, and may or may not correspond to the keytop labels on your keyboard. These keys are described later.

When you press a key, one of the following happens:

- An ASCII value is entered.
- A string is sent to the computer.
- A function is initiated.
- The meaning of another key, or keys, is changed.

When a key is pressed (a keystroke), the keyboard sends a scancode to the computer. This scancode is interpreted by the keyboard driver. The interpretation of scancodes may be modified so that keys can function differently from their default actions.

There are three special occurrences, or keystrokes, which do the following:

- Switch screens
- Send signals
- Change the value of previous character, characters, or string

Switching screens (Multiscreen)

To get to the next consecutive screen, enter ⟨Ctrl⟩⟨PrtSc⟩. Any active screen may be selected by entering ⟨Alt⟩⟨F*n*⟩ where ⟨F*n*⟩ is one of the function keys. ⟨F1⟩ refers to the PC display (*/dev/tty01*).

Signals

A signal affects some process or processes. Examples of signals are ⟨Ctrl⟩**d** (end of input, exits from shell), ⟨Ctrl⟩**e** (quits a process), ⟨Ctrl⟩**s** (stop output to the screen), and ⟨Ctrl⟩**q** (resume sending output). Typically, characters are mapped to signals using **stty**(C).

Altering values

The actual code sent to the keyboard driver can be changed by using certain keys in combination. For example, the ⟨Shift⟩ key changes the ASCII values of the alphanumeric keys. Holding down the ⟨Ctrl⟩ key while pressing another

key sends a control code (⟨Ctrl⟩**d**, ⟨Ctrl⟩**s**, ⟨Ctrl⟩**q**, etc.).

Special keys

To help you find the special keys, the following table shows which keys on a typical console correspond to UNIX system keys. These are examples and can differ between shells, applications and so forth.

UNIX Name	Keytop	Action
INTR	⟨Del⟩	Stops current action and returns to the shell. This key is also called the RUB OUT or INTERRUPT key.
BACKSPACE	←	Deletes the first character to the left of the cursor. Note that the "cursor left" key also has a left arrow (←) on its keytop, but you cannot backspace using that key.
⟨Ctrl⟩**d**	⟨Ctrl⟩**d**	Signals the end of input from the keyboard; also exits current shell.
⟨Ctrl⟩**h**	⟨Ctrl⟩**h**	Deletes the first character to the left of the cursor. Also called the ERASE key.
⟨Ctrl⟩**q**	⟨Ctrl⟩**q**	Restarts printing after it has been stopped with ⟨Ctrl⟩**s**.
⟨Ctrl⟩**s**	⟨Ctrl⟩**s**	Suspends printing on the screen (does not stop the program).
⟨Ctrl⟩**u**	⟨Ctrl⟩**u**	Deletes all characters on the current line. Also called the KILL key.
⟨Ctrl⟩\	⟨Ctrl⟩\	Quits current command and creates a *core* file, if allowed. (Recommended for debugging only.)
ESCAPE	⟨Esc⟩	Special code for some programs. For example, changes from insert mode to command mode in the **vi**(C) text editor.
RETURN	(down-left arrow or ⟨Enter⟩)	Terminates a command line and initiates an action from the shell.

(Continued on next page)

(Continued)

UNIX Name	Keytop	Action
F*n*	F*n*	Function key *n*. ⟨F1⟩-⟨F12⟩ are unshifted, ⟨F13⟩-⟨F24⟩ are shifted ⟨F1⟩-⟨F12⟩, ⟨F25⟩-⟨F36⟩ are ⟨Ctrl⟩⟨F1⟩ through ⟨F12⟩, and ⟨F37⟩-⟨F48⟩ are ⟨Ctrl⟩⟨Shift⟩⟨F1⟩ through ⟨F12⟩. The next F*n* keys (⟨F49⟩-⟨F60⟩) are on the number pad (unshifted): ⟨F49⟩ - '7' ⟨F55⟩ - '6' ⟨F50⟩ - '8' ⟨F56⟩ - '+' ⟨F51⟩ - '9' ⟨F57⟩ - '1' ⟨F52⟩ - '-' ⟨F58⟩ - '2' ⟨F53⟩ - '4' ⟨F59⟩ - '3' ⟨F54⟩ - '5' ⟨F60⟩ - '0' For keys ⟨F61⟩ through ⟨F96⟩, see */usr/lib/keyboard/strings*. These function keys are not available on all keyboards, but you can map other keys to represent them.

The keyboard mapping is performed through a structure defined in */usr/include/sys/keyboard.h*. Each key can have ten states, generated by holding down the key itself and (optionally) one or more additional keys. The first eight are:

Base
⟨Shift⟩
⟨Ctrl⟩
⟨Alt⟩
⟨Ctrl⟩⟨Shift⟩
⟨Alt⟩⟨Shift⟩
⟨Alt⟩⟨Ctrl⟩
⟨Alt⟩⟨Ctrl⟩⟨Shift⟩

(Where "Base" indicates just the key on its own.)

There are two additional states indicated by two special bytes. The first is a "special state" byte whose bits indicate whether the key is "special" in one or more of the first eight states.

The second is one of four characters (**C**, **N**, **B**, **O**) which indicate how the lock keys affect the particular key. This is discussed further in the next section, "Scan codes".

Keyboard mode

Most keyboards normally are in a PC compatibility mode, though some can be put into a native AT keyboard mode. The UNIX utility **kbmode**(ADM) can be used to determine if a keyboard supports AT mode, and can also be used to put the keyboard into AT mode until the next time the system is rebooted. A system can also be configured to boot with the keyboard in AT mode with the **configure**(ADM) utility.

Enhanced keyboards are more programmable in AT mode. Also, two ⟨Ctrl⟩ keys and an ⟨Alt⟩ key can be recognized in AT mode.

Scan codes

The following table describes the default contents of */usr/lib/keyboard/keys*. The column headings are:

SCAN CODE The scan code generated by the keyboard hardware when a key is pressed. There is no user access to the scan code generated by releasing a key.

BASE The normal value of a key press.

SHIFT The value of a key press when the ⟨Shift⟩ key is also being held down.

LOCK Indicates which lock keys affect that particular key:

- **C** indicates ⟨Capslock⟩
- **N** indicates ⟨Numlock⟩
- **B** indicates both
- **O** indicates locking is off

Keys affected by the lock keys **C**, **B**, or **N** send the shifted value (scan code) of current state when that lock key is on. When the ⟨Shift⟩ key is depressed while a lock key is also on, the key reverts (toggles) to its original state.

The other columns are the values of key presses when combinations of the ⟨Ctrl⟩, ⟨Alt⟩ and ⟨Shift⟩ keys are also held down.

All values, except for keywords, are ASCII character values. The keywords refer to the special function keys.

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
0	nop	nop	nop	nop	nop	nop	nop	nop	O
1	esc	esc	nop	nop	esc	esc	nop	nop	O
2	'1'	'!'	nop	nop	'1'	'!'	nop	nop	O
3	'2'	'@'	nop	nop	'2'	'@'	nop	nop	O
4	'3'	'#'	nop	nop	'3'	'#'	nop	nop	O
5	'4'	'$'	nop	nop	'4'	'$'	nop	nop	O
6	'5'	'%'	nop	nop	'5'	'%'	nop	nop	O
7	'6'	'^'	rs	rs	'6'	'^'	rs	rs	O
8	'7'	'&'	nop	nop	'7'	'&'	nop	nop	O
9	'8'	'*'	nop	nop	'8'	'*'	nop	nop	O
10	'9'	'('	nop	nop	'9'	'('	nop	nop	O
11	'0'	')'	nop	nop	'0'	')'	nop	nop	O
12	'-'	'_'	ns	ns	'-'	'_'	ns	ns	O
13	'='	'+'	nop	nop	'='	'+'	nop	nop	O
14	bs	bs	del	del	bs	bs	del	del	O
15	ht	btab	nop	nop	ht	btab	nop	nop	O
16	'q'	'Q'	dc1	dc1	'q'	'Q'	dc1	dc1	C
17	'w'	'W'	etb	etb	'w'	'W'	etb	etb	C
18	'e'	'E'	enq	enq	'e'	'E'	enq	enq	C
19	'r'	'R'	dc2	dc2	'r'	'R'	dc2	dc2	C
20	't'	'T'	dc4	dc4	't'	'T'	dc4	dc4	C
21	'y'	'Y'	em	em	'y'	'Y'	em	em	C
22	'u'	'U'	nak	nak	'u'	'U'	nak	nak	C
23	'i'	'I'	ht	ht	'i'	'I'	ht	ht	C
24	'o'	'O'	si	si	'o'	'O'	si	si	C
25	'p'	'P'	dle	dle	'p'	'P'	dle	dle	C
26	'['	'{'	esc	esc	'['	'{'	esc	esc	O
27	']'	'}'	gs	gs	']'	'}'	gs	gs	O
28	cr	cr	nl	nl	cr	cr	nl	nl	O
29	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	ctrl	O
30	'a'	'A'	soh	soh	'a'	'A'	soh	soh	C
31	's'	'S'	dc3	dc3	's'	'S'	dc3	dc3	C
32	'd'	'D'	eot	eot	'd'	'D'	eot	eot	C
33	'f'	'F'	ack	ack	'f'	'F'	ack	ack	C
34	'g'	'G'	bel	bel	'g'	'G'	bel	bel	C
35	'h'	'H'	bs	bs	'h'	'H'	bs	bs	C
36	'j'	'J'	nl	nl	'j'	'J'	nl	nl	C
37	'k'	'K'	vt	vt	'k'	'K'	vt	vt	C
38	'l'	'L'	np	np	'l'	'L'	np	np	C
39	';'	':'	nop	nop	';'	':'	nop	nop	O
40	'\''	'"'	nop	nop	'\''	'"'	nop	nop	O
41	'`'	'~'	nop	nop	'`'	'~'	nop	nop	O
42	lshift	lshift	lshift	lshift	lshift	lshift	lshift	lshift	O

(Continued on next page)

(Continued)

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
43	'\\'	'|'	fs	fs	'\\'	'|'	fs	fs	O
44	'z'	'Z'	sub	sub	'z'	'Z'	sub	sub	C
45	'x'	'X'	can	can	'x'	'X'	can	can	C
46	'c'	'C'	etx	etx	'c'	'C'	etx	etx	C
47	'v'	'V'	syn	syn	'v'	'V'	syn	syn	C
48	'b'	'B'	stx	stx	'b'	'B'	stx	stx	C
49	'n'	'N'	so	so	'n'	'N'	so	so	C
50	'm'	'M'	cr	cr	'm'	'M'	cr	cr	C
51	','	'<'	nop	nop	','	'<'	nop	nop	O
52	'.'	'>'	nop	nop	'.'	'>'	nop	nop	O
53	'/'	'?'	nop	nop	'/'	'?'	nop	nop	O
54	rshift	rshift	rshift	rshift	rshift	rshift	rshift	rshift	O
55	'*'	'*'	nscr	nscr	'*'	'*'	nscr	nscr	O
56	alt	alt	alt	alt	alt	alt	alt	alt	O
57	' '	' '	' '	' '	' '	' '	' '	' '	O
58	clock	clock	clock	clock	clock	clock	clock	clock	O
59	fkey1	fkey13	fkey25	fkey37	scr1	scr11	scr1	scr11	O
60	fkey2	fkey14	fkey26	fkey38	scr2	scr12	scr2	scr12	O
61	fkey3	fkey15	fkey27	fkey39	scr3	scr13	scr3	scr13	O
62	fkey4	fkey16	fkey28	fkey40	scr4	scr14	scr4	scr14	O
63	fkey5	fkey17	fkey29	fkey41	scr5	scr15	scr5	scr15	O
64	fkey6	fkey18	fkey30	fkey42	scr6	scr16	scr6	scr16	O
65	fkey7	fkey19	fkey31	fkey43	scr7	scr7	scr7	scr7	O
66	fkey8	fkey20	fkey32	fkey44	scr8	scr8	scr8	scr8	O
67	fkey9	fkey21	fkey33	fkey45	scr9	scr9	scr9	scr9	O
68	fkey10	fkey22	fkey34	fkey46	scr10	scr10	scr10	scr10	O
69	nlock	nlock	dc3	dc3	nlock	nlock	dc3	dc3	O
70	slock	slock	del	del	slock	slock	del	del	O
71	fkey49	'7'	'7'	'7'	'7'	'7'	'7'	'7'	N
72	fkey50	'8'	'8'	'8'	'8'	'8'	'8'	'8'	N
73	fkey51	'9'	'9'	'9'	'9'	'9'	'9'	'9'	N
74	fkey52	'-'	'-'	'-'	'-'	'-'	'-'	'-'	N
75	fkey53	'4'	'4'	'4'	'4'	'4'	'4'	'4'	N
76	fkey54	'5'	'5'	'5'	'5'	'5'	'5'	'5'	N
77	fkey55	'6'	'6'	'6'	'6'	'6'	'6'	'6'	N
78	fkey56	'+'	'+'	'+'	'+'	'+'	'+'	'+'	N
79	fkey57	'1'	'1'	'1'	'1'	'1'	'1'	'1'	N
80	fkey58	'2'	'2'	'2'	'2'	'2'	'2'	'2'	N
81	fkey59	'3'	'3'	'3'	'3'	'3'	'3'	'3'	N
82	fkey60	'0'	'0'	'0'	'0'	'0'	'0'	'0'	N
83	del	'.'	del	del	del	del	del	del	N

(Continued on next page)

(Continued)

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
84	nop	nop	nop	nop	nop	nop	nop	nop	O
85	fkey11	fkey23	fkey35	fkey47	scr11	scr11	scr11	scr11	O
86	fkey12	fkey24	fkey36	fkey48	scr12	scr12	scr12	scr12	O

The following scan codes exist only for keyboards which support, and are in, native AT mode, rather than PC compatibility mode.

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
87	fkey11	fkey23	fkey35	fkey47	scr11	scr11	scr11	scr11	O
88	fkey12	fkey24	fkey36	fkey48	scr12	scr12	scr12	scr12	O
89	nop	nop	nop	nop	nop	nop	nop	nop	O
90	nop	nop	nop	nop	nop	nop	nop	nop	O
91	nop	nop	nop	nop	nop	nop	nop	nop	O
92	nop	nop	nop	nop	nop	nop	nop	nop	O
93	nop	nop	nop	nop	nop	nop	nop	nop	O
94	nop	nop	nop	nop	nop	nop	nop	nop	O
95	nop	nop	nop	nop	nop	nop	nop	nop	O
96	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	O
97	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O
98	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O
99	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	O
100	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	O
101	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	O
102	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	O
103	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	O
104	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	O
105	del	del	del	del	del	del	del	del	N
106	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	fkey54	O
107	nop	nop	nop	nop	nop	nop	nop	nop	O
108	nop	nop	nop	nop	nop	nop	nop	nop	O
109	nop	nop	nop	nop	nop	nop	nop	nop	O
110	nop	nop	nop	nop	nop	nop	nop	nop	O
111	nop	nop	nop	nop	nop	nop	nop	nop	O
112	nop	nop	nop	nop	nop	nop	nop	nop	O
113	nop	nop	nop	nop	nop	nop	nop	nop	O
114	nop	nop	nop	nop	nop	nop	nop	nop	O
115	nop	nop	nop	nop	nop	nop	nop	nop	O
116	nop	nop	nop	nop	nop	nop	nop	nop	O
117	nop	nop	nop	nop	nop	nop	nop	nop	O

(Continued on next page)

(Continued)

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK
118	nop	nop	nop	nop	nop	nop	nop	nop	O
119	nop	nop	nop	nop	nop	nop	nop	nop	O
120	nop	nop	nop	nop	nop	nop	nop	nop	O
121	nop	nop	nop	nop	nop	nop	nop	nop	O
122	nop	nop	nop	nop	nop	nop	nop	nop	O
123	nop	nop	nop	nop	nop	nop	nop	nop	O
124	nop	nop	nop	nop	nop	nop	nop	nop	O
125	nop	nop	nop	nop	nop	nop	nop	nop	O
126	nop	nop	nop	nop	nop	nop	nop	nop	O
127	nop	nop	nop	nop	nop	nop	nop	nop	O
128	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	O
129	ralt	ralt	ralt	ralt	ralt	ralt	ralt	ralt	O
130	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	fkey60	O
131	del	del	del	del	del	del	del	del	N
132	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	fkey49	O
133	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	fkey57	O
134	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	fkey51	O
135	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	fkey59	O
136	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O
137	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	fkey55	O
138	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	fkey50	O
139	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O
140	'/'	nop	nop	nop	'/'	nop	nop	nop	O
141	cr	cr	nl	nl	cr	cr	nl	nl	O

The next table lists the "value" of each of the special keywords used in */usr/lib/keyboard/keys* (and the preceding table). **mapkey**(M) places a "value" in the **ioctl**(S) buffer during key mapping. The keywords are only used in the scan code file (*/usr/lib/keyboard/keys*) for readability.

Name	Value	Meaning
nop	0	No operation — no action from keypress
lshift	2	Left-hand shift
rshift	3	Right-hand shift
clock	4	⟨Capslock⟩
nlock	5	⟨Numlock⟩
slock	6	⟨Scroll lock⟩
alt	7	⟨Alt⟩ key
btab	8	Back tab key — generates fixed sequence (ESC [Z)
ctrl	9	⟨Ctrl⟩ key
nscr	10	Switch to the next screen
scr1	11	Switch to screen #1
...		...
scr16	26	Switch to screen #16
fkey1	27	Function key #1
...		...
fkey96	122	Function key #96
rctl	128*	Right ⟨Ctrl⟩ Key
ralt	129*	Right ⟨Alt⟩ Key

* AT-style 101/102 key keyboard only.

This table lists names and decimal values that are interchangeable in the *mapkey* file. Names are used in place of numeric constants to make it easier to read the scan code table. Again, only the decimal values are placed in the **ioctl** buffer. These are taken from **ascii**(M).

Name	Value	Name	Value
nul	0	dc1	17
soh	1	dc2	18
stx	2	dc3	19
etx	3	dc4	20
eot	4	nak	21
enq	5	syn	22
ack	6	etb	23
bel	7	can	24
bs	8	em	25
ht	9	sub	26
nl	10	esc	27
vt	11	fs	28
np	12	gs	29
cr	13	rs	30
so	14	ns	31
si	15	del	127
dle	16		

Keyboard mapping

The PC keyboard is mapped as part of terminal emulation. This kind of mapping is performed only on the computer keyboard, not on remote terminals. Use **mapkey**(M) to change keyboard mapping. To change the mapping for individual channels (multiscreens), use **mapchan**(M).

Keyboard mapping can also be performed using **ioctl**(S). The syntax is the same as for string key mapping (see previous section).

For keyboard mapping, ***cmd*** is **GIO_KEYMAP** to display the current map, and **PIO_KEYMAP** puts the prepared buffer into place.

String key mapping

To map string (function) keys, use the **mapstr** (see **mapkey**(M)) utility. **mapstr** modifies the string mapping table where function keys are defined.

The string mapping table is an array of 512 bytes (**`typedef strmap_t`**) containing null-terminated strings that redefine the function keys. The first null-terminated string is assigned to the first string key, the second string to the second string key, and so on.

There is no limit to the length of any particular string as long as the whole table does not exceed 512 bytes, including nulls. Strings are made null by the introduction of extra null characters.

The following is a list of default function key values:

Default Function Key Values

Key Number	Function Key	Function
1	⟨F1⟩	ESC [M
2	⟨F2⟩	ESC [N
3	⟨F3⟩	ESC [O
4	⟨F4⟩	ESC [P
5	⟨F5⟩	ESC [Q
6	⟨F6⟩	ESC [R
7	⟨F7⟩	ESC [S
8	⟨F8⟩	ESC [T
9	⟨F9⟩	ESC [U
10	⟨F10⟩	ESC [V
11	⟨F11⟩	ESC [W
12	⟨F12⟩	ESC [X
13	⟨Shift⟩⟨F1⟩	ESC [Y
14	⟨Shift⟩⟨F2⟩	ESC [Z
15	⟨Shift⟩⟨F3⟩	ESC [a
16	⟨Shift⟩⟨F4⟩	ESC [b
17	⟨Shift⟩⟨F5⟩	ESC [c
18	⟨Shift⟩⟨F6⟩	ESC [d
19	⟨Shift⟩⟨F7⟩	ESC [e
20	⟨Shift⟩⟨F8⟩	ESC [f
21	⟨Shift⟩⟨F9⟩	ESC [g
22	⟨Shift⟩⟨F10⟩	ESC [h
23	⟨Shift⟩⟨F11⟩	ESC [i
24	⟨Shift⟩⟨F12⟩	ESC [j
25	⟨Ctrl⟩⟨F1⟩	ESC [k
26	⟨Ctrl⟩⟨F2⟩	ESC [l
27	⟨Ctrl⟩⟨F3⟩	ESC [m
28	⟨Ctrl⟩⟨F4⟩	ESC [n
29	⟨Ctrl⟩⟨F5⟩	ESC [o
30	⟨Ctrl⟩⟨F6⟩	ESC [p
31	⟨Ctrl⟩⟨F7⟩	ESC [q
32	⟨Ctrl⟩⟨F8⟩	ESC [r
33	⟨Ctrl⟩⟨F9⟩	ESC [s
34	⟨Ctrl⟩⟨F10⟩	ESC [t
35	⟨Ctrl⟩⟨F11⟩	ESC [u
36	⟨Ctrl⟩⟨F12⟩	ESC [v
37	⟨Ctrl⟩⟨Shift⟩⟨F1⟩	ESC [w
38	⟨Ctrl⟩⟨Shift⟩⟨F2⟩	ESC [x
39	⟨Ctrl⟩⟨Shift⟩⟨F3⟩	ESC [y
40	⟨Ctrl⟩⟨Shift⟩⟨F4⟩	ESC [z
41	⟨Ctrl⟩⟨Shift⟩⟨F5⟩	ESC [@
42	⟨Ctrl⟩⟨Shift⟩⟨F6⟩	ESC [[
43	⟨Ctrl⟩⟨Shift⟩⟨F7⟩	ESC [\
44	⟨Ctrl⟩⟨Shift⟩⟨F8⟩	ESC []

(Continued on next page)

Default Function Key Values
(Continued)

Key Number	Function Key	Function
45	⟨Ctrl⟩⟨Shift⟩⟨F9⟩	ESC [^
46	⟨Ctrl⟩⟨Shift⟩⟨F10⟩	ESC [_
47	⟨Ctrl⟩⟨Shift⟩⟨F11⟩	ESC ['
48	⟨Ctrl⟩⟨Shift⟩⟨F12⟩	ESC [{
49	⟨Home⟩	ESC [H
50	↑	ESC [A
51	⟨PgUp⟩	ESC [I
52	⟨-⟩	-
53	←	ESC [D
54	⟨5⟩	ESC [E
55	→	ESC [C
56	⟨+⟩	+
57	⟨End⟩	ESC [F
58	↓	ESC [B
59	⟨PgDn⟩	ESC [G
60	⟨Ins⟩	ESC [L

You can also map string keys using **ioctl**(S). The syntax is:

```
#include <sys/keyboard.h>
ioctl(fd,cmd,buf)
int fd, cmd;
char *buf;
...
```

Use this for string key mapping where ***cmd*** is **GIO_STRMAP** to display the string mapping table and **PIO_STRMAP** to put the new string mapping table in place.

Files

/usr/lib/keyboard/keys
/usr/lib/keyboard/strings

See also

configure(ADM), **kbmode**(ADM), **mapchan**(F), **mapchan**(M), **mapkey**(M), **multiscreen**(M), **scancode**(HW), **screen**(HW), **setkey**(C), **stty**(C)

log

interface to STREAMS error logging and event tracing

Description

log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes (.CM strerr ADM , **strace**(ADM)). **log** presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit **log** messages; and a subset of **ioctl**(S) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own **log** messages.

Kernel interface

log messages are generated within the kernel by calls to the function **strlog**:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in *sys/strlog.h* and *sys/log.h*. **mid** is the STREAMS module id number for the module or driver submitting the **log** message. **sid** is an internal sub-id number usually used to identify a particular minor device of a driver. **level** is a tracing level that allows for selective screening out of low priority messages from the tracer. **flags** are any combination of **SL_ERROR** (the message is for the error logger), **SL_TRACE** (the message is for the tracer), **SL_FATAL** (advisory notification of a fatal error), and **SL_NOTIFY** (request that a copy of the message be mailed to the system administrator). **fmt** is a **printf**(S) style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to **NLOGARGS** (currently 3) numeric or character arguments can be provided.

User interface

log is opened via the clone interface, */dev/log*. Each open of */dev/log* obtains a separate stream to **log**. In order to receive **log** messages, a process must first notify **log** whether it is an error logger or trace logger via a STREAMS **I_STR ioctl** call (see below). For the error logger, the **I_STR ioctl** has an `ic_cmd` field of **I_ERRLOG** with no accompanying data. For the trace logger, the **ioctl** has an `ic_cmd` field of **I_TRCLOG**, and must be accompanied by a data buffer containing an array of one or more **`struct trace_ids`** elements. Each **`trace_ids`** structure specifies an **mid**, **sid**, and **level** from which messages will be accepted. **strlog** will accept messages whose **mid** and **sid** exactly match those in the **`trace_ids`** structure, and whose level is less than or equal to the level given in the **`trace_ids`** structure. A value of -1 in any of the fields of the **`trace_ids`** structure indicates that any value is accepted for that field.

At most, one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the **ioctl** call, **log** will begin sending up messages subject to the restrictions noted above. These messages are obtained via the **getmsg**(S) system call. The control part of this message contains a **log_ctl** structure, which specifies the **mid**, **sid**, **level**, **flags**, time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of **log** messages can be determined.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic, some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by **NLOGARGS** words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to **log**, even if it is not an error or trace logger. The only fields of the **log_ctl** structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by **log** before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to **NLOGARGS**) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an **I_TRCLOG** or **I_ERRLOG** when a logging process of the given type already exists will result in the error **ENXIO** being returned. Similarly, **ENXIO** is returned for **I_TRCLOG ioctls** without any **trace_ids** structures, or for any unrecognized **I_STR ioctl** calls. Incorrectly formatted **log** messages sent to the driver by a user process are silently ignored (no error results).

Examples

Example of **I_ERRLOG** notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;              /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR0, &ioc);
```

Example of **I_TRCLOG** notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;            /* any sub-id will be allowed */
tid[1].ti_level = -1;          /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a log message (no arguments).

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

Files

/dev/log
sys/log.h
sys/strlog.h

See also

clone(HW), **getmsg**(S), **Intro**(S), **putmsg**(S), **strace**(ADM), **strerr**(ADM)

lp, lp0, lp1, lp2

line printer device interfaces

Description

The *lp0*, *lp1*, and *lp2* files provide access to the optional parallel ports of the computer. The *lp0* and *lp2* files provide access to parallel ports 1 and 2, respectively. The *lp1* file provides access to the parallel port on the monochrome adaptor.

Only one of *lp0* and *lp1* may be used on a given system. To access two parallel printers on a system, use either *lp0* or *lp1*, and *lp2*.

The minor device numbers of the lp special files are used to control the behavior of the parallel port driver. Three bits may be set in the minor device number, with the following effects:

Bit 7 Reset printer on each open.

Bit 6 Force polling, rather than interrupts and polling (as normal). This setting is provided for situations when the printer appears to be running slowly as a result of lost interrupts.

Bit 5 Tandy printer translation. This setting is provided to perform translation for driving Tandy printers. Firstly, all standard output post-processing is turned off. (This is equivalent to **stty -opost**.) Secondly, tabs and formfeeds are expanded at the driver level. (Tabs are expanded to 8-column tabstops; formfeeds are expanded to 66 lines per page, using carriage returns: the page length may be changed using an **ioctl**(S) call. Line count and printhead position are tracked to enable the driver to expand tabs and formfeeds correctly.) Thirdly, the character following a backspace is never translated or acted on in any way except to output it. (Tabs and formfeeds are not expanded; returns and newlines do not increment the line count and other characters do not increment the printhead position.)

This type of behavior is required to support certain Tandy printers. It is not required in any other circumstances, and should not normally be used.

Files

/dev/lp0?
/dev/lp1?
/dev/lp2?

where the *?* is optional and may be any one of the following:

i Reset on open

p Force polling

f Tandy printer translation

See also

lp(C), **lpadmin**(ADM), **lpsched**(ADM), **parallel**(HW)

Notes

The standard **lp** ports, *lp0*, *lp1*, and *lp2* send a printer initialization string the first time the file is opened after the system is booted.

Not all computers have an alternate parallel port slot.

mouse

system mouse

Description

UNIX supports mice attached directly to controller cards on the bus, mice attached to standard serial ports, and PS/2 keyboard mice. The command:

mkdev mouse

is used to configure a new mouse or to reconfigure an existing mouse.

Files

/dev/mouse	Directory for mouse-related special device files
/dev/mouse/bus[0-1]	Bus mouse device files
/dev/mouse/vpix[0-1]	VP/ix-mouse device files
/dev/mouse/kb0	Keyboard mouse device files
/dev/mouse/mp[0...]	Slave pseudo-mouse device files
/dev/mouse/pmp[0...]	Master pseudo-mouse device files
/etc/default/usemouse	Default map file for **usemouse**(C)
/usr/lib/event/devices	File containing device information for mice
/usr/lib/event/ttys	File listing ttys eligible to use mice
*/usr/lib/mouse/**	Alternate map files for **usemouse**(C)

See also

mkdev(ADM), **usemouse**(C)

"Adding mice and other graphic input devices" in the *System Administrator's Guide.*

Value added

mouse is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

parallel

parallel interface devices

Description

There are several parallel devices:

/dev/lp0 Main parallel adapter

/dev/lp1 Adapter on monochrome video card

/dev/lp2 Alternate parallel adapter (on appropriate machines)

It is not possible to have all three parallel devices on one system. Some AT computers allow the use of two parallel devices, */dev/lp2*, and either */dev/lp0* or */dev/lp1*.

If a parallel device fails to interrupt properly, the parallel driver enters "poll mode". Once interrupts are received from the device, the driver returns to its original mode.

The parallel driver delays a certain amount of time when a parallel device is closed. The amount of delay can affect printer performance, but is necessary to compensate for different sizes of printer buffers and printer speeds. For example, this command sets the delay on close to 1 second, specified in tenths of a second:

stty time 10< /dev/lp0

When given from a prompt, this command will only work if the port is open. It is recommended that a variation of this command be placed in the interface script used with the parallel device to achieve the same results:

stty time 10 0< &1

Notes

Parallel adapters on add-on cards will function, but switches must be set correctly. Some compatible computers have ports LP0 and LP1 reversed.

The **stty**(C) command for output processing is supported on a parallel device. **stty** options that have no effect on a parallel device are ignored and no error messages are displayed.

Usage

Usually invoked by **lp**(C), but can be written to directly.

Files

/dev/lp0
/dev/lp1
/dev/lp2

See also

lp(C), **lp**(HW), **lpadmin**(ADM), **lpsched**(ADM), **serial**(HW)

prf

operating system profiler

Description

The special file */dev/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 indicating 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 */dev/prf* is a pseudo-device with no associated hardware.

File

/dev/prf

See also

profiler(ADM)

ramdisk

memory block device

Description

The **ramdisk** device driver provides a block interface to memory. A ramdisk can be used like any other block device, including making it into a filesystem using **mkfs**(ADM). There are eight ramdisks available.

The characteristics of a ramdisk file are determined by its minor device number. The bits in the minor device number encode its size, longevity, and which of the eight possible ramdisks it is.

The three low-order bits of the minor device number determine which of the eight ramdisks is being accessed.

The next four bits of the minor device number determine the size of the ramdisk. The size of a ramdisk must be a power of 2, and must be at least 16K. Since 4 bits are available, there are 16 possible sizes, starting at 16K and doubling every time the size indicator is incremented, to produce possible sizes of 16K, 32K, 64K, and up.

The high-order bit is a longevity indicator. If set, memory is permanently allocated to that ramdisk, and can be deallocated only by rebooting the system. Permanent ramdisks can only be allocated by the super user. However, once a permanent ramdisk is allocated (by opening it), it can be read and written by anyone having the appropriate permissions on the ramdisk inode.

If clear, the ramdisk is deallocated when no processes have it open. An unmounted ramdisk will be deallocated immediately when it is closed. To create an easily removable, but semi-permanent ramdisk, use a separate process to keep the device open for as long as necessary.

Since a complete set of ramdisks (8) would consume 256 inodes if all possible minor numbers were used, only one example 16K ramdisk (*/dev/ram00*) is created when the system is installed. The system administrator can check this existing file to determine the major device number for any other required ramdisks. All ramdisks will use the same major device number.

The following table shows how the minor device number is constructed:

Example Minor Device Number Construction

Description	Longe-vity	Size (see next table)				Ram Disk No.			Minor Device Number
16K (#1) (Temporary)	0	0	0	0	0	0	0	1	1
16K (#1) (Permanent)	1	0	0	0	0	0	0	1	129
64K (#0) (Temporary)	0	0	0	1	0	0	0	0	16
512K (#7) (Permanent)	1	0	1	0	1	1	1	1	175

The contents of the size field and the corresponding ramdisk size is shown in the next table.

Size Bits				Ramdisk Size
0	0	0	0	16K
0	0	0	1	32K
0	0	1	0	64K
0	0	1	1	128K
0	1	0	0	256K
0	1	0	1	512K
0	1	1	0	1M
0	1	1	1	2M
1	0	0	0	4M
1	0	0	1	8M
1	0	1	0	16M
1	0	1	1	32M
1	1	0	0	64M
1	1	0	1	128M
1	1	1	0	256M
1	1	1	1	512M

To create a ramdisk, follow these steps:

1. *Create the device node.*

   You must first create the device that the ramdisk will reside on. It has the form:

   mknod *device_name* **[b | c] 31** *minor_device_number*

   "b" is for blocked devices and is the one you will use. "31" is the major number for this type of device. The minor number is derived from the table above. The minor number is the sum of the three attribute columns.

Longevity: permanent = 128 non-permanent = 0

Size:

16K = 0	128K = 24	1 Meg = 48	8 Meg = 72
32K = 8	256K = 32	2 Meg = 56	16 Meg = 80
64K = 16	512K = 40	4 Meg = 64	32 Meg = 88

Ramdisk number: 0 through 7
Note: There are only 8 devices available. Two different size devices may not share the same number.

For example, to create a 64K permanent ramdisk, the minor number could vary from 144 to 151. If the disk number was 1, the **mknod** command would be:

mknod /dev/ram64 b 31 145

2. *Make a filesystem.*

This creates a filesystem on the the ramdisk. In this example, **mkfs**(ADM) has the form:

mkfs ***device_name size_of_file_in_BSIZE_blocks***

In this example, the command to create a 64K filesystem would be:

mkfs /dev/ram64 64

3. *Mount the filesystem.*

This mounts the selected device on the specified mountpoint. It has the form:

mount ***device_name mount_point***

In order to mount the example 64K ramdisk on */mnt* the command would be:

mount /dev/ram64 /mnt

To make a filesystem on a non-permanent ramdisk, the device file must be held open between the **mkfs** and the **mount**(ADM) operations. Otherwise, the ramdisk is allocated at the start of the **mkfs** command, and deallocated at its end, before it can be mounted. Once the ramdisk is mounted, it is open until it is unmounted.

The following shell fragment shows one way to use **mkfs** on a non-permanent 512K ramdisk, and then mount it:

```
(       /etc/mkfs /dev/ram40 512
        /etc/mount /dev/ram40 /mnt
) < /dev/ram40
```

The procedure is executed in a sub-shell taking its standard input from */dev/ram40* in order to keep */dev/ram40* open between its creation and the time at which it is mounted.

Notes

Ramdisks must occupy contiguous memory. If free memory is fragmented, opening a ramdisk may fail even though there is enough total memory available. Ideally, all ramdisks should be allocated at system startup. This helps prevent the ramdisks themselves from fragmenting memory.

Ramdisks are geared towards use in specialized applications. In many cases, you will notice a **decrease in system performance** when ramdisks are used, because UNIX can generally put the memory to better use elsewhere.

File

/dev/ram00

See also

mkfs(ADM), **mknod**(C), **mount**(ADM)

Value added

ramdisk is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

rtc

real time clock interface

Description

The **rtc** driver supports the real time clock chip, allowing it to be set with the correct local time and allowing the time to be read from the chip.

ioctl calls

RTCRTIME This call is used to read the local time from the real time clock chip. The argument to the **ioctl**(S) is the address of a buffer of **RTC_NREG** unsigned characters (**RTC_NREG** is defined as *<sys/rtc.h>*). The **ioctl** will fill in the buffer with the contents of the chip registers. Currently, **RTC_NREG** is 14, and the meanings of the byte registers are as follows:

Register	Contents
0	Seconds
1	Second alarm
2	Minutes
3	Minute alarm
4	Hours
5	Hour alarm
6	Day of week
7	Date of month
8	Month
9	Year
A	Status register A
B	Status register B
C	Status register C
D	Status register D

For further information on the functions of these registers, see your hardware technical reference manual.

RTCSTIME This call is used to set the time into the real time clock chip. The argument to the **ioctl** is the address of a buffer of **RTC_NREGP** unsigned characters (**RTC_NREGP** as defined in *<sys/rtc.h>*). These bytes should be the desired chip register contents. Currently, **RTC_NREGP** is 10, representing registers 0-9 as shown above. Note that only the super user may open the real time clock device for writing and that the **RTCSTIME ioctl** will fail for any other than the super user.

File

/dev/rtc

scancode

PC-scancode capable terminal

Description

Like any terminal, a PC-scancode capable terminal is used to enter and display data. Unlike other terminals, which send information to the operating system only in the form of keytop values (the characters that appear on the faces of the keys), a PC-scancode capable terminal can also send scancodes (unique values associated with the depression and release of each key). The PC-scancode capable terminal sends scancodes to the **sio** driver when the terminal is set to scancode mode. When the terminal is in character mode, it sends keytop values.

Running a terminal in PC-scancode mode lets a scancode application access more distinct keystrokes than character mode would provide. For example, if you set your terminal to character mode and press the key labeled "A", your terminal sends a single value (the ASCII value of "a") to your application. However, if you set your terminal to scancode mode and press the key labeled "A", your terminal sends one value when you depress the key and a second value when you release the key. A scancode application translates these scancode values according to a predetermined map.

For more information on scancodes, including mapping tables, see **keyboard**(HW).

Line discipline ioctl calls

The line discipline **ioctl** calls control scancode settings on the device driver side.

The following **ioctl** calls and flags are defined in */usr/include/sys/termio.h*:

TCSETSC sets flags for a PC-scancode capable terminal. The argument values are:

- **KB_XSCANCODE** the device driver translates PC-scancodes to ASCII
- **KB_ISSCANCODE** the terminal device sends PC-scancodes

TCGETSC gets PC-scancode terminal flags.

The following **ioctl** calls are defined in */usr/include/sys/scankbd.h*:

KDGKBMAP gets the keyboard state bitmap (`sc_bitmap`). This bitmap is an array of eight unsigned longs that describe the current state of all keyboard keys.

AIOCSETSS sets the start/stop characters that the serial terminal sends. The argument to the **ioctl** call is of the type (**struct termss ***), defined in */usr/include/sys/termio.h*.

The following **ioctl** calls are available on a terminal device if the **KB_ISSCANCODE** flag is set on the device:

GIO_STRMAP gets mapping table for a function-key string. See **keyboard**(HW).

PIO_STRMAP puts mapping table for a function-key string. See **keyboard**(HW).

GIO_KEYMAP gets key-mapping table. See **keyboard**(HW).

PIO_KEYMAP puts key-mapping table. See **keyboard**(HW).

KDGKBSTATE returns the Shifted, Ctrl, or Alt state of the keyboard. Returns a Boolean combination of:

1 Shifted
2 Ctrl
4 Alt

KDGKBMODE gets keyboard translation mode (**K_RAW, K_XLATE**). Mode is returned where the ***arg*** parameter points.

KDSKBMODE sets keyboard translation mode (**K_RAW, K_XLATE**).

ioctl(S) can be used to define or obtain the current definition of a function key. The ***arg*** parameter of the **ioctl** call uses the **fkeyarg** data structure:

```
struct fkeyarg {
        unassigned int keynum;
        char keydef [MAXFK];
        /* Comes from ioctl.h via comcrt.h */
        char flen;
}
```

You can use the following **ioctl** requests to obtain or assign function key definitions:

GETFKEY gets the current definition of a function key. The function key number must be passed in `keynum`. The string currently assigned to the key is returned in `keydef`, and the length of the string is returned in `flen` when the **ioctl** call is performed.

SETFKEY assigns a given string to a function key. The function key number must be passed in `keydef`, and the length of the string (number of characters) must be passed in `flen`.

terminfo settings

The following *terminfo* database strings control scancode settings from the terminal side:

String	Definition
smsc	enters PC-scancode mode (string)
rmsc	leaves PC-scancode mode (string)
xonc	alternates **XON** character (int)
xoffc	alternates **XOFF** character (int)
dispc	displays a PC character (string)

Files

/usr/include/sys/comcrt.h
/usr/include/sys/ioctl.h
/usr/include/sys/sc_keys.h
/usr/include/sys/scancode.h
/usr/include/sys/scankbd.h
/usr/include/sys/termio.h

See also

ioctl(S), **keyboard**(HW), **mapstr**(C), **screen**(HW), **terminfo**(M), **tput**(C)

screen

tty [01-n], color, monochrome, ega, vga display adapter and video monitor

Description

The *tty* [*01-n*] device files provide character I/O between the system and the video display monitor and keyboard. Each file corresponds to a separate teletype device. Although there is a maximum of 12 screens, the exact number available (*n*) depends upon the amount of memory in the computer. The screens are modeled after a 25-line, 80-column ASCII terminal, unless specified otherwise.

System error messages from the kernel are written to */dev/console*, which is normally the current multiscreen. If the */dev/console* is the default output device for system error messages, and the display being used is switched to graphics mode, console messages are not displayed. When the video device returns to text mode, a notice message is displayed and the text of the kernel error can be recovered from *usr/adm/messages*.

Although all *tty*[*01-n*] devices may be open concurrently, only one of the corresponding devices can be active at any given time. The active device displays its own screen and takes sole possession of the keyboard. It is an error to attempt to access the *color*, *monochrome*, *ega*, or *vga* file when no corresponding adapter exists or no multiscreens are associated with it.

To get to the next consecutive screen, enter ⟨Ctrl⟩⟨PrtSc⟩. Any active screen may be selected by entering ⟨Alt⟩⟨F*n*⟩, where ⟨F*n*⟩ is one of the function keys. For example, ⟨F1⟩ refers to the *tty01* device.

Code examples are included in the section "Examples" to help programmers use the **ioctls** documented here.

Control modes

Multiscreens can be reassigned to different adapters (in multi-adapter systems) with these **ioctls** :

SWAPMONO Selects the monochrome display as the output device for the multiscreen.

SWAPCGA Selects the regular color display as the output device for the multiscreen.

SWAPEGA Selects the enhanced color display as the output device for the multiscreen.

SWAPVGA Selects the video graphics array color display as the output device for the multiscreen.

To find out which display adapter type is currently attached to the multiscreen, you can use **ioctl**(S) with the following request:

CONS_CURRENT Returns the display adapter type currently associated with the multiscreen. The return value can be one of: **MONO, CGA, EGA,** or **VGA.**

Display modes

The following **ioctls** can be used to change the video display mode:

SW_B80x25 Selects 80x25 black and white text display mode. (MONO, CGA, EGA, VGA)

SW_C80x25 Selects 80x25 color text display mode. (CGA, EGA, VGA)

SW_B40x25 Selects 40x25 black and white text display mode. (MONO, CGA, EGA, VGA)

SW_C40x25 Selects 40x25 color text display mode. (CGA, EGA, VGA)

SW_BG320 Selects 320x200 black and white graphics display mode. (CGA, EGA, VGA)

SW_CG320 Selects 320x200 color graphics display mode. (CGA, EGA, VGA)

SW_BG640 Selects 640x200 black and white graphics display mode. (CGA, EGA, VGA)

SW_EGAMONO80x25 Selects EGA (Enhanced Graphics Adapter) mode 7 - emulates support provided by the monochrome display. (EGA, VGA)

SW_EGAMONOAPA Selects EGA support for 640x350 graphics display mode EGA (mode F). (EGA with mono monitor)

SW_ENH_MONOAPA2 Selects EGA mode F*. (EGA with mono monitor)

SW_ENHB40x25 Selects enhanced EGA support for 40x25 black and white text display mode. (EGA, VGA)

SW_ENHC40x25 Selects enhanced EGA support for the 40x25 color text display mode. (EGA, VGA)

SW_ENHB80x25 Selects enhanced EGA support for 80x25 black and white text display mode. (EGA, VGA)

SW_ENHC80x25	Selects enhanced EGA support for 80x25 color text display mode. (EGA, VGA)
SW_ENHB80x43	Selects enhanced EGA support for 80x43 black and white text display mode. (EGA, VGA)
SW_ENHC80x43	Selects enhanced EGA support for 80x43 color text display mode. (EGA, VGA)
SW_CG320_D	Selects EGA support for 320x200 graphics display mode. (EGA mode D.) (EGA, VGA)
SW_CG640_E	Selects EGA support for 640x200 graphics display mode EGA (mode E). (EGA, VGA)
SW_CG640x350	Selects EGA support for 640x350 graphics display mode EGA (mode 10). (EGA, VGA)
SW_ENH_CG640	Selects EGA mode 10*. (EGA, VGA)
SW_MCAMODE	Reinitializes the monochrome adapter. (MONO)
SW_VGA40x25	Selects VGA support for the 40x25 color text display mode (VGA mode 1+). (VGA)
SW_VGA80x25	Selects VGA support for the 80x25 black and white text display mode (VGA mode 2+). (VGA)
SW_VGAM80x25	Selects VGA mode 7+ — emulates support provided by the monochrome display. (VGA with mono monitor)
SW_VGA11	Selects VGA support for the 640x480 graphics display mode (VGA mode 11). (VGA)
SW_VGA12	Selects VGA support for the 640x480 graphics display mode (VGA mode 12). (VGA)
SW_VGA13	Selects VGA support for the 320x200 graphics display mode (VGA mode 13). (VGA)

Switching to an invalid display mode for a display device will result in an error.

Getting display modes

The following **ioctl( )** requests are provided to obtain information about the current display modes:

CONS_GET	Returns the current display mode setting for current display adapter. (All)

CGA_GET Returns the current display mode setting of the color graphics adapter. (CGA only)

EGA_GET Returns the current display mode setting of the enhanced graphics adapter. (EGA only)

MCA_GET Returns the current display mode setting of the monochrome adapter. (MONO only)

VGA_GET Returns the current display mode of the video graphics adapters. (VGA only)

CONS_GETINFO Returns structure **vid_info** (below). Size of structure (first field) must be filled in by user.

```
struct vid_info
{
  short   size;            /* must be first field          */
  short   m_num;           /* multiscreen number, 0 based  */
  ushort  mv_row, mv_col;  /* cursor position              */
  ushort  mv_rsz, mv_csz;  /* text screen size             */
  struct  colors mv_norm,  /* normal attributes            */
                 mv_rev,   /* reverse video attributes     */
                 mv_grfc;  /* graphic character attributes */
  uchar_t mv_ovscan;       /* border color                 */
  uchar_t mk_keylock;      /* caps/num/scroll lock         */
};
```

CONS_6845INFO Returns structure **m6845_info** (below). Size of structure (first field) must be filled in by user.

```
struct m6845_info
{
  short   size;          /* must be first field       */
  ushort  screen_top;    /* offset of screen in video */
  ushort  cursor_type;   /* cursor shape              */
};
```

CONSADP Returns number of the multiscreen displayed on adaptor associated with that multiscreen.

GIO_ATTR Return value of **ioctl** is 6845-style attribute byte in effect.

GIO_COLOR Return value of **ioctl** is 0 or 2 depending on whether the device supports color.

GIO_SCRNMAP Gets the 256-byte screen map table, which is the mapping of ASCII values (0-256) onto the PC video ROM font characters (0-256). Note that control characters (ASCII values less than hex 20) have control functions and do not display ROM characters (for example, ^J is newline).

This is often used to map the low font values that normally correspond to ASCII control values to higher ASCII values, thus displaying the desired ROM characters.

PIO_SCRNMAP Puts the 256-byte screen map table (see **GIO_SCRNMAP**).

GIO_KEYMAP See **keyboard**(HW).

GIO_FONT8X*n* Gets font, where ***n*** is 8, 14, and 16. Argument is a pointer to a font table. Size of 8X8 font table is 8X256 bytes, 8X14 is 14X256 bytes, etc.

PIO_FONT8X*n* Puts font, where ***n*** is 8, 14, and 16. Argument is a pointer to a font table. Size of 8X8 font table is 8X256 bytes, 8X14 is 14X256 bytes, etc.

Memory mapping modes

The **ioctl**(S) routine is used to map the display memory of the various devices into the user's data space.

Note that the **MAP* ioctl**s map the memory associated with the current mode. You must put the adapter into the desired mode before performing mapping, or the pointers returned will not be appropriate. Refer to your hardware manual for details on various displays, adapters, and controllers.

These **ioctl**( ) requests can be used to map the display memory:

MAPCONS Maps the display memory of the adapter currently being used into the user's data space. (All)

MAPMONO Maps the monochrome adapter's display memory into the user's data space. (MONO only)

MAPCGA Maps the color adapter's display memory into the user's data space. (CGA only)

MAPEGA Maps the enhanced graphics adapter's display memory into the user's data space. (EGA only)

MAPVGA Maps the video graphics adapter's display memory into the user's data space. (VGA only)

For example, the following code can be used to acquire a pointer to the start of the user data space associated with the color graphics adapter display memory:

```
char *dp;
int retval;
        .
        .
        .
/* fd is a file descriptor for a multiscreen device */
retval = ioctl (fd, MAPCONS,0L);
dp = (char *) retval;
        .
        .
        .
```

Note that when the display memory is mapped into the user space, the adapter's m6845 start address registers are not set. The start address can be reset in two ways, so that the start address of the display memory corresponds to the upper left hand corner of the screen:

1. Switch modes with an **ioctl**(). (The "switch" can be to the present mode). See the "Display modes" section of this manual page.
2. Change the start address high and low addresses with the in-on-port/out-on-port **ioctl**().

The in-on-port/out-on-port **ioctl**() can also be used to determine the current value in the start address register, and then set up a pointer to the offset in the mapped-in data space.

MAP_CLASS Package **ioctl** that gives I/O privileges to an arbitrary list of ports and maps an arbitrary frame buffer into the user's address space identified by a string found in the **struct vidclass** `vidclasslist[]`. For example:

```
char *
ioctl(fd, MAP_CLASS, video_class_name)
char *video_class_name;
```

This returns a pointer to the frame buffer. See */etc/conf/pack.d/cn/class.h* for descriptions of the existing classes. Note that the link kit *must* be installed in order to find this file. (The *class.h* file is normally generated by **mkdev graphics**.)

EGA_IOPRIVL
VGA_IOPRIVL These add the list of I/O ports found on standard EGA and VGA cards into the process' TSS I/O permission bitmap. This allows the process to access the EGA I/O ports directly from user space with 386 **IN** and **OUT** instructions. (See sample code under "Examples".) I/O instructions executed in this manner are somewhat slower than I/O instructions executed when the I/O privilege level is raised to 3 (see instruction timings in Intel's *80386 Programmer's Reference Manual*.)

A process' I/O privilege level can be set, allowing for the faster execution of I/O instructions with the **sysi86( )** subfunction **V86SC_IOPL**:

sysi86 (SI86V86, V86SC_IOPL, 0x3000)

This sets the I/O privilege to 3 as described above. Only the super user can do this.

KDDISPTYPE This call returns display information to the user. The argument expected is the buffer address of a structure of type **kd_disparam** into which display information is returned to the user. The **kd_disparam** structure is defined as follows:

```
struct kd_disparam
       {
          long type;     /* display type */
          char *addr;    /* display memory address */
       }
```

Possible values for the `type` field include:

KD_MONO for the IBM monochrome display adapter

KD_HERCULES for the Hercules monochrome graphics adapter

KD_CGA for the IBM color graphics adapter

KD_EGA for the IBM enhanced graphics adapter

KD_VGA for the IBM video graphics adapter

KDDISPINFO Returns **struct kd_dispinfo**, which contains adapter type and physical address of frame buffer, as follows:

```
struct kd_dispinfo{
         char *vaddr;          /* memory address */
         paddr_t physaddr;     /* memory address */
         unsigned long size;   /* memory size */
}
```

KIOCSOUND Starts sound generation. Turns on sound. The *arg* is the period of the bell tone in units of 840.3 nanoseconds. A value of 0 turns off the sound. This is useful for generating tones while in graphics mode.

KDGETLED Gets keyboard LED status. The argument is a pointer to a character. The character will be filled with a Boolean combination of the following values:

0x10	Caps lock and Scroll lock are on
0x11	Scroll lock and Num lock are on
0x04	Scroll lock is on
0x02	Num lock is on
0x01	Caps lock is on

KDSETLED Sets keyboard LED status. The argument is a character whose value is the Boolean combination of the values listed under **"KDGETLED"**.

KDMKTONE (See **KIOCSOUND.**) The argument is a 32 bit value, with the lower 16 bits set to the frequency and the upper 16 bits set to the duration (in milliseconds).

KDSETMODE Sets console in text or graphics mode. The argument is of type integer, which should contain one of the following values:

KD_TEXT (sets console to text mode)
KD_GRAPHICS (sets console in graphics mode)

Note, the user is responsible for programming the color/graphics adapter registers for the appropriate graphical state.

KDGETMODE Gets current mode of console. Returns integer argument containing either **KD_TEXT** or **KD_GRAPHICS** as defined in the **KDSETMODE ioctl** description.

KDENABIO Enable in's and out's to video adaptor ports. No argument.

KDDISABIO Disable in's and out's to video adaptor ports. No argument.

KDGKBTYPE Always returns 0.

KIOCINFO Always returns 0x6B64.

VT_SETMODE Sets the virtual terminal mode. The argument is a pointer to a **vt_mode** structure, as defined below.

VT_GETMODE Determines what mode the active virtual terminal is currently in, either **VT_AUTO** or **VT_PROCESS**. The argument to the **ioctl** is the address of the following type of structure:

```
struct vt_mode {
      char     mode;   /* VT mode */
      char     waitv;  /* if !=0, vt hangs on writes when not active */
      short    relsig; /* signal to use for release request */
      short    acqsig; /* signal to use for display acquired */
      short    frsig;  /* signal to use for forced release */
      }

      #define  VT_AUTO       0x00    /* automatic VT switching */
      #define  VT_PROCESS    0x01    /* process controls switching */
```

The **vt_mode** structure will be filled in with the current value for each field.

VT_RELDISP Tells the virtual terminal manager whether the display has been released by the process.

0 release refused
1 release acknowledged
2 acquire acknowledged

VT_ACTIVATE Makes the multiscreen number specified in the argument the active multiscreen. The video driver will cause a switch to occur in the same manner as if a hotkey sequence had been typed at the keyboard. If the specified multiscreen is not open or does not exist, the call will fail and **errno** will be set to **EINVAL**.

Graphics adapter port I/O

You can use **ioctl**(S) to read or write a byte from or to the graphics adapter port. The *arg* parameter of the **ioctl** call uses the **port_io_arg** data structure:

```
struct port_io_arg {
        struct port_io_struct args[4];
};
```

As shown above, the **port_io_arg** structure points to an array of four **port_io_struct** data structures. The **port_io_struct** structure has the following format:

```
struct port_io_struct {
       char dir;    /* direction flag (in vs. out) */
       unsigned short port;   /* port address */
       char data;   /* byte of data */
};
```

You may specify one, two, three, or four of the **port_io_struct** structures in the array for one **ioctl** call. The value of `dir` can be either **IN_ON_PORT** to specify a byte being input to the graphics adapter port or **OUT_ON_PORT** to specify a byte being output to the graphics adapter port. `port` is an integer specifying the port address of the desired graphics adapter port. `data` is the byte of data being input or output as specified by the call.

If you are not using any of the **port_io_struct** structures, load the `port` with 0, and leave the unused structures at the end of the array. Refer to hardware manuals for port addresses and functions for the various adapters.

You can use the following **ioctl**(S) commands to input or output a byte on the graphics adapter port:

CONSIO Inputs or outputs a byte on the current graphics adapter port as specified. (All)

MCAIO Inputs or outputs a byte on the monochrome adapter port as specified. (MONO only)

CGAIO Inputs or outputs a byte on the color graphics adapter port as specified. (CGA only)

EGAIO Inputs or outputs a byte on the enhanced graphics adapter port as specified. (EGA only)

VGAIO Inputs or outputs a byte on the video graphics array adapter port as specified. (VGA only) To input a byte on any of the graphics adapter ports, load `dir` with **IN_ON_PORT** and load `port` with the port address of the graphics adapter. The byte input from the graphics adapter port will be returned in `data`.

To output a byte, load `dir` with **OUT_ON_PORT**, load `port` with the port address of the graphics adapter, and load `data` with the byte you want output to the graphics adapter port.

Function keys

ioctl(S) can be used to define or obtain the current definition of a function key. The ***arg*** parameter of the **ioctl** call uses the **fkeyarg** data structure:

```
struct fkeyarg {
        unsigned short keynum;
        char keydef [MAXFK];
        char flen;
}
```

You can use the following **ioctl**(S) request to assign function key definitions:

SETLOCKLOCK Toggles the ⟨Caps Lock⟩ and ⟨Num Lock⟩ keys to be either global to all the multiscreens, or local to each individual multiscreen. To make the ⟨Caps Lock⟩ global (its default), set the ***arg*** parameter to 1. To make the ⟨Caps Lock⟩ local to each screen, set the ***arg*** parameter to 0.

ANSI screen attribute sequences

The following character sequences are defined by ANSI X3.64-1979 and may be used to control and modify the screen display. Each ***n*** is replaced by the appropriate ASCII number (decimal) to produce the desired effect. The last column is for **termcap**(M) codes, where "n/a" means not applicable.

The use of 7 or 8 bit characters in the escape sequence is a valid invocation for each action defined. For example the ANSI ED command can be invoked via the "ESC[*n* J" (0x1b-0x5b-*n*-0x4a, 7 bit chars) sequence or the "CSI*n*J" (0x9b-*n*-0x4n, 8 bit chars) sequence.

ISO	Sequence	Action	Termcap Code
ED (Erase in Display)	CSI*n*J	Erases all or part of a display. **n=0**: erases from active position to end of display. **n=1**: erases from the beginning of display to active position. **n=2**: erases entire display.	cd
EL (Erase in Line)	CSI*n*K	Erases all or part of a line. **n=0**: erases from active position to end of line. **n=1**: erases from beginning of line to active position. **n=2**: erases entire line.	ce
ECH (Erase Character)	CSI*n*X	Erases ***n*** characters.	n/a
CBT (Cursor Backward Tabulation)	CSI*n*Z	Moves active position back ***n*** tab stops.	bt
SU (Scroll Up)	CSI*n*S	Scrolls screen up ***n*** lines, introducing new blank lines at bottom.	sf
SD (Scroll Down)	CSI*n*T	Scrolls screen down ***n*** lines, introducing new blank lines at top.	sr
CUP (Cursor Position)	CSI*m;n*H	Moves active position to location ***m*** (vertical) and ***n*** (horizontal).	cm
HVP (Horizontal & Vertical Position)	CSI*m;n*f	Moves active position to location ***m*** (vertical) and ***n*** (horizontal).	n/a

(Continued on next page)

(Continued)

ISO	Sequence	Action	Termcap Code
CUU (Cursor Up)	CSI*n*A	Moves active position up *n* number of lines.	up (ku)
CUD (Cursor Down)	CSI*n*B	Moves active position down *n* number of lines.	do (kd)
CUF (Cursor Forward)	CSI*n*C	Moves active position *n* spaces to the right.	nd (kr)
CUB (Cursor Backward)	CSI*n*D	Moves active position *n* spaces backward.	bs (kl)
HPA (Horizontal Position Absolute)	CSI*n*	Moves active position to column given by *n*.	n/a
HPR (Horizontal Position Relative)	CSI*n*a	Moves active position *n* characters to the right.	n/a
VPA (Vertical Position Absolute)	CSI*n*d	Moves active position to line given by *n*.	n/a
VPR (Vertical Position Relative)	CSI*n*e	Moves active position down *n* number of lines.	n/a
IL (Insert Line)	CSI*n*L	Inserts *n* new, blank lines.	al
ICH (Insert Character)	CSI*n*@	Inserts *n* blank places for *n* characters.	ic
DL (Delete Line)	CSI*n*M	Deletes *n* lines.	dl
DCH (Delete Character)	CSI*n*P	Deletes *n* number of characters.	dc

(Continued on next page)

(Continued)

ISO	Sequence	Action	Termcap Code
CPL (Cursor to Previous Line)	CSI*n*F	Moves active position to beginning of line, *n* lines up.	n/a
CNL (Cursor Next Line)	CSI*n*E	Moves active position to beginning of line, *n* lines down.	n/a
SGR (Select Graphic Rendition)	CSI*n*m	Character attributes, as summarized in the chart below. Up to three attributes can be specified in the form: CSI *n1; n2; n3* m (See list of parameters below.)	n/a
SM (Set Mode)	CSI2h	Locks keyboard. Ignores keyboard input until unlocked. Characters are not saved.	n/a
MC (Media Copy)	CSI2i	Sends screen to host. Current screen contents are sent to the application.	n/a
RM (Reset Mode)	CSI2l	Unlocks keyboard. Re-enables keyboard input.	n/a

Select Graphic Rendition (SGR) Chart

n	Meaning
0	all attributes off (normal display)
1	bold intensity (or light color)
4	underscore on (if hardware supports it)
5	blink on (if hardware supports it)
7	reverse video
8	sets blank (non-display)
10	selects the primary font
11	selects the first alternate font; lets ASCII characters less than 32 be displayed as ROM characters
12	selects a second alternate font; toggles high bit of extended ASCII code before displaying as ROM characters
30	black foreground
31	red foreground
32	green foreground
33	brown foreground
34	blue foreground
35	magenta foreground
36	cyan foreground
37	white foreground
38	enables underline option; white foreground with white underscore
39	disables underline option
40	black background
41	red background
42	green background
43	brown background
44	blue background
45	magenta background
46	cyan background
47	white background

Additional screen attribute sequences

Name	Sequence	Action	Termcap Code
n/a	CSI=*p;d*B	Sets the bell parameter to the decimal values of *p* and *d*. *p* is the period of the bell tone in units of 840.3 nanoseconds, and *d* is the duration of the tone in units of 100 milliseconds.	n/a
n/a	CSI=*s;e*C	Sets the cursor to start on scan-line *s* and end on scanline *e*.	n/a
n/a	CSI=*x*D	Turns on or off (*x*=1 or 0) the intensity of the background color.	n/a
n/a	CSI=*x*E	Sets or clears (*x*=1 or 0) the Blink vs. Bold background bit in the 6845 CRT controller.	n/a
n/a	CSI=*c*A	Sets overscan color to color *c*. *c* is a decimal value taken from Color Table above. (This sequence may not be supported on all hardware.)	n/a
n/a	CSI=*c*F	Sets normal foreground color to *c*. (*c* is a decimal parameter taken from Color Table.)	n/a
n/a	CSI=*c*G	Sets normal background. (See Color Table.)	n/a
n/a	CSI=*c*H	Sets reverse foreground. (See Color Table.)	n/a
n/a	CSI=*c*I	Sets reverse background. (See Color Table.)	n/a
n/a	CSI=*c*J	Sets graphic foreground. (See Color Table.)	n/a
n/a	CSI=*c*K	Sets graphic background. (See Color Table.)	n/a
n/a	CSI*ng*	Accesses alternate graphics set. Not the same as "graphics mode". Refer to your hardware manual for decimal/character codes (*Pn*) and possible output characters.	n/a

(Continued on next page)

(Continued)

Name	Sequence	Action	Termcap Code
n/a	CSI*n*L	Fills new regions with current (*n*=0) or normal (*n*=1) attributes. Default is 0.	n/a
n/a	CSI*n*M	Returns current foreground color attributes, with *n*=0 for normal, 1 for reverse, and 2 for graphic. The colors are sent back in the keyboard data input stream as text decimal values separated by a space and terminated with a newline. For example, if the current foreground color is lt_red on black, "12 0\n" is returned.	n/a
n/a	CSIs	Saves current cursor position.	n/a
n/a	CSIu	Restores saved cursor position.	n/a
n/a	ESC7	Saves current cursor position.	n/a
n/a	ESC8	Restores saved cursor position.	n/a
n/a	ESCQ *Fn'string'*	Defines function key *Fn* with *string*. String delimiters ' and ' may be any character not in *string*. *Fn* is defined as the key number starting at zero plus the ASCII value of zero. For example, ⟨F1⟩ = 0... ⟨F16⟩ = " ? ", and so on. In this escape sequence, the " ^ " character will cause the next character to have 32 subtracted from its ASCII value. Thus " ^! " results in a SOH (^A) character.	n/a
n/a	CSI*n*z	Switches to screen *n*. If the screen does not exist, no action will take place.	n/a

Color Table

Cn	Color	Cn	Color
0	Black	8	Grey
1	Blue	9	Lt. Blue
2	Green	10	Lt. Green
3	Cyan	11	Lt. Cyan
4	Red	12	Lt. Red
5	Magenta	13	Lt. Magenta
6	Brown	14	Yellow
7	White	15	Lt. White

Examples

The following module includes useful examples of such operations as getting the display mode, screen switching, I/O privilege, and memory mapping.

Sample code (part 1 of 4)

```
#include <stdio.h>

#include <sys/types.h>
#include <sys/signal.h>
#include <sys/vtkd.h>

#define SIG_REL SIGUSR1
#define SIG_ACQ SIGUSR2

int Isdisplayed;                /* flag: when are we flipped away */
char *Screenmem;                /* physical map to the video RAM  */
int graf();                     /* Set everything up              */
void grafend();                 /* Restore user's text mode       */
void grafquit();                /* Clean-up and exit              */
void rel_screen(), acq_screen();
int Oldmode;                    /* save mode of user shell screen */

/*
 * Set up the graphics multiscreen stuff and call another
 * routine to set up card.
 */
graf()
{
        struct vt_mode smode;

        Isdisplayed = 1;
/*
 *  Set up to catch the screen switch signals.
 */
        signal(SIG_REL, rel_screen);
        signal(SIG_ACQ, acq_screen);
/*
 * Set up the data structure that asks the driver
 * to send you signals when the screens are switched.
 * mode == VT_PROCESS means send screen switch signals.
```

```
 * mode == VT_AUTO means turn off screen switch signals (regular mode).
 * relsig == the signal you want when the user switches away.
 * acqsig == the signal you want when the user switches back to you.
 */
        smode.mode = VT_PROCESS;
        smode.waitv = 0;        /* not implemented, reserved */
        smode.relsig = SIG_REL;
        smode.acqsig = SIG_ACQ;
        smode.frsig  = SIGINT;  /* not implemented, reserved */

        if(-1 == ioctl(0, VT_SETMODE, &smode))
        {
                perror("screen switch signal ioctl VT_SETMODE");
                exit(1);
        }
        signal(SIGINT, grafquit);
        grafmode();
}
```

Sample code (part 2 of 4)

```
/*
 * this is the signal handler for when the user screen flips
 * away from us.
 */
void
rel_screen()
{
        signal(SIG_REL, rel_screen);
        Isdisplayed = 0;
        ega_save();
/*
 * Tell the video driver that you have saved your state
 * and it can now have the card to switch to the new screen.
 * The video driver waits (forever) for this ioctl before
 * it will complete the screen switch requested by the user.
 * If you don't make this ioctl the screen switcher will
 * be wedged until it gets one.  It is best to have a
 * small one line reldisp.c program to unwedge your screen
 * switcher when development programs screw up from time
 * to time.
 */
        ioctl(0, VT_RELDISP, VT_TRUE);
}

/*
 * this is the signal handler for when the user screen flips
 * back to us.
 */
void
acq_screen()
{
        signal(SIG_ACQ, acq_screen);
        Isdisplayed = 1;
        ega_restore();
/*
```

```
 * Tell the video driver that you have restored your state
 * and screen switching can now continue.
 */
        ioctl(0, VT_RELDISP, VT_ACKACQ);
}

void
grafquit()
{
        grafend();
        exit(0);
}
```

Sample code (3 of 4)

```
/*
 * restore text mode.
 */
void
grafend()
{
        ioctl(0, MODESWITCH | Oldmode, (char *)0);
}

grafmode()
{
        int adapter, privlcmd;
/*
 * Confirm that we are on a supported video adapter.
 */
        adapter = ioctl(0, CONS_CURRENT, (char *)0);
        if(EGA != adapter && VGA != adapter)
        {
                puts("Stdin must be an EGA or VGA multiscreen\n");
                exit(0);
        }
```

Sample code (4 of 4)

```
/*
 * Save the user's current text mode so you
 * can restore it on exit.
 */
        Oldmode = ioctl(0, CONS_GET, (char *)0);
/*
 * Get privilege to do direct INs and OUTs to the video card.
 */
        if(EGA == adapter)
                privlcmd = EGA_IOPRIVL;
        else
                privlcmd = VGA_IOPRIVL;
        if(-1 == ioctl(0, privlcmd, 1))
        {
                perror("I/O privilege denied");
                exit(1);
        }
/*
 * Have the video driver reprogram the card for EGA 640x350 16 color mode.
 */
        ega_grafmode();
/*
 * Map the video card's frame buffer into your address space.
 * This must be done after the mode switch command or you get
 * frame buffer address for the wrong mode mapped in.
 */
        Screenmem = (char *)ioctl(0, MAPCONS, (char *)0);
}
```

Files

/dev/console
/dev/tty[02-n]
/dev/color
/dev/monochrome
/dev/ega
/dev/vga

Include files:
/usr/include/sys/vtkd.h
/usr/include/sys/comcrt.h
/usr/include/sys/console.h
/usr/include/sys/kb.h

See also

console(M), **ioctl**(S), **keyboard**(HW), **keymap**(M), **mapkey**(M), **mapchan**(M), **multiscreen**(M), **scancode**(HW), **setcolor**(C), **stty**(C), **systty**(M), **vidi**(C), **termcap**(M), **tty**(M)

scsi

small computer systems interface

Description

SCSI provides a standard interface to peripherals such as hard disks, printers, tape drives and others. SCSI is run via a host adapter card that can support up to 8 controllers, each supporting up to 8 devices. The minor device numbering scheme for SCSI disk devices is the same as the standard minor device number scheme for non-SCSI disk devices.

Each SCSI controller has its own major device number.

See also

hd(HW), **mkdev**(ADM), **tape**(HW)

System Administrator's Guide

Value added

scsi is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.

serial: tty1[a-h] , tty1[A-H] , tty2[a-h] , tty2[A-H]

interface to serial ports

Description

The *tty1[a-h], tty1[A-H], tty2[a-h]* and *tty2[A-H]* files provide access to the standard and optional serial ports of the computer. Each file corresponds to one of the serial ports (with or without modem control). Files are named according to the following conventions:

- The first number in the filename corresponds to the COM expansion slot.
- Lowercase letters indicate no modem control.
- Uppercase letters indicate the line has modem control.

tty1a and *tty1A* both refer to COM1, whereas *tty2a* and *tty2A* both refer to COM2.

For example, with a four-port expansion board installed at COM1 and a single port board installed at COM2, you can access:

tty1a	*tty1A*
tty1b	*tty1B*
tty1c	*tty1C*
tty1d	*tty1D*
tty2a	*tty2A*

Each serial port has modem and non-modem invocations. The device names in the following table refer to the serial ports, with and without modem control. The first section of the table describes boards at COM1 and the second section describes boards installed at COM2. "Minor" is the minor device number for the port (see **mknod**(C)).

Serial Lines				
Board Type	Non-Modem Control		Modem Control	
	Minor	*Name*	*Minor*	*Name*
1 Port	0	tty1a	128	tty1A
4 Port	1	tty1b	129	tty1B
	2	tty1c	130	tty1C
	3	tty1d	131	tty1D
8 Port	4	tty1e	132	tty1E
	5	tty1f	133	tty1F
	6	tty1g	134	tty1G
	7	tty1h	135	tty1H
1 Port	8	tty2a	136	tty2A
4 Port	9	tty2b	137	tty2B
	10	tty2c	138	tty2C
	11	tty2d	139	tty2D
8 Port	12	tty2e	140	tty2E
	13	tty2f	141	tty2F
	14	tty2g	142	tty2G
	15	tty2h	143	tty2H
16 Port	16	tty2i	144	tty2I
	17	tty2j	145	tty2J
	18	tty2k	146	tty2K
	19	tty2l	147	tty2L
	20	tty2m	148	tty2M
	21	tty2n	149	tty2N
	22	tty2o	150	tty2O
	23	tty2p	151	tty2P

Interrupt Vectors:

All board(s) installed at COM1 - 4
All board(s) installed at COM2 - 3

For a list of I/O addresses, see the *Release Notes.*

Access

The files may only be accessed if the corresponding serial interface card is installed and its jumper I/O address correctly set. Also, for multi-port expansion cards, you must use the **mkdev**(ADM) program to create more than the default number of files. Unless other COM slots are specifically referred to in your hardware documentation, only COM1 and COM2 may be used.

The serial ports must also be defined in the system configuration. Check your hardware manual to determine how your system is configured, via a CMOS database or by switch settings on the main system board. If your system is configured using a CMOS database, the ports are defined in the database (see **cmos**(HW)). Otherwise, define the ports by setting the proper switches on the main system board. Refer to your computer hardware manual for switch settings.

It is an error to attempt to access a serial port that has not been installed and defined.

The serial ports can be used for a variety of serial communication purposes such as connecting login terminals to the computer, attaching printers, or forming a serial network with other computers. Note that a serial port may operate at most of the standard baud rates, and that the ports (on most computers) have a DTE (Data Terminal Equipment) configuration. The following table defines how each pin is used for 25-pin and 9-pin connections:

25-Pin	9-Pin	Description
2	2	Transmit Data
3	3	Receive Data
4	7	Request to Send
5	8	Clear to Send
7	5	Signal Ground
8	1	Carrier Detect (Data Set Ready)
20	4	Data Terminal Ready

Only pins 2, 3, and 7 (2, 3, and 5 for 9-pin) are necessary for a terminal (or direct) connection.

A modem control device (port) uses pins 2, 3, and 7 in the same way as a non-modem control device: send on pin 2 and receive on pin 3. Pin 7 is data ground. On a non-modem control device, pins 4 and 20 (RTS and DTR) are asserted, but pin 8 is not. On a modem control device, pins 4 and 20 (RTS & DTR) are asserted and the port will not open until pin 8 (CXD) is asserted. That is, no signal travels from pin 2 until pin 8 is asserted from another source. The modem control device monitors the status of pin 8.

Files

/dev/tty1[a-h]
/dev/tty1[A-H]
/dev/tty2[a-h]
/dev/tty2[A-H]

See also

cmos(HW), **csh**(C), **cu**(C), **getty**(ADM), **mkdev**(ADM), **mknod**(C), **nohup**(C), **open**(S), **termio**(M), **tty**(M), **uucp**(C)

"Administering terminals" in the *System Administrator's Guide.*

Notes

If you log in via a modem control serial line, hanging up logs that line out and kills your background processes. See **nohup**(C) and **csh**(C).

You cannot use the same serial port with both modem and non-modem control at the same time. For example, you cannot use *tty1a* and *tty1A* simultaneously.

Use a modem cable to connect your modem to a computer.

streamio

STREAMS ioctl commands

Syntax

#include <stropts.h>
int ioctl
(fildes, command, arg)
int
fildes, command;

Description

STREAMS ioctl commands are a subset of **ioctl**(S) system calls which perform a variety of control functions on streams. The arguments ***command*** and ***arg*** are passed to the file designated by ***fildes*** and are interpreted by the stream head. Certain combinations of these arguments may be passed to a module or driver in the stream.

fildes is an open file descriptor that refers to a stream. ***command*** determines the control function to be performed as described below. ***arg*** represents additional information that is needed by this command. The type of ***arg*** depends upon the command, but it is generally an integer or a pointer to a ***command***-specific data structure.

Since these STREAMS commands are a subset of **ioctl**, they are subject to the errors described there. In addition, the call will fail with **errno** set to **EINVAL**, without processing a control function, if the stream referenced by ***fildes*** is linked below a multiplexer, or if ***command*** is not a valid value for a stream.

Also, as described in **ioctl**, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the stream head containing an error value. This causes subsequent system calls to fail with **errno** set to this value.

Command functions

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by ***arg*** onto the top of the current stream, just below the stream head. It then calls the open routine of the newly-pushed module. On failure, **errno** is set to one of the following values:

- **[EINVAL]** Invalid module name.
- **[EFAULT]** ***arg*** points outside the allocated address space.
- **[ENXIO]** Open routine of new module failed.
- **[ENXIO]** Hangup received on ***fildes***.

I_POP Removes the module just below the stream head of the stream pointed to by ***fildes***. ***arg*** should be 0 in an **I_POP** request. On failure, **errno** is set to one of the following values:

[EINVAL] No module present in the stream.

[ENXIO] Hangup received on ***fildes***.

I_LOOK Retrieves the name of the module just below the stream head of the stream pointed to by ***fildes***, and places it in a null terminated character string pointed at by ***arg***. The buffer pointed to by ***arg*** should be at least **FMNAMESZ** bytes long. An [**#include** <*sys/conf.h*>] declaration is required. On failure, **errno** is set to one of the following values:

[EFAULT] ***arg*** points outside the allocated address space.

[EINVAL] No module present in stream.

I_FLUSH This request flushes all input and/or output queues, depending on the value of ***arg***. Legal ***arg*** values are:

FLUSHR Flush read queues.

FLUSHW Flush write queues.

FLUSHRW Flush read and write queues.

On failure, **errno** is set to one of the following values:

[ENOSR] Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.

[EINVAL] Invalid ***arg*** value.

[ENXIO] Hangup received on ***fildes***.

I_SETSIG Informs the stream head that the user wishes the kernel to issue the **SIGPOLL** signal (see **signal**(S) and **sigset**(S)) when a particular event has occurred on the stream associated with ***fildes***. **I_SETSIG** supports an asynchronous processing capability in STREAMS. The value of ***arg*** is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT A non-priority message has arrived on a stream head read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.

S_HIPRI A priority message is present on the stream head read queue. This is set even if the message is of zero length.

S_OUTPUT The write queue just below the stream head is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.

S_MSG A STREAMS signal message that contains the **SIGPOLL** signal has reached the front of the stream head read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value **S_HIPRI**.

Processes that wish to receive **SIGPOLL** signals must explicitly register to receive them using **I_SETSIG**. If several processes register to receive this signal for the same event on the same stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals. On failure, **errno** is set to one of the following values:

[EINVAL] *arg* value is invalid or *arg* is zero and process is not registered to receive the **SIGPOLL** signal.

[EAGAIN] Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above. On failure, **errno** is set to one of the following values:

[EINVAL] Process not registered to receive the **SIGPOLL** signal.

[EFAULT] *arg* points outside the allocated address space.

I_FIND Compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present. On failure, **errno** is set to one of the following values:

[EFAULT] *arg* points outside the allocated address space.

[EINVAL] *arg* does not contain a valid module name.

I_PEEK Allows a user to retrieve the information in the first message on the stream head read queue without taking the message off the queue. *arg* points to a **strpeek** structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The "maxlen" field in the **ctlbuf** and **databuf strbuf** structures (see **getmsg**(S)) must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets `flags` to **RS_HIPRI**, **I_PEEK** will only look for a priority message on the stream head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the stream head read queue, or if the **RS_HIPRI** flag was set in `flags` and a priority message was not present on the stream head read queue. It does not wait for a message to arrive. On return, `ctlbuf` specifies information in the control buffer, `databuf` specifies information in the data buffer, and `flags` contains the value 0 or **RS_HIPRI**. On failure, **errno** is set to one of the following values:

[EFAULT] *arg* points, or the buffer area specified in `ctlbuf` or `databuf` is, outside the allocated address space.

[EBADMSG] Queued message to be read is not valid for **I_PEEK**

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM Byte-stream mode, the default.

RMSGD Message-discard mode.

RMSGN Message-nondiscard mode.

Read modes are described in **read**(S). On failure, **errno** is set to the following value:

[EINVAL] *arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in **read**(S). On failure, **errno** is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the stream head read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the stream head read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, **errno** is set to the following value:

[EFAULT] *arg* points outside the allocated address space.

I_FDINSERT Creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a **strfdinsert** structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fildes;
int             offset;
```

The "len" field in the **ctlbuf strbuf** structure (see **putmsg**(S)) must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. `fildes` in the **strfdinsert** structure specifies the file descriptor of the other stream. `offset`, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer. This pointer will be the address of the read queue structure of the driver for the stream corresponding to `fildes` in the **strfdinsert** structure. The "len" field in the **databuf strbuf** structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

`flags` specifies the type of message to be created. A non-priority message is created if `flags` is set to 0, and a priority message is created if `flags` is set to **RS_HIPRI**. For non-priority messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For priority messages, **I_FDINSERT** does not block on this condition. For non-priority messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** is set. Instead, it fails and sets **errno** to **EAGAIN**.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the stream, regardless of priority or whether **O_NDELAY** has been specified. No partial message is sent. On failure, **errno** is set to one of the following values:

[EAGAIN] A non-priority message was specified, the **O_NDELAY** flag is set, and the stream write queue is full due to internal flow control conditions.

[ENOSR] Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified in `ctlbuf` or `databuf` is, outside the allocated address space.

[EINVAL] One of the following: `fildes` in the **strfdinsert** structure is not a valid, open stream file descriptor; the size of a pointer plus `offset` is greater than the `len` field for the buffer specified through `ctlptr`; `offset` does not specify a properly aligned location in the data buffer; an undefined value is stored in `flags`.

[ENXIO] Hangup received on `fildes` of the **ioctl** call or `fildes` in the **strfdinsert** structure.

[ERANGE] The "len" field for the buffer specified through `databuf` does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the "len" field for the buffer specified through `databuf` is larger than the maximum configured size of the data part of a message, or the "len" field for the buffer specified through `ctlbuf` is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the stream head of the stream corresponding to `fildes` in the **strfdinsert** structure. In this case, **errno** will be set to the value in the message.

I_STR Constructs an internal STREAMS **ioctl** message from the data pointed to by ***arg*** and sends that message downstream.

This mechanism is provided to send user **ioctl** requests to downstream modules and drivers. It allows information to be sent with the **ioctl** and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgment message or until the request "times out". If the request times out, it fails with **errno** set to **ETIME**.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the stream head. The default timeout interval for these requests is 15 seconds. The **O_NDELAY** (see **open**(S)) flag has no effect on this call.

To send requests downstream, ***arg*** must point to a **strioctl** structure which contains the following members:

```
int     ic_cmd;         /* downstream command */
int     ic_timout;      /* ACK/NAK timeout */
int     ic_len;         /* length of data arg */
char    *ic_dp;         /* ptr to data arg */
```

`ic_cmd` is the internal ioctl command intended for a downstream module or driver; and `ic_timout` is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgment before timing out. `ic_len` is the number of bytes in the data argument and `ic_dp` is a pointer to the data argument. The `ic_len` field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by `ic_dp` should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The stream head will convert the information pointed to by the **strioctl** structure to an internal ioctl command message and send it downstream. On failure, **errno** is set to one of the following values:

[ENOSR] Unable to allocate buffers for the **ioctl** message due to insufficient STREAMS memory resources.

[EFAULT] *arg* points, or the buffer area specified by `ic_dp` and `ic_len` (separately for data sent and data returned) is, outside the allocated address space.

[EINVAL] `ic_len` is less than 0 or `ic_len` is larger than the maximum configured size of the data part of a message or `ic_timout` is less than -1.

[ENXIO] Hangup received on *fildes*.

[ETIME] A downstream **ioctl** timed out before acknowledgment was received.

An **I_STR** can also fail while waiting for an acknowledgment if a message indicating an error or a hangup is received at the stream head. In addition, an error code can be returned in the positive or negative acknowledgment message, in the event the ioctl command sent downstream fails. For these cases, **I_STR** will fail with **errno** set to the value in the message.

I_SENDFD Requests the stream associated with *fildes* to send a message, containing a file pointer, to the stream head at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user id and group id associated with the sending process are also inserted. This message is placed directly on the read queue (see **intro**(S)) of the stream head at the other end of the stream pipe to which it is connected. On failure, **errno** is set to one of the following values:

[EAGAIN] The sending stream is unable to allocate a message block to contain the file pointer.

[EAGAIN] The read queue of the receiving stream head is full and cannot accept the message sent by **I_SENDFD**.

[EBADF] *arg* is not a valid, open file descriptor.

[EINVAL] *fildes* is not connected to a stream pipe.

[ENXIO] Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD ioctl** over a stream pipe. ***arg*** is a pointer to a data buffer large enough to hold an **strrecvfd** data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

`fd` is an integer file descriptor. `uid` and `gid` are the user id and group id, respectively, of the sending stream.

If **O_NDELAY** is not set (see **open**(S)), **I_RECVFD** will block until a message is present at the stream head. If **O_NDELAY** is set, **I_RECVFD** will fail with **errno** set to **EAGAIN** if no message is present at the stream head.

If the message at the stream head is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the `fd` field of the **strrecvfd** structure. The structure is copied into the user data buffer pointed to by ***arg***. On failure, **errno** is set to one of the following values:

[EAGAIN] A message was not present at the stream head read queue, and the **O_NDELAY** flag is set.

[EBADMSG] The message at the stream head read queue was not a message containing a passed file descriptor.

[EFAULT] ***arg*** points outside the allocated address space.

[EMFILE] **NOFILES** file descriptors are currently open.

[ENXIO] Hangup received on ***fildes***.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two streams, where ***fildes*** is the file descriptor of the stream connected to the multiplexing driver, and ***arg*** is the file descriptor of the stream connected to another driver. The stream designated by ***arg*** gets connected below the multiplexing driver. **I_LINK** requires the multiplexing driver to send an acknowledgment message to the stream head regarding the linking operation. This call returns a multiplexer ID number (an identifier used to disconnect the multiplexer, see **I_UNLINK**) on success, and a -1 on failure. On failure, **errno** is set to one of the following values:

[ENXIO] Hangup received on ***fildes***.

[ETIME] Time out before acknowledgment message was received at stream head.

[EAGAIN] Temporarily unable to allocate storage to perform the **I_LINK**.

[ENOSR] Unable to allocate storage to perform the **I_LINK** due to insufficient STREAMS memory resources.

[EBADF] ***arg*** is not a valid, open file descriptor.

[EINVAL] ***fildes*** stream does not support multiplexing.

[EINVAL] ***arg*** is not a stream, or is already linked under a multiplexer.

[EINVAL] The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given stream head is linked into a multiplexing configuration in more than one place.

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of ***fildes***. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, **I_LINK** will fail with **errno** set to the value in the message.

I_UNLINK Disconnects the two streams specified by ***fildes*** and ***arg***. ***fildes*** is the file descriptor of the stream connected to the multiplexing driver. ***fildes*** must correspond to the stream on which the **ioctl I_LINK** command was issued to link the stream below the multiplexing driver. ***arg*** is the multiplexer ID number that was returned by the **I_LINK**. If ***arg*** is -1, then all Streams which were linked to ***fildes*** are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink. On failure, **errno** is set to one of the following values:

[ENXIO] Hangup received on ***fildes***.

[ETIME] Time out before acknowledgment message was received at stream head.

[ENOSR] Unable to allocate storage to perform the **I_UNLINK** due to insufficient STREAMS memory resources.

[EINVAL] ***arg*** is an invalid multiplexer ID number or ***fildes*** is not the stream on which the **I_LINK** that returned ***arg*** was performed.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of ***fildes***. In addition, an error code can be returned in the positive or negative acknowledgment message. For these cases, **I_UNLINK** will fail with **errno** set to the value in the message.

See also

close(S), **fcntl**(S), **getmsg**(S), **Intro**(S), **ioctl**(S), **open**(S), **poll**(S), **putmsg**(S), **read**(S), **signal**(S), **sigset**(S), **write**(S)

Diagnostics

Unless specified otherwise above, the return value from **ioctl** is 0 upon success and -1 upon failure with **errno** set as indicated.

tape

magnetic tape device

Description

The **tape** device implements the UNIX interface with a tape drive. QIC-02 cartridge tape drives are supported by the *ct* device driver, QIC-40 and QIC-80 tape drives connected to the floppy disk controller are supported by the *ft* device driver, and Irwin tape drives connected to the floppy disk controller are supported by the *mc* device driver. SCSI tape devices are supported by the *Stp* driver, and *cpqs* controls Compaq SCSI tape devices. Typically, the **tar**(C), **cpio**(C), **dd**(C), **backup**(ADM), **xbackup**(ADM), **xrestore**(ADM), or **restore**(ADM) commands are used to access a tape drive.

Each device supports a single tape drive with a raw (character, non-blocking) interface, except for the SCSI tape driver which supports up to four devices. There are four standard tape device types. Devices beginning with the "r" prefix, (for "raw device"), should be used for most normal tape work, while devices with the "n" prefix, (for "no rewind on close"), should be used for storing and restoring multiple files. Devices beginning with the "x" prefix are control devices, which are used for issuing **ioctl**(S) calls to the tape subsystem.

Devices beginning with the "e" prefix (for ECC device) support a 2/64 error recovery scheme. Thus, two 512-byte blocks out of every 64 blocks can be bad and the driver will correct the errors. This software ECC support provides a high degree of error recovery.

ECC encoding and decoding for the *ft* and *mc* floppy tape drivers is automatically used with the standard "r" device; the *ft* and *mc* floppy tape drivers do not support the "n" or "e" device types. ECC encoding and decoding is automatically used with the standard "r" device. On the QIC-40, QIC-80 and Irwin 80 MB drives, for every 29K written to the tape, 3K of ECC data is written with it to provide error recovery. On the Irwin 10, 20, 40 and 60 MB drives, for every 16K written to the tape, 2K of ECC data is written.

QIC-40 and QIC-80 tapes must be formatted before use; use pre-formatted QIC-40 and QIC-80 tapes for best results. Similarly, Irwin tapes must be first servo-written and then formatted with **tape**(C) before use, unless you use pre-formatted Irwin tapes. The *mc* driver can read tapes formatted and written under the old version of UNIX but cannot write to them.

The following table summarizes the base naming conventions for the tape drives supported:

ct0,1	QIC24 unit 0,1
ct2,3	QIC11 unit 0,1
Stp0,1,2,3	SCSI tape unit 0,1,2,3
rmt/cst0,1,2,3,4	Compaq SCSI tape unit 0,1,2,3,4
ft0\ \	QIC-40 or QIC-80 floppy tape unit
mc0\ \	Irwin floppy tape unit
ctmini	default mini-cartridge device
mt0,1	reel to reel unit 0,1 1600 bpi
mt2,3	reel to reel unit 0,1 800 bpi
mt4,5	reel to reel unit 0,1 6250 bpi

The default tape device is stored in the file */etc/default/tape*, which is also used by **tape**(C). */etc/default/tape* should always contain the "x" (control) device name of the default device, and is normally updated by **mkdev**(ADM) **tape**. If the default device is a QIC-40, QIC-80 or Irwin tape drive, the appropriate device from the table above will be linked to the ctmini device node. QIC-02 tape drives will always be accessed by the ct0,1 device nodes as shown in the table. If a SCSI tape drive is installed as the default device and there is no QIC-02 drive installed, it will be linked to the ct0 device node. If both SCSI and QIC-02 drives are installed, the SCSI device node cannot be linked to the ct0 device node. Compaq SCSI tape devices use a different convention and are discussed in a separate section.

Certain DAT drives (HP, for example) support advanced features such as partitions and setmarks and have special device nodes. See **dat**(HW) for more information.

tape(C) describes the commands used to access tape drives.

Definition of ioctl calls

The following **ioctl**(S) calls can be used with the various tape device drivers supported under UNIX. The letters following each description indicate which drivers support each **ioctl** call:

- A All drivers
- C QIC-02 cartridge tape driver
- S SCSI tape driver
- F QIC-40 and QIC-80 mini-cartridge tape drivers
- I Irwin mini-cartridge tape driver

MT_STATUS Returns a device-independent structure holding the status of the drive. The **tape_info** structure is defined in */usr/include/sys/tape.h.* (C,S,F)

MT_DSTATUS Returns a device-dependent structure holding status information of the drive. (C,S,F)

MT_RESET Resets the driver software and the tape drive. Interrupts tape operations in progress. (C,S,F)

MT_REPORT Returns an integer code that determines the type of device which the driver controls. The type numbers are defined in */usr/include/sys/tape.h.* (C,S,F)

MT_RETEN Winds the tape forward to EOT and then backward to BOT. (A)

MT_REWIND Rewinds the tape to BOT. (A)

MT_ERASE Erases the data on the tape and retensions the cartridge. (C,S,F)

MT_AMOUNT Returns an integer count of the amount of the last data transfer. (C,S,F)

MT_FORMAT Formats the tape. Expects as an argument the number of tracks to format, which must be an even number. If no argument is provided, the default is 20 tracks for QIC-40 drives, and 28 tracks for QIC-80 drives. (FI)

MT_GETHDR Expects as an argument a pointer to a **struct ft_header** or **struct ir_header** and copies the header of the current tape into it. (FI)

MT_PUTHDR Takes a pointer to a **struct ft_header** or **struct ir_header** and writes it onto the tape. This call should be used with caution. (FI)

MT_GETNEWBB Takes a pointer to a **struct ft_newbbt** or **struct ir_newbbt** and copies in a list of bad blocks detected on the last write operation. (FI)

MT_PUTNEWBB Takes a pointer to a **struct ft_newbbt** or **struct ir_newbbt,** reads in the header from the tape, then writes a new bad block onto the tape with the new bad blocks from the provided bad block table. (FI)

MT_SETBLK Sets the block size (in bytes) of the tape blocks. Takes as an argument the size in bytes. (S)

MT_GETVTBL Takes a pointer to a **struct ft_vtbl** and copies in the volume table from the tape. (FI)

MT_PUTVTBL	Takes a pointer to a **struct ft_vtbl** and writes the volume table onto the tape. This call should be used with caution. (FI)
MT_RFM	Winds the tape forward to the next filemark. (A)
MT_WFM	Writes a filemark at the current location on the tape. (A)
MT_LOAD	On devices which are capable of doing so, loads the tape into the drive. (S)
MT_UNLOAD	On devices which are capable of doing so, unloads the tape from the drive. (S)

Floppy tape drive configuration and media

This section is concerned with QIC-40 and QIC-80 tape drives and supported floppy tape cartridges.

Configuration options for the floppy tape driver (ft)

Five configuration options are available for the floppy tape driver (*ft*) for QIC-40 and QIC-80 tape drives. To alter these variables, it is necessary to edit the configuration file */etc/conf/pack.d/ft/space.c*, relink the kernel, and reboot the system with the new kernel. While all options may be edited manually in *space.c*, the first and second options, **ft_drivetype** and **ft_select_mode**, can also be modified using the **mkdev tape** command. These options are summarized in the following table:

Floppy Tape Drive Configuration Options

Variable	Values	Editing	
		mkdev	Manual
ft_drive_type	**FT_QIC40** **FT_QIC40XL** (FT_QIC60) **FT_QIC80** **FT_QIC80XL**	yes	yes
ft_select_mode	**FT_HARD_SELECT** **FT_SOFT_SELECT**	yes	yes
ft_alloc_switch	**FT_OPEN_TIME** **FT_INIT_TIME**	no	yes
ft_minbufs	2 <= x <= (ft_maxbufs)	no	yes
ft_maxbufs	(ft_minbufs) <= x <= 20	no	yes

ft_drivetype
When set from **FT_QIC40** to **FT_QIC40XL**, or from **FT_QIC80** to **FT_QIC80XL**, **ft_drivetype** enables support of extended length (307.5 ft) DC2120 mini-cartridges in the driver. **FT_QIC60** is an obsolete value; **FT_QIC40XL** should be used instead.

A QIC-40 drive should never be operated with **ft_drivetype** set to **FT_QIC80** or **FT_QIC80XL**. Also, a QIC-80 drive should never be operated with **ft_drivetype** set to **FT_QIC40** or **FT_QIC40XL**. If they are, your system may panic or hang, or data may be lost from the tape media

ft_select_mode
When set to **FT_HARD_SELECT**, the *ft* driver expects a normal tape drive with a drive select jumper to be present. If **ft_select_mode** is set to **FT_SOFT_SELECT**, the *ft* driver uses Soft Select mode to access the tape drive, which must be a Soft Select-compatible QIC-40 or QIC-80 tape drive with no drive select jumpers installed.

ft_alloc_switch
When set to **FT_OPEN_TIME**, the *ft* driver allocates memory when it is needed, and frees memory once it is no longer needed. Typically, memory is allocated at open time and released at close time. When **ft_alloc_switch** is set to **FT_INIT_TIME**, all memory allocation is done at **init** time (when the system boots) and the *ft* driver retains this memory. The advantage of **init** time allocation is that more memory is available for the *ft* buffers. The disadvantage is that this memory is held by the tape driver and cannot be used elsewhere in the system.

ft_minbufs
ft_maxbufs
These variables set the minimum and maximum number of 32K buffers that the *ft* driver is allowed to use. The lower limit of the minimum is hard set to 2, with any value lower than 2 interpreted as 2. The upper limit of the maximum is hard set to 20, with numbers greater than 20 interpreted as 20. When the *ft* driver allocates memory for buffers, it requests **ft_maxbufs** first. If this fails, *ft* reduces its request by one and tries again. This continues until either the request is satisfied or **ft_minbufs** buffers fail to be allocated. If the request is satisfied, then *ft* proceeds. Otherwise, the tape operation fails and **ENOMEM** (Error, no memory for buffers) is returned.

The exception to this is when **ft_alloc_switch** is set to **FT_INIT_TIME** (allocation takes place at **init** time). If the **ft_minbufs** are not available at **init** time, then *ft* resorts to allocation as needed (as if **ft_alloc_switch** were set to **FT_OPEN_TIME**). If this condition occurs, a warning is printed at **init** time.

It is recommended that **ft_minbufs** be set to at least 3. Although the driver can operate with only two buffers, performance is seriously degraded; the tape never streams, and in fact produces an underrun for every 32K block written to the tape.

Floppy tape cartridge media

There are three mini-cartridge types appropriate to floppy tape drives:

Floppy Tape Mini-Cartridge Types

Cartridge	Length (ft)	Width (in)	Maximum Density (ftpi)
DC2000	205	0.25	12,500
DC2080	205	0.25	14,700
DC2120	307.5	0.25	14,700

DC2000 and DC2080 are available unformatted or preformatted from the manufacturer. Irwin format tapes are incompatible with the floppy tape driver.

If a 2120 cartridge is inserted into a drive, it can be detected as an extended length tape and will be formatted for 60 Mbytes on QIC-40 drives, and 120 Mbytes on QIC-80 drives. Some older QIC-40 drives are not able to detect extended length tapes. These models should never be operated with a DC2120 inserted, since they will expect a 205 foot tape length. Drives cannot distinguish between blank (unformatted) DC2000 & DC2080. Never attempt to format a DC2080 in a QIC-40 drive, or a DC2000 in a QIC-80 drive.

Media configurations for floppy tape drives are summarized in the following table:

Floppy Tape Drive Media Configurations

Drive	ft_drivetype	Media	Formatted Capacity (MB)	Operations
QIC-40	**FT_QIC40**	DC2000	40	r/w/f
QIC-40	**FT_QIC40XL**	DC2000	40	r/w/f
		DC2120	60	r/w/f
QIC-80	**FT_QIC80**	DC2000	40	r
		DC2080	80	r/w/f
QIC-80	**FT_QIC80XL**	DC2000	40	r
		DC2120	60	r
		DC2080	80	r/w/f
		DC2120	120	r/w/f

Irwin-specific ioctl interface

Device-specific functions of the Irwin tape drive are accessed via special commands passed to the Irwin driver using the **ioctl**() interface. An Irwin driver interface library is available. This library provides a system-independent interface to **ioctl**() via the entry point **mcioctl**():

```
#include "mc.h"

int mcioctl(fh, cmd, arg)
int fh;         /* File handle from open() */
int cmd;        /* MCCTL_* command code */
void *arg;      /* Additional argument pointer */

mcioctl(fh, MCCTL_NOP, NULL)
mcioctl(fh, MCCTL_VERSION, verbuf)
mcioctl(fh, MCCTL_CAPACITY, capp)
mcioctl(fh, MCCTL_LSEEK, lskbuf);
mcioctl(fh, MCCTL_REWIND)
mcioctl(fh, MCCTL_RETEN)
mcioctl(fh, MCCTL_REWIND_NW)
mcioctl(fh, MCCTL_RETEN_NW)
mcioctl(fh, MCCTL_GETDRVCFG, cfgbuf)
mcioctl(fh, MCCTL_GETCFG, cfgbuf)
mcioctl(fh, MCCTL_SETCFG, cfgbuf)
mcioctl(fh, MCCTL_GETTHDR, hdrbuf)
mcioctl(fh, MCCTL_PUTTHDR, hdrbuf)
mcioctl(fh, MCCTL_GETDLISTS, listbuf)
mcioctl(fh, MCCTL_FLUSH)
mcioctl(fh, MCCTL_FORMAT, fmtbuf)
mcioctl(fh, MCCTL_FMTSTAT, fmtbuf)
mcioctl(fh, MCCTL_ABORT)
mcioctl(fh, MCCTL_DEVSTAT, dstatp)
mcioctl(fh, MCCTL_GETERCTL, erctlp)
mcioctl(fh, MCCTL_SETERCTL, erctlp)
mcioctl(fh, MCCTL_GETER, ierrp)
struct mcver *verbuf;    /* version buffer */
long *capp;              /* capacity in bytes */
struct mclseek *lskbuf;  /* tape logical position descriptor */
struct mccfg *cfgbuf;    /* configuration buffer */
char *hdrbuf;            /* 1024 byte header buffer */
unsigned short * listbuf; /* 2048 byte defect list buffer */
struct mcfmt *fmtbuf;    /* format control/status buffer */
unsigned short *dstatp;  /* device status word */
unsigned short *erctlp;  /* error control word */
unsigned short *ierrp;   /* device specific error */
```

mcioctl() provides a system-independent **ioctl()** interface to the Irwin driver. This subroutine is essentially a pass-through; that is, arguments are passed through to **ioctl()**. If a device-specific error occurs (that is, a non-system error) at completion of the system **ioctl()** and the function is not a **MCCTL_NOP** or **MCCTL_VERSION**, **mcioctl()** executes **ioctl(MCCTL_GETER)** to retrieve the device-specific error.

The following **ioctl** calls are available for the Irwin driver:

MCCTL_NOP No operation. The argument is ignored. A success status is returned. This may be used as an aid in determining if a special file refers to the *mc* driver.

MCCTL_VERSION
Gets driver version information. The argument is the address of version information buffer (see **struct mcver** in */usr/include/sys/mc.h*) to which the driver writes.

MCCTL_CAPACITY
Gets a tape's capacity in bytes. The argument is the address of a long integer.

MCCTL_REWIND,
MCCTL_RETEN,
MCCTL_REWIND_NW,
MCCTL_RETEN_NW
These four calls physically position the tape at high speed. **MCCTL_RETEN** and **MCCTL_RETEN_NW** run the tape to the early warning hole first. All four calls return the tape to the load-point hole. **MCCTL_REWIND_NW** and **MCCTL_RETEN_NW** start a request but do not wait for completion.

MCCTL_GETDRVCFG,
MCCTL_GETCFG,
MCCTL_SETCFG
These three function calls provide access to configuration parameters for a particular mini cartridge tape unit. The structure of these parameters is **struct mccfg** (defined in */usr/include/sys/mc.h*). This structure has driver, tape drive, and cartridge related fields. Both **MCCTL_GETDRVCFG** and **MCCTL_GETCFG** copy the driver's **mccfg** structure to the caller's buffer. When **MCCTL_GETDRVCFG** is used, members of **struct mccfg** with driver and tape drive related fields are returned. No error is given when a cartridge is absent. When **MCCTL_GETCFG** is used successfully, all fields are returned with valid data. An error is returned if no cartridge is present. **MCCTL_SETCFG** allows the caller to adjust certain fields in the driver's configuration.

MCCTL_GETTHDR,
MCCTL_PUTTHDR

MCCTL_GETTHDR and **MCCTL_PUTTHDR** read and write the 1024-byte tape header in block 0. **MCCTL_PUTTHDR** assumes an Irwin-style header. Then the following procedure is used to write the header:

Tape block 0 is read to a buffer. The caller's 1024-byte header buffer is copied to the first, fifth, and when space permits, the ninth and thirteenth 1024-byte sectors in the buffer. When the cartridge format uses ECC (that is, other than 110 cartridge format), the header's ECC in use field is set. When the cartridge format uses ECC, ECC is encoded. A checksum is calculated for the buffer. The buffer is written back to block 0. Block 0 is reread and the cartridge state is redetermined. A new checksum is calculated and compared against the original.

MCCTL_GETDLISTS

Returns lists used by the driver's flaw management. The caller gives the address of a buffer which is at least 2K in length. Four lists are copied to the buffer. Each list comprises of physical tape block numbers stored as unsigned short integers and terminated with the value 0xffff. The lists are contiguous and given in the following order:

Primary Defect List (PDL)
Working Defect List (WDL)
Grown Defect List (GDL)
Relocation List (RL)

MCCTL_FLUSH Flushes dirty buffers to tape. **MCCTL_FLUSH** forces dirty buffers in the Irwin driver's cache to be written to tape. The pointer argument is ignored. Control returns when data is written. Buffers are automatically flushed upon a **close**() or when the device is idle for a certain period (see `mc_autoflush` in **`struct mccfg`** in */usr/include/sys/mc.h*).

MCCTL_FORMAT,
MCCTL_FMTSTAT

MCCTL_FORMAT starts an erase, servo-format-certify-initialize header or re-certify operation. The argument is the address of **`struct mcfmt`** (see */usr/include/sys/mc.h*). Formatting operations performed depend upon the values in the structure's `fm_cmd` and `fm_option` fields, and **`struct mccfg`** `mc_cartstate` field. When an **MCCTL_FORMAT** call completes successfully, **MCCTL_FMTSTAT** is used to determine the progress (when a no-wait flag is set) or results of formatting. Like **MCCTL_FORMAT**, **MCCTL_FMTSTAT** uses the **`struct mcfmt`** structure (typically the same one passed to **MCCTL_FORMAT**).

MCCTL_ABORT Used to interrupt and terminate operations started by **MCCTL_FORMAT**. The pointer argument is ignored. Control returns after formatting has terminated.

MCCTL_DEVSTAT
Returns a 16-bit device status word to an unsigned short integer whose address is passed in the third argument of **ioctl**(). This field is intended for use by applications that use the tape drive interactively. The status bits are defined in **struct mclseek** in */usr/include/sys/mc.h*.

MCCTL_GETERCTL,
MCCTL_SETERCTL
MCCTL_GETERCTL and **MCCTL_SETERCTL** give application access to the state of, and control over, certain error mechanisms. The argument is the address of a 16-bit error control variable which the Irwin driver writes with current values for **MCCTL_GETERCTL** and reads for **MCCTL_SETERCTL**. Certain flags may or may not have an effect depending on the implementation. Bit values for the error control variable are defined in */usr/include/sys/mc.h*.

MCCTL_GETER Gets device-specific error: **IE_***. In general, the value 0 is returned to indicate success or -1 to indicate an error. When **mcioctl**() returns the value -1, an error has occurred. The error condition may have been detected in the operating system or in the driver. In order to tell where the error comes from, the global **_mcerrno** should be examined first (before **errno**). If **_mcerrno** is non-zero, the error was returned by the driver. Values for **_mcerrno** are defined in */usr/include/ierrno.h* with an **IE_** prefix.

Irwin drive and cartridge models

This section is concerned with Irwin tape drives and cartridges supported.

Drive models

Many Irwin mini cartridge drives have a three-digit model number. Each digit has a meaning. The high-order digit encodes the form factor and cabinetry:

1xx 5-¼ inch drive (mounted in system cabinet).

2xx 3-½ inch drive (mounted in system cabinet).

3xx 5-¼ inch drive in a metal cabinet with power supply.

4xx 3-½ inch drive in a plastic cabinet (no supply).

7xx 3-½ inch drive in a metal cabinet with power supply.

The middle digit gives the approximate capacity, in 10 Megabyte units, for a standard capacity (not extra-long) tape:

1	10 Megabytes
2	20
4	40
6	60
8	80

The low digit encodes the drive's normal data transfer rate (that is, the floppy controller data clock rate).

xx0	250 Kilobits/Second
xx5	500 Kilobits/Second
xx7	1 Megabit/Second

In addition, a new 4-digit model numbering system is in use. These model numbers are associated with drives which are adaptable to different system hardware environments with accessory hardware kits.

2020	3-½ inch, 20 Megabyte, 250 Kilobits/Second
2040	3-½ inch, 40 Megabyte, 500 Kilobits/Second
2080	3-½ inch, 80/120 Megabyte, 500 Kilobits/Second
2120	3-½ inch, 80/120 Megabyte, 1 Megabit/Second

Mini cartridges

There are three primary physical mini cartridges types:

DC1000	185 feet of 0.150 inch wide tape (same as TC-200)
DC2000	205 feet of 0.250 inch wide tape (same as TC-400)
DC2120	307.5 feet of 0.250 inch wide tape

The DC1000 cartridge is physically thinner than DC2000 and DC2120 cartridges. The DC2000 and DC2120 have the same physical form but the DC2120 has a longer tape. These cartridges are distinguished by their labels. Each physical cartridge type has at least two cartridge formats:

Mini (Irwin) Cartridge Format Parameters

Cartridge Format	AccuTrak Reorder Number (see Note)	Cartridge	Total Tape Blocks	Trks	Sectors Blocks per Track	per Block Data	ECC	Density (FTPI)
110	1000-10	DC1000	1264	8	158	8	0	6400
120	2000-20	DC2000	1190	14	85	16	2	6400
120XL	2000-30	DC2120	1792	14	128	16	2	6400
125	1000-20	DC1000	1320	12	110	16	2	10000
145	2000-40	DC2000	2480	20	124	16	2	10000
145XL	2000-60	DC2120	3720	20	186	16	2	10000
165	2000-64	DC2000	3936	24	164	16	2	13200
285	2000-80	DC2000	2752	32	86	29	3	11600
285XL	2000-120	DC2120	4160	32	130	29	3	11600

Notes: The suffix part of the AccuTrak Reorder Number is an approximate cartridge capacity in Megabytes. All formats use 1024-byte MFM encoded sectors.

Drive Read/Write Compatibility for Mini Cartridge Formats

	Drive Model (See Note) 2020			2040				
Cartridge Format	410 310 110	720 420 320 220 120	725 425 325 225 125	745 445 345 245 145	765 465 265 165	2080 785 485 285	2120 787 487 287	Cartridge
110	rw	rw	r-	r-	r-	r-	r-	DC1000
120	—	rw	—	r-	r-	r-	r-	DC2000
120XL	—	rw	—	r-	r-	r-	r-	DC2120
125	—	—	rw	rw	r-	r-	r-	DC1000
145	—	—	—	rw	r-	r-	r-	DC2000
145XL	—	—	—	rw	r-	r-	r-	DC2120
165	—	—	—	—	rw	r-	r-	DC2000
285	—	—	—	—	—	rw	rw	DC2000
285XL	—	—	—	—	—	rw	rw	DC2120

Key:

r Drive reads cartridge format

w Drive writes cartridge format

- Incompatible: When a cartridge is formatted but incompatible for reading or writing, the driver reports that the cartridge is either incompatible or erased.

Extra long (XL) DC2120 cartridge compatibility

Extra long (that is, DC2120) cartridges are incompatible with the following drives; the drive will not physically accommodate the cartridge:

110, 310, 410, 125, 225, 325, 425, and 725

Even though DC2120 cartridges are physically accepted in the following drives, you may not be able to format them:

120, 220, 320, 420, 720, 2020, 145, 245, 345, 445, 745, 2040

Drives manufactured prior to 1989 don't recognize the longer tape. However, the *mc* driver is able to read and write preformatted extra long tapes in these drives, but it is unable to correctly format them. Formatting will start, but terminate in error. To determine whether a drive supports formatting of DC2120 cartridges, use the **mcart** utility. See **tape**(C) for information about **mcart**. If the command **mcart drive** reports a drive type with the suffix XL, formatting of DC2120 cartridges is supported.

Compaq SCSI Tape Devices

Compaq tape drives use a different set of device name conventions. The device node format for accessing DAT and/or 320/525 tape drive on a SCSI tape adapter or a SCSI tape compression adapter follows:

/dev/rmt/cst*n*[cin][-150]

where *n* is the SCSI ID of the tape drive (0-4).

The following table describes the available options:

Option	Description
c	Access the tape drive using the compression chip on the tape adapter. To use this option, no other drives on the adapter can be in use by other processes. Once a drive is being accessed using the compression chip, the other drives on the adapter cannot be used until the process using the compression mode closes the tape drive.
i	Immediate mode. Certain commands, like erase or retension will return to the application program before the command actually finishes on the tape drive.
n	No rewind. A rewind command will not be issued to the drive when the device is closed.
-150	Access the tape in QIC-150 density. This option is only relevant on a 320/525 tape drive.

The devices listed access a DAT or 320/525 tape drive. The auto-density mode selects the format to match the data on the media for reading. The highest density for the media type present is selected for writing. On the 320/525 tape drive, the QIC-150 mode is used to force QIC-150 format when writing on high density media so that the tape can be read on 150/250 type drives. If a 6150 or 6250 tape media is used, the drive writes QIC-150 format in the auto-density mode.

On both a DAT and 320/525 tape drive, immediate mode returns without waiting for the command to complete, which is useful for executing retension and erase tape commands.

Compaq tape formats

The Compaq SCSI adapter and 320/525 SCSI tape drive support the tape formats listed in the following table:

Recording Format

Media Type	QIC-24	QIC-120	QIC-150	QIC-320
DC-300	RD	N/A	N/A	N/A
DC-600A	RD	RD	N/A	N/A
DC-6150	RD	RD	RD/WR(150 M)	N/A
DC-6320	RD	RD	RD/WR(150 M)	RD/WR(320 M)
DC-6525	RD	RD	RD/WR(250 M)	RD/WR(525 M)

The default device automatically determines the tape format to use. For reading, the data written on the tape media determines the tape density and format. The media type determines the default on write operations; that is, the highest density for the type of media present is written. If you want to use a lower tape density, use a device type that explicitly selects lower density.

SCSI minor device numbers

The minor device numbering scheme for SCSI tape devices is as follows (other than Compaq SCSI) :

SCSI Tape Minor Devices

Bits								
7	6	5	4	3	2	1	0	Description
-	-	-	-	-	-	X	X	Unit (LUN)
-	-	-	-	-	X	-	-	No unload on close
-	-	-	-	X	-	-	-	No rewind on close
-	-	-	X	-	-	-	-	High density (6250 BPI)
-	-	X	-	-	-	-	-	ECC
-	X	-	-	-	-	-	-	Partition (HP DAT)
X	-	-	-	-	-	-	-	Control/overrride device

Files

/dev/rStp0
/dev/nrStp0
/dev/xStp0
/dev/rft0
/dev/xft0
/dev/rmc1

/dev/rct0
/dev/nrct0
/dev/rct2
/dev/nrct2
/dev/xct0
/dev/mcdaemon

/dev/erct0
/dev/xct0
/dev/rctmini
/dev/xctmini
/dev/rmc0

/dev/rmt/cst0
/dev/rmt/cst0n
/dev/rmt/cst0c
/dev/rmt/cst0cn
/dev/rmt/cst0i
/dev/rmt/cst0in
/dev/rmt/cst0ci
/dev/rmt/cst0cin

/dev/rmt/cst0-150
/dev/rmt/cst0n-150
/dev/rmt/cst0c-150
/dev/rmt/cst0cn-150
/dev/rmt/cst0i-150
/dev/rmt/cst0in-150
/dev/rmt/cst0ci-150
/dev/rmt/cst0cin-150

Include files:
/usr/include/sys/Stp.h
/usr/include/sys/ft.h
/usr/include/sys/mc.h
/usr/include/sys/ct.h
/usr/include/sys/tape.h
/usr/include/sys/ir.h
/usr/include/sys/mcheader.h

Notes

After certain tape operations are executed, the system returns a prompt before the tape controller has finished its operation. If the user enters another tape command too quickly, a "device busy" error is returned until the tape device is finished with its previous operation.

Periodic tape cartridge retensioning and tape head cleaning are necessary for continued error-free operation of the tape subsystem. Use **tape**(C) to retension the tape.

See also

backup(ADM), **cpio**(C), **dat**(HW), **dd**(C), **format**(C), **mkdev**(ADM), **restore**(ADM), **tape**(C), **tar**(C), **xbackup**(ADM), **xrestore**(ADM)

terminal

login terminal

Description

A **terminal** is any device used to enter and display data. It may be connected to the computer:

- By a serial wire, either direct or dialup
- As a virtual terminal, for example with emulator software
- Through a display adapter

A terminal has an associated device file */dev/tty**.

File

*/dev/tty**

See also

console(M), **disable**(C), **enable**(C), **mkdev**(ADM), **serial**(HW), **stty**(C), **term**(F), **termcap**(M), **terminals**(M), **vidi**(C)

timod

Transport Interface cooperating STREAMS module

Description

timod is a STREAMS module for use with the Transport Layer Interface (TLI) functions of the Network Services library. The **timod** module converts a set of **ioctl**(S) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Layer Interface. This allows a user to initiate certain TLI functions as atomic operations.

The **timod** module must only be pushed onto a stream terminated by a transport protocol provider which supports the TLI.

All STREAMS messages, with the exception of the message types generated from the **ioctl** commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following **ioctl** commands are recognized and processed by the **timod** module. The format of the **ioctl** call is:

```
#include <sys/stropts.h>
                .
                .
struct strioctl strioctl;
                .
                .
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf
ioctl(fildes, I_STR, &strioctl);
```

where, on issuance, `size` is the size of the appropriate TLI message to be sent to the transport provider and on return, `size` is the size of the appropriate TLI message from the transport provider in response to the issued TLI message. `buf` is a pointer to a buffer large enough to hold the contents of the appropriate TLI messages. The TLI message types are defined in *<sys/tihdr.h>*. The possible values for the `cmd` field are:

TI_BIND Bind an address to the underlying transport protocol provider. The message issued to the **TI_BIND ioctl** is equivalent to the TLI message type **T_BIND_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TLI message type **T_BIND_ACK**.

TI_UNBIND Unbind an address from the underlying transport protocol provider. The message issued to the **TI_UNBIND ioctl** is equivalent to the TLI message type **T_UNBIND_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TLI message type **T_OK_ACK**.

TI_GETINFO Get the TLI protocol specific information from the transport protocol provider. The message issued to the **TI_GETINFO** **ioctl** is equivalent to the TLI message type **T_INFO_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TLI message type **T_INFO_ACK**.

TI_OPTMGMT Get, set, or negotiate, protocol specific options with the transport protocol provider. The message issued to the **TI_OPTMGMT** **ioctl** is equivalent to the TLI message type **T_OPTMGMT_REQ**, and the message returned by the successful completion of the **ioctl** is equivalent to the TLI message type **T_OPTMGMT_ACK**.

Files

<sys/timod.h>
<sys/tiuser.h>
<sys/tihdr.h>
<sys/errno.h>

See also

tirdwr(HW)

Diagnostics

If the **ioctl** system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TLI error codes as defined in *<sys/tiuser.h>*. If the TLI error is of type **TSYSERR**, then the next 8 bits of the return value will contain an error as defined in *<sys/errno.h>* (see **intro**(S)).

tirdwr

Transport Interface read/write interface STREAMS module

Description

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Layer Interface (TLI) functions of the Network Services library (see Section NSL). This alternate interface allows a user to communicate with the transport protocol provider using the **read**(S) and **write**(S) system calls. The **putmsg**(S) and **getmsg**(S) system calls may also be used. However, **putmsg** and **getmsg** can only transfer data messages between user and stream.

The **tirdwr** module must only be pushed (see **I_PUSH** in **streamio**(STR)) onto a stream terminated by a transport protocol provider which supports the TLI. After the **tirdwr** module has been pushed onto a stream, none of the Transport Layer Interface functions can be used. Subsequent calls to TLI functions will cause an error on the stream. Once the error is detected, subsequent system calls on the stream will return an error with **errno** set to **EPROTO**.

The following are the actions taken by the **tirdwr** module when pushed on the stream, popped (see **I_POP** in **streamio**(STR)) off the stream, or when data passes through it.

push When the module is pushed onto a stream, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the stream that relate to process management, such as messages that generate signals to the user processes associated with the stream. If any other messages are present, the **I_PUSH** will return an error with **errno** set to **EPROTO**.

write The module will take the following actions on data that originated from a **write** system call:

All messages with the exception of messages that contain control portions (see the **putmsg** and **getmsg** system calls) will be transparently passed onto the module's downstream neighbor.

Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.

Any messages with control portions will generate an error, and any further system calls associated with the stream will fail with **errno** set to **EPROTO**.

read The module will take the following actions on data that originated from the transport protocol provider:

All messages with the exception of those that contain control portions (see the **putmsg** and **getmsg** system calls) will be transparently passed onto the module's upstream neighbor.

The action taken on messages with control portions will be as follows:

Messages that represent expedited data will generate an error. All further system calls associated with the stream will fail with **errno** set to **EPROTO**.

Any data messages with control portions will have the control portions removed from the message prior to passing the message to the upstream neighbor.

Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the stream. The orderly release message itself will be freed by the module.

Messages that represent an abortive disconnect indication from the transport provider will cause all further **write** and **putmsg** system calls to fail with **errno** set to **ENXIO**. All further **read** and **getmsg** system calls will return zero length data (indicating end of file) once all previous data has been read.

With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the stream will fail with **errno** set to **EPROTO**.

Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop When the module is popped off the stream or the stream is closed, the module will take the following action:

If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

See also

getmsg(S), **intro**(NSL), **intro**(S), **putmsg**(S), **read**(S), **streamio**(HW), **timod**(HW), **write**(S)

xt

multiplexed tty driver for AT&T windowing terminals

Description

The **xt** driver provides virtual **tty**(M) circuits multiplexed onto real tty lines. It interposes its own channel multiplexing protocol as a line discipline between the real device driver and the standard tty line disciplines.

The **xt** driver can be configured using the **mkdev layers** script. See **mkdev**(ADM) for more information.

Virtual ttys are named */dev/xt??[0-7]* and are allocated in groups of up to eight. Filenames end in three digits, where the first two digits represent the group and the last digit represents the virtual tty number of the group. The */dev/xt* form of the name increases the size of */dev*, which adversely affects some commands; the */dev/xt/* form is not understood by most commands.

Allocation of a new channel group is done dynamically by attempting to open a name ending in "0" with the **O_EXCL** flag set. After a successful **open**, the tty file onto which the channels are to be multiplexed should be passed to **xt** via the **XTIOCLINK ioctl**(S) request. Afterwards, all the channels in the group will behave as normal tty files, with data passed in packets via the real tty line.

The **xt** driver implements the protocol described in **xtproto**(M) and in **layers**(M). Packets are formatted as described in **xtproto**(M), while the contents of packets conform to the description in **layers**(M).

There are three groups of **ioctl** requests recognized by **xt**. The first group contains all the normal tty **ioctl** requests described in **termio**(M), plus 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 of **ioctl** requests concerns control of the windowing terminal, and is described in the header file *<sys/jioctl.h>*. The requests are as follows:

JTYPE, JMPX Both return the value **JMPX**. These are used to identify a terminal device as an **xt** channel.

JBOOT, JTERM Both generate an appropriate command packet to the windowing terminal affecting the layer associated with the file descriptor argument to **ioctl**. They may return the error code **EIO** if the system **clist** is empty.

JTIMO, JTIMOM **JTIMO** specifies the timeouts in seconds, and **JTIMOM** in milliseconds. Invalid except on channel 0. They may return the error code **EIO** if the system **clist** is empty.

JWINSIZE Requires the address of a **jwinsize** structure as an argument. The window sizes of the layer associated with the file descriptor argument to **ioctl** are copied to the structure.

JZOMBOOT Generate a command packet to the windowing terminal to enter download mode on the channel associated with the file descriptor argument to **ioctl**, like **JBOOT**; but when the download is finished, make the layer a zombie (ready for debugging). It may return the error code **EIO** if the system **clist** is empty.

JAGENT Sends the supplied data as a command packet to invoke a windowing terminal agent routine, and return the terminal's response to the calling process. Invalid except on the file descriptor for channel 0. See **jagent**(M). It may return the error code **EIO** if the system **clist** is empty.

The third group of **ioctl** requests concerns the configuration of **xt**, and is described in the header file *<sys/xt.h>*. The requests are as follows:

XTIOCTYPE Returns the value **XTIOCTYPE**.

XTIOCLINK Requires an argument that is a structure, **xtioclm**, containing a file descriptor (`fd`) for the file to be multiplexed and the maximum number of channels allowed. Invalid except on channel 0. This request may return one of the following errors:

EINVAL `nchans` has an illegal value.

ENOTTY `fd` does not describe a real tty device.

ENXIO **linesw** is not configured with **xt**.

EBUSY An **XTIOCLINK** request has already been issued for the channel group.

ENOMEM There is no system memory available for allocating to the tty structures.

EIO The **JTIMOM** packet described above could not be delivered.

HXTIOCLINK Like **XTIOCLINK**, but specifies that encoding mode be used.

XTIOCTRACE Requires the address of a **tbuf** structure as an argument. The structure is filled with the contents of the driver trace buffer. Tracing is enabled. This request is invalid if tracing is not configured.

XTIOCNOTRACE Tracing is disabled. This request is invalid if tracing is not configured.

XTIOCSTATS Requires an argument that is the address of an array of size **S_NSTATS**, of type **Stats_t**. The array is filled with the contents of the driver statistics array. This request is invalid if statistics are not configured.

XTIOCDATA Requires the address of a maximum-sized **Link** structure as an argument. The structure is filled with the contents of the driver **Link** data. This request is invalid if data extraction is not configured.

Files

/dev/xt/??[0-7]	multiplexed special files
/usr/include/sys/jioctl.h	packet command types
/usr/include/sys/xtproto.h	channel multiplexing protocol definitions
/usr/include/sys/xt.h	driver specific definitions

See also

ioctl(S), **jagent**(M), **layers**(C), **layers**(M), **libwindows**(S), **mkdev**(ADM), **open**(S), **termio**(M), **tty**(M)

SCO
OPEN SYSTEMS SOFTWARE

Please help us to write computer manuals that meet your needs by completing this form. Please post the completed form to the Technical Publications Research Coordinator nearest you: The Santa Cruz Operation, Ltd., Croxley Centre, Hatters Lane, Watford WD1 8YN, United Kingdom; The Santa Cruz Operation, Inc., 400 Encinal Street, P.O. Box 1900, Santa Cruz, California 95061, USA or SCO Canada, Inc., 130 Bloor Street West, 10th Floor, Toronto, Ontario, Canada M5S 1N5.

Volume title: _____
(Copy this from the title page of the manual, for example, SCO UNIX Operating System User's Guide)

Product: _____
(for example, SCO UNIX System V Release 3.2 Operating System Version 4.0)

How long have you used this product?

❑ Less than one month ❑ Less than six months ❑ Less than one year

❑ 1 to 2 years ❑ More than 2 years

How much have you read of this manual?

❑ Entire manual ❑ Specific chapters ❑ Used only for reference

	Agree				*Disagree*
The software was fully and accurately described	❑	❑	❑	❑	❑
The manual was well organized	❑	❑	❑	❑	❑
The writing was at an appropriate technical level (neither too complicated nor too simple)	❑	❑	❑	❑	❑
It was easy to find the information I was looking for	❑	❑	❑	❑	❑
Examples were clear and easy to follow	❑	❑	❑	❑	❑
Illustrations added to my understanding of the software	❑	❑	❑	❑	❑
I liked the page design of the manual	❑	❑	❑	❑	❑

If you have specific comments or if you have found specific inaccuracies, please report these on the back of this form or on a separate sheet of paper. In the case of inaccuracies, please list the relevant page number.

May we contact you further about how to improve SCO UNIX documentation? If so, please supply the following details:

Name _____ *Position* _____

Company _____

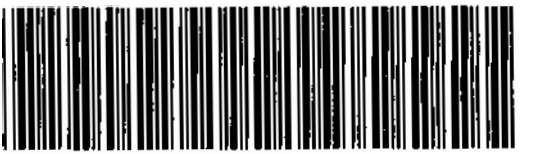
Address _____

City & Post/Zip Code _____

Country _____

Telephone _____ *Facsimile* _____

31 January 1992



BH01209P000

56269

SCO UNIX Operating System

System Administrator's Reference

For Reference

Not to be taken from this room

