UNIPLUS+ SYSTEM V

Administrator's Manual

UniSoft SYSTEMS

INTRODUCTION

This manual is intended to supplement the information contained in the *UniPlus+ User's Manual* and to provide an easy reference volume for those who must administer the UniPlus+ system. Accordingly, only those commands and descriptions deemed appropriate for system administrators have been included here.

This manual is divided into three sections:

1. System Maintenance Commands and Application Programs
7. Special Files
8. System Maintenance Procedures

Throughout this volume, each reference of the form *name*(1M), *name*(7), or *name*(8), refers to entries in this manual, while all other references to entries of the form *name*(N), where *N* is a number possibly followed by a letter, refer to entry *name* in Section *N* of the *UniPlus+ User's Manual.*

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

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

Section 8 (*System Maintenance Procedures*) discusses crash recovery and boot procedures.

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

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

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

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

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

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.

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

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

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

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE** part gives example(s) of usage, where appropriate.

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

The **SEE ALSO** part gives pointers to related information.

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

The **WARNINGS** part points out potential pitfalls.

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

On most systems, all entries are available on-line via the *man*(1) command, q.v.

Permuted Index

At the front of each volume there is a table of contents and a permuted index. The permuted index is a computer-generated index that uses the information in the **NAME** part of each entry in the User's and Administrator's Manuals. The permuted index contains three columns. The center column is an alphabetic list of keywords as they appear in the **NAME** part of the entries. The last column is the entry that the keyword in the center column refers to. This entry is followed by the appropriate section number in parentheses. The first column contains the remaining information from the **NAME** part that either precedes or follows the keyword.

For example, to look for a text editor, scan the center column for the word "editor". There are several index lines containing an "editor" reference, i.e.:

ed, red: text editor. . . . . ed(1)
files. ld: link editor for common object . . . . . ld(1)

You can then turn to the entries listed in the last column, *ed*(1) and *ld*(1), to find information on the editor.

TABLE OF CONTENTS

1. System Maintenance Commands and Application Programs

7. Special Files

8. System Maintenance Procedures

PERMUTED INDEX

NAME

intro − introduction to system maintenance commands and application programs

DESCRIPTION

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

COMMAND SYNTAX

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

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

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

SEE ALSO

getopt(1), getopt(3C).
UniPlus+ User's Manual.
UniPlus+ Administrator's Guide.

DIAGNOSTICS

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

BUGS

Regretfully, many commands do not adhere to the aforementioned syntax.

NAME

accept, reject − allow/prevent LP requests

SYNOPSIS

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

DESCRIPTION

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

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

SYNOPSIS

/usr/lib/acct/acctdisk

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

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

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

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

Process accounting is performed by the 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*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

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

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

Acctdusg reads its standard input (usually from **find / −print**) and computes disk resource consumption (including indirect blocks) by login. If **−u** is given, records consisting of those 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**.

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

Acctwtmp writes a *utmp*(4) record to its standard output. The record contains the current time and a string of characters that describe the *reason*. A record type of ACCOUNTING is assigned (see *utmp*(4)). *Reason* must be a string of 11 or less characters, numbers, **$**, or spaces. For example, the following 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 sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4).
"UNIX System Accounting" in the *UniPlus+ Administrator's Guide.*

NAME

acctcms − command summary from per-process accounting records

SYNOPSIS

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

DESCRIPTION

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

- **−a** Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, and "hog factor", as in *acctcom*(1). Output is normally sorted by total kcore-minutes.
- **−c** Sort by total CPU time, rather than total kcore-minutes.
- **−j** Combine all commands invoked only once under "***other".
- **−n** Sort by number of command invocations.
- **−s** Any file names encountered hereafter are already in internal summary format.

EXAMPLE

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

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

SEE ALSO

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

NAME

acctcon1, acctcon2 − connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

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

−p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

−t *Acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **−t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

−l *file* *File* is created to contain a summary of line usage showing line name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hang-up, termination of *login*(1) and termination of the login shell generate a logoff records, so that the number of logoffs is often three to four times the number of sessions. See *init*(1M) and *utmp*(4).

−o *file* *File* is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

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

EXAMPLE

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

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

FILES

/etc/wtmp

SEE ALSO

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

BUGS

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

NAME

acctmerg − merge or add total accounting files

SYNOPSIS

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

DESCRIPTION

Acctmerg reads its standard input and up to nine additional files, all in the **tacct** format (see *acct*(4)), 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.

EXAMPLE

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

```
acctmerg −v <file1 >file2
        edit file2 as desired...
acctmerg −a <file2 >file1
```

SEE ALSO

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

NAME

acctprc1, acctprc2 – process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

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

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

EXAMPLE

These commands are typically used as shown below:

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

FILES

/etc/passwd

SEE ALSO

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

BUGS

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

NAME

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

SYNOPSIS

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

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [mmdd]

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

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

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

/usr/lib/acct/startup

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

DESCRIPTION

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

Ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* 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.

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 executed via *cron*(1M) on the first day of each month. *Monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

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

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

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

where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing *prdaily*. Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct*.

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

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

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

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

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

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used if pacct gets large and during execution of daily accounting procedure
/etc/wtmp	login/logoff summary
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/acct/sum	summary directory, should be saved

SEE ALSO

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

NAME

badblk − program to set or update bad block information

SYNOPSIS

badblk [**−w**] [**−m** N] /dev/rXYZ [**#S**]

DESCRIPTION

Badblk sets or updates bad block information for those disk drives that support soft sector bad block remapping.

If invoked with the **−w** option, write/verify is performed to determine if there is a bad block; otherwise only read is done.

If invoked with the **−m**N option, the number of alternate blocks will be set to N. *Badblk* returns an error if N > NICALT (currently 50).

/dev/rXYZ is the device name.

#S is one or more block numbers separated by blanks.

If invoked with no specific block numbers and no bad block verification has been done before, then each block on the disk is checked (either read or write/verify) and bad block information in block 0 is set up from scratch.

If invoked with no specific block numbers, but block 0 already contains bad block information set up earlier, then a verification on the whole disk is performed; any new bad blocks not already on the block 0 table will be added.

If invoked with the device name plus block numbers, then only the indicated blocks are updated in block 0.

After alternate blocks are assigned, block 0 is updated and the updated blocks are verified to make sure alternate blocks are good. If alternate blocks are not good, new alternate block numbers are assigned.

The raw device that accesses the entire disk and allows for writing block zero should be specified.

EXAMPLE

```
badblk -w /dev/rw1hw0
```

does a full write/verify on winchester 1 and updates the header block. The rw1hw0 specifies raw (r) winchester 1 (w1), the full disk (h), with the capability of writing block 0 (w0).

```
badblk /dev/rw1hw0 3754 8123
```

adds blocks 3754 and 8123 to the badblock list.

NAME

bcopy — interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

Bcopy dates from a time when neither the UNIX file system nor disk drives were as reliable as they are now. *Bcopy* copies from and to files starting at arbitrary block (512-byte) boundaries.

The following questions are asked:

to: (you name the file or device to be copied to).
offset: (you provide the starting "to" block number).
from: (you name the file or device to be copied from).
offset: (you provide the starting "from" block number).
count: (you reply with the number of blocks to be copied).

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

Two consecutive carriage returns terminate *bcopy*.

SEE ALSO

cpio(1), dd(1).

NAME

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

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

/etc/powerfail

DESCRIPTION

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

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

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

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

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

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

SEE ALSO

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

NAME

chgnod – change current UNIX system nodename

SYNOPSIS

chgnod new-name [kernel-file]

DESCRIPTION

Chgrp accesses the structure defined in **<sys/utsname.h>**:

```
struct utsname {
        char    sysname[9];
        char    nodename[9];
        char    release[9];
        char    version[9];
};
```

Chgnod changes the nodename in the kernel-file argument to new-name. If no kernel-file is specified, /unix is assumed. *Nodename* is a null terminated string containing the name that the system is known by on a communications network.

New-name must be no longer than eight characters; longer names are truncated to eight. The old and new nodenames are printed on completion.

EXAMPLE

```
chgnod user10 /unix.current
```

changes the nodename of /unix.current to user10.

SEE ALSO

uname(2).

NAME

chroot — change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

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

Notice that:

```
chroot newroot command >x
```

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

This command is restricted to the super-user.

The new root path name 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.

EXAMPLE

```
chroot /users/asa ls /src
```

will cause the command "ls /src" to list the directory "/users/asa/src" since "/users/asa" is now effectively "/".

SEE ALSO

chdir(2).

BUGS

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

NAME

clri — clear i-node

SYNOPSIS

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

DESCRIPTION

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

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

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

EXAMPLE

clri /dev/yyyy n

where "yyyy" is a legitimate system device name, and "n" is the inode number to be cleared, will cause inode numbered "n" for device "/dev/yyyy" to be cleared to 64-bytes of 0s. Note: this instruction should only be used with caution.

SEE ALSO

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

BUGS

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

NAME

cron – clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file **/usr/lib/crontab**. Because *cron* never exits, it should be executed only once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(1M)).

The file **crontab** consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify in order:

minute (0-59),
hour (0-23),
day of the month (1-31),
month of the year (1-12),
and day of the week (0-6, with 0=Sunday).

Each of these patterns may contain:

a number in the (respective) range indicated above;
two numbers separated by a minus (indicating an inclusive range);
a list of numbers separated by commas (meaning all of these numbers); or
an asterisk (meaning all legal values).

The sixth field is a string that is executed by the shell at the specified time(s). A % in this field is translated into a new-line character. Only the first line (up to a % or the end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

Cron examines **crontab** once a minute to see if it has changed; if it has, *cron* reads it. Thus it takes only a minute for entries to become effective.

EXAMPLE

If the shell file **/etc/rc** contains the command line

```
/etc/cron
```

the clock daemon will be started every time **/etc/rc** is invoked, i.e., each time the system goes into multi-user mode after booting.

FILES

/usr/lib/crontab
/usr/adm/cronlog

SEE ALSO

init(1M), sh(1).

DIAGNOSTICS

A history of all actions by *cron* are recorded in **/usr/adm/cronlog**.

BUGS

Cron reads **crontab** only when it has changed, but it reads the in-core version of that table once a minute. A more efficient algorithm could be used. The overhead in running *cron* is about one percent of the CPU, exclusive of any commands executed by *cron*.

NAME

dcopy − copy file systems for optimal access time

SYNOPSIS

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

DESCRIPTION

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

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

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

−d leave order of directory entries as is (default is to move 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 not given, the values from the *inputfs* are used. UniPlus^{+}.I Dcopy catches interrupts and quits and reports on its progress. To terminate *dcopy*, send a quit signal and *dcopy* will no longer catch interrupts or quits. *Dcopy* also attempts to modify its command line arguments so its progress can be monitored with *ps*(1).

SEE ALSO

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

NAME

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

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

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

EXAMPLE

```
/etc/devnm /usr
```

produces

```
rp1 /usr
```

if **/usr** is mounted on **/dev/rp1**.

FILES

/dev/rp*, /dev/dsk*
/etc/mnttab

SEE ALSO

brc(1M), setmnt(1M).

NAME

df − report number of free disk blocks

SYNOPSIS

df [**−t**] [**−f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/dsk1**) or by mounted directory name (e.g., **/usr**). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

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

If the **−f** flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

/dev/dsk*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

NAME

diskformat - format a disk

SYNOPSIS

diskformat [-size #] [-dens #] [-cyl f[-t]] [-sec f[-t]] [-i] #] device

DESCRIPTION

Diskformat initializes a hard disk or floppy disk and formats it according to your specifications.

The following parameters may be specified ("device" is required):

device	device to be formatted (must be raw device)
−size #	specify sector size in bytes
−dens #	specify density
−cyl #[−#]	format cylinders *f* to *t* (default *f*). A specification such as #- means "until the end".
−head #[−#]	Format heads *f* to *t* (default *f*). A specification such as #- means "until the end".
−sec #[-#]	Format sectors *f* to *t* (default *f*). A specification such as #- means "until the end".
−il #	Interleave factor for the disk.

EXAMPLE

```
diskformat /dev/rfdc0 -dens 1 -size 128 -il 3
```

will format the floppy disk on drive 0, single density, 128 bytes per sector with an interleave factor of 3. This format is the only truly portable floppy format.

NAME

disktune - tune floppy disk settling time parameters

SYNOPSIS

disktune [−srt #] [−hlt #] [−hut #] device

DESCRIPTION

Disktune tunes floppy disk settling time parameters. These include the motor stepping rate and the rate at which the head loads and unloads. *Disktune* thus enables you to obtain the most efficient operation from your floppy on those systems that support it.

If no settable parameters are given, *disktune* will report the current settings on *device*. *Disktune* retains the current settings on parameters which are not specified.

The raw device, **/dev/rflop**, must be specified.

The settable parameters are:

−srt # seek motor stepping rate time in milliseconds

−hlt # head loading time in milliseconds

−hut # head unload time in milliseconds

EXAMPLE

```
disktune -srt 3 /dev/rfdc0
```

will set the step rate time on the floppy controller to 3 ms per step.

NAME

errdead – extract error records from dump

SYNOPSIS

/etc/errdead dumpfile [namelist]

DESCRIPTION

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

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

FILES

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

DIAGNOSTICS

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

SEE ALSO

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

NAME

errdemon – error-logging daemon

SYNOPSIS

/usr/lib/errdemon [file]

DESCRIPTION

The error logging daemon *errdemon* collects error records from the operating system by reading the special file **/dev/error** and places them in *file*. If *file* is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that *file* is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by *errdemon*; that responsibility is left to *errpt*(1M). The error-logging daemon is terminated by sending it a software kill signal (see *signal*(2)). Only the super-user may start the daemon, and only one daemon may be active at any time.

FILES

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

DIAGNOSTICS

The diagnostics produced by *errdemon* are intended to be self-explanatory.

SEE ALSO

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

NAME

errpt – process a report of logged errors

SYNOPSIS

errpt [options] [files]

DESCRIPTION

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

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

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

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

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

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

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

–d *devlist* A detailed report is limited to data about devices given in *devlist*, where *devlist* can be one of two forms: a list of device identifiers separated from one another by a comma, or a list of device identifiers enclosed in double quotes and separated from one another by a comma and/or more spaces. *Errpt* is familiar with the common form of identifiers (see Section 7 of this volume). The devices for which errors are logged are system dependent. Additional identifiers are **int** and **mem** which include detailed reports of stray-interrupt and memory-parity type errors respectively.

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

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

FILES

/usr/adm/errfile default error file

SEE ALSO

errdemon(1M), errfile(4).

NAME

errstop − terminate the error-logging daemon

SYNOPSIS

/etc/errstop [namelist]

DESCRIPTION

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

FILES

/unix default system namelist

DIAGNOSTICS

The diagnostics produced by *errstop* are intended to be self-explanatory.

SEE ALSO

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

NAME

ff − list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it to be a file system, saving i-node data for files which match the selection criteria. Output consists of the path name for each saved i-node, plus any other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node 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 i-node number after each path name.

−l Generate a supplementary list of all path names for multiply linked files.

−p *prefix* The specified *prefix* will be added to each generated path name. The default is ..

−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 i-node has been accessed in *n* days.

−m *n* Select if the i-node has been modified in *n* days.

−c *n* Select if the i-node has been changed in *n* days.

−n *file* Select if the i-node has been modified more recently than the argument *file*.

−i *i-node-list*
Generate names for only those i-nodes specified in *i-node-list*.

EXAMPLE

ff −I /dev/diskroot

generates a list of the names of all files on a specified file system.

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

produces an index of files and i-numbers which are on a file system and have been modified in the last 24 hours.

ff −i 451,76 /dev/rrp7

obtains the path names for i-nodes 451 and 76 on a specified file system.

SEE ALSO

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

BUGS

Only a single path name out of any possible ones will be generated for a multiply linked i-node, unless the **−l** option is specified. When **−l** is

specified, no selection criteria apply to the names generated. All possible names for every linked file on the file system will be included in the output. On very large file systems, memory may run out before *ff* does.

NAME

filesave, tapesave – daily/weekly UNIX file system backup

SYNOPSIS

/etc/filesave.?
/etc/tapesave

DESCRIPTION

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

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

SEE ALSO

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

NAME

finc − fast incremental backup

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

−**c** *n* True if the i-node has been changed in *n* days.

−**n** *file* True for any file which has been modified more recently than the argument *file*.

EXAMPLE

```
finc −m −2 /dev/rdiskusr /dev/rtp0
```

writes a tape consisting of all files from file-system **/usr** modified in the last 48 hours.

SEE ALSO

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

NAME

frec − recover files from a backup tape

SYNOPSIS

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

DESCRIPTION

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

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

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

−f *reqfile* Specifies a file which contains recovery requests. The format is i-number:newname, one per line.

EXAMPLE

```
frec /dev/rmt0 1216:junk
```

recovers a file, i-number 1216 when backed-up, into a file named **junk** in your current working directory.

```
frec −p /usr/src/cmd /dev/rmt0 14156:a 1232:b
3141:/usr/joe/a.c
```

recovers files with i-numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**.

SEE ALSO

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

BUGS

While paving a path (i.e. creating the intermediate directories contained in a pathname) *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.

NAME

fsck, dfsck − file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [−y] [−n] [−sX] [−SX] [−t file] [−q] [−D] [−f] [file-systems]

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

DESCRIPTION

Fsck

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

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

The following options are interpreted by *fsck*.

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

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

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

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

−s3 (RP03)
−s4 (RP04, RP05, RP06)
−sBlocks-per-cylinder:Blocks-to-skip (for anything else)

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

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

−t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **−t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **−t** flag, *fsck* will prompt the operator for the name of the scratch file. The

file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

−q Quiet *fsck*. Do not print size-check messages in Phase 1. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock 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.

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

Inconsistencies checked are as follows:

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

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

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

Dfsck

Dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A − is the separator between the file system groups.

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

answer refers to the first or second file system group).

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

EXAMPLE

fsck /dev/rdisk0

checks the consistency of device *rdisk0.*

FILES

/etc/checklist contains default list of file systems to check.

SEE ALSO

clri(1M), ncheck(1M), checklist(4), fs(4), crash(8).
Setting up the UNIX System

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

DIAGNOSTICS

The diagnostics produced by *fsck* are intended to be self-explanatory.

WARNING

There are some areas that can only be corrected on a raw device with no sync. Some devices which are in use such as **/dev/console**, **/dev/systty**, **/dev/syscon** will not be corrected by *fsck*.

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [–]

DESCRIPTION

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

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

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

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

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

is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. 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 i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. Inodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

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

EXAMPLE

386i prints i-number 386 in an i-node format. This now becomes the current working i-node.

ln=4 changes the link count for the working i-node to 4.

ln=+1
increments the link count by 1.

fc prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd prints the first 32 directory entries for the root i-node of this file system.

d5i.fc changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.

512B.p0o
prints the superblock of this file system in octal.

2i.a0b.d7=3
changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

d7.nm="name"
changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.

a2b.p0d
prints the third block of the current inode as directory entries.

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

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

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

DESCRIPTION

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

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

EXAMPLE

fuser –ku /dev/dsk1?
: will terminate all processes that are preventing disk drive one from being unmounted if typed by the super-user, listing the process ID and login name of each as it is killed.

fuser –u /etc/passwd
: will list process IDs and login names of processes that have the password file open.

fuser –ku /dev/dsk1? –u /etc/passwd
: will do both of the above examples in a single command line.

Note that the device names for disks are system dependent.

FILES

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

SEE ALSO

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

NAME

fwtmp, wtmpfix − manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [−**ic**]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

Fwtmp

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

The argument −**ic** is used to denote that input is in ASCII form, and output is 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 − can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon1* 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, a **$** 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 *acctcon1* will fail when processing connect accounting records.

FILES

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

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

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

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **−h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **−t** flag plus *timeout* in seconds, specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds. The optional second argument, *speed*, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* what speed to initially run at, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate. (By typing a <*break*> character.) The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands the following types:

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

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

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, newline characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

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

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

A check option is provided. When *getty* is invoked with the **-c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), tty(7).

NAME

init, telinit − process control initialization

SYNOPSIS

/etc/init [0123456SsQq]

/etc/telinit [0123456sSQqabc]

DESCRIPTION

Init

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

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

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

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

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

When *init* comes up initially and whenever it switches out of *SINGLE USER* state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/syscon**, to those modes saved in the file **/etc/ioctl.syscon**. This file is

written by *init* whenever *SINGLE USER* mode is entered. If this file doesn't exist when *init* wants to read it, a warning is printed and default settings are assumed.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. If this is the first time *init* has entered a *run-level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

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

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

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

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

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

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

Telinit

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

0–6 tells *init* to place the system in one of the *run-levels* **0–6**.

a,b,c
tells *init* to process only those **/etc/inittab** file entries having the **a**, **b** or **c** *run-level* set.

Q,q tells *init* to re-examine the **/etc/inittab** file.

s,S tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/syscon**, is changed to the terminal from which the command was executed.

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

FILES

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

SEE ALSO

getty(1M), login(1), sh(1), who(1), kill(2), inittab(4), utmp(4).

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.

NAME

install − install commands

SYNOPSIS

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

DESCRIPTION

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

If no options or directories (*dirx* ...) are given, *install* 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 other than −**c** and −**f**.

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

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

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

SEE ALSO

make(1).

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

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

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

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

FILES

/etc/shutdown

SEE ALSO

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

NAME

link, unlink − exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

Link and *unlink* perform their respective system calls on their arguments, abandoning all error checking. These commands may only be executed by the super-user, who (it is hoped) knows what he or she is doing.

EXAMPLE

link file1 file2

creates a directory entry for "file2" with the same inode number as "file1". NOTE: *link* should be used with extreme caution.

SEE ALSO

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

NAME

lpadmin – configure the LP spooling system

SYNOPSIS

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

DESCRIPTION

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

Exactly one of the **-p**, **-d** or **-x** options must be present for every legal invocation of *lpadmin*.

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

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

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

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

-c*class* inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.

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

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

-i*interface* establishes a new interface program for *P*. *Interface* is the path name of the new program.

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

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

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

-v*device* associates a new *device* with printer *P*. *Device* is the path-name of a file that is writable by the LP administrator, *lp*. Note that there is nothing to stop an administrator from

associating the same *device* with more than one *printer*. If only the **-p** and **-v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

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

Models.

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

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

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

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

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

- **-c** compressed print
- **-e** expanded print

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

EXAMPLE

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

```
/usr/lib/lpadmin -php2 -mhp
```

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

```
lp -dhp2 -o-c files
```

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

```
/usr/lib/lpadmin -pst1 -v/dev/tty20 -m1640
```

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

```
nroff -T450 files | lp -dst1 -of
nroff -T450-12 files | lp -dst1 -of
nroff -T37 files | col | lp -dst1
```

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

```
lp -dst1 -o12 /etc/passwd
```

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

FILES

/usr/spool/lp/*

SEE ALSO

450(1), accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

mkfs – construct a file system

SYNOPSIS

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

DESCRIPTION

Mkfs constructs a 1024-byte file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *sectors* interpreted as a decimal number. This is the number of 512-byte sectors the file system will occupy. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of *logical blocks* divided by 4.

If the second argument is a file name 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 newlines. The first token is the name of a file to be copied onto block zero as the bootstrap program. The second token is a number specifying the size of the created file system in 512-byte sectors. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of inodes in the file system. The maximum number of inodes configurable is 65500. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID, the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

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

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

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

A sample prototype specification follows:

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

```
        c0      c--644 3 1 0 0
        $
$
```

In both command syntaxes, the rotational *gap* and the number of *sectors/cyl* can be specified.

The *default* will be used if the supplied *gap* and *sectors/cyl* are considered illegal values or if a short argument count occurs.

EXAMPLE

```
mkfs /dev/fd0 2000 8 50
```

makes a file system on device */dev/fd0* in which 2000 is number of physical sectors in the file system, 8 is a sector interleave number which is used to stagger the blocks in the free list for more rapid reading (logical blocks in the free list are allocated every eighth physical sector) and 50 is a modulus that forces *mkfs* to first allocate all sectors in the first cylinder, then the next cylinder, etc.

NOTE: The proper selection of the *gap* and *sectors/cyl* parameters can improve disk efficiency. Disks which have full or partial track buffering should specify a *m* and *n* of 2 and 2; *m* and *n* for other disks must be determined by trial and error as the disk latency is related to rotational latency and cpu speed.

SEE ALSO

dir(4), fs(4), boot(8).

BUGS

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

Because of the conversion from physical sectors to logical blocks necessary for 1024-byte file systems, *blocks*, *gap*, and *blocks/cyl* should all be multiples of 2.

NAME

mkfs1b − construct a file system

SYNOPSIS

mkfs1b special size [m n]
mkfs1b special proto

DESCRIPTION

Mkfs1b constructs a file system by writing on the special file *special*. In the first form of the command a numeric size is given and *mkfs1b* builds a file system with a single empty directory on it. The number of i-nodes is calculated as a function of the filesystem size. *M* is an interleave factor for building the freelist and *n* is a modulo for *m*. See the example for usage.

N.B.: All filesystems should have a *lost+found* directory for *fsck*(1M); this should be created for each file system by running *mklost+found*(1M) in the root directory of a newly created file system, after the file system is first mounted.

In bootstrapping, the second form of *mkfs1b* is sometimes used. In this form, the file system is constructed according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto sector zero as the bootstrap program. The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the number of i-nodes in the i-list. The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user ID the group ID, and the initial contents of the file. The syntax of the contents field depends on the mode.

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

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

If the file is a regular file, the next token is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkfs1b* makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token **$**.

A sample prototype specification follows:

```
/usr/mdec/uboot
4872 55
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

EXAMPLE

```
mkfs1b /dev/fd0 2000 7 50
```

makes a file system in which 2000 is the total size of the file system to be put on **/dev/fd0**; 7 is a sector interleave number which is used to stagger the disk blocks for more rapid reading, every 7 blocks, and 50 is a modulo operator that forces the sector interlace number first to allocate all blocks in the first 50 sectors, then the next 50, etc.

NOTE: The proper selection of the *m* and *n* parameters can improve disk efficiency. Disks which have full or partial track buffering should specify a *m* and *n* of 1 and 1. *M* and *n* for other disks must be determined by trial and error as the disk latency is related to rotational latency and cpu speed.

SEE ALSO

fsck(1M), mklost+found(1M), dir(4).

BUGS

The default is 3500, which is probably not useful on any disk.
There should be some way to specify links.
There should be some way to specify bad blocks.
Should make *lost+found* automatically.

NAME

mklost+found — make a lost+found directory for fsck

SYNOPSIS

mklost+found

DESCRIPTION

A directory **lost+found** is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(1M). This command should be run immediately after first mounting and changing directory to a newly created file system. For small file systems, it is sufficient (and much faster) to simply make a lost+found directory. Up to 30 files can be recovered in it.

EXAMPLE

```
mklost+found
```

in the current directory, creates a directory with empty slots named **lost+found**.

SEE ALSO

fsck(1M), mkfs(1M)

BUGS

Should be done automatically by *mkfs*.

NAME

mknod – build special file

SYNOPSIS

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

DESCRIPTION

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

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file **conf.c**.

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

EXAMPLE

```
mknod /dev/tty4 c 3 4
```

would create file **/dev/tty4** as a character special device with major number 3 and minor number 4.

SEE ALSO

mknod(2).

NAME

mount, umount − mount and dismount file system

SYNOPSIS

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

/etc/umount special

DESCRIPTION

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

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

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

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

FILES

/etc/mnttab mount table

EXAMPLE

```
mount /dev/xxxx /t
```

mounts device **/dev/xxxx** as file system **/t.**

SEE ALSO

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

DIAGNOSTICS

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

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

BUGS

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

NAME

mvdir − move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

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

Only super-user can use *mvdir*.

EXAMPLE

```
mvdir dir1 dir2
```

renames existing directory "dir1" to be a new directory "dir2".

SEE ALSO

mkdir(1).

NAME

ncheck – generate names from i-numbers

SYNOPSIS

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

DESCRIPTION

N.B.: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(1M).

Ncheck with no argument generates a path name vs. i-number list of all files on a set of default file systems. Names of directory files are followed by /.. The **–i** option reduces the report to only those files whose i-numbers follow. The **–a** option allows printing of the names . and .., which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

EXAMPLE

```
ncheck /dev/rdisk1
```

will report the pathnames and i-numbers of files on the specified device.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

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

NAME

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

SYNOPSIS

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

DESCRIPTION

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

Prfld 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 *namelist*.

Prfstat 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.

Prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* 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*.

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

FILES

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

NAME

pstat − print system facts

SYNOPSIS

pstat [**−aixptuf**] [suboptions] [file]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *file* is given, the tables are sought there, otherwise in **/dev/mem.** The required namelist is taken from **/unix.** Options are:

-a Under **−p**, describe all process slots rather than just active ones.

-i Print the inode table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- L locked
- U update time *fs*(5) must be corrected
- A access time must be corrected
- M file system is mounted here
- W wanted by another process (L flag is on)
- T contains a text file
- C changed time must be corrected

CNT Number of open file table entries for this inode.

DEV Major and minor device number of file system in which this inode resides.

INO I-number within the device.

MODE Mode bits, see *chmod*(2).

NLK Number of links to this inode.

UID User ID of owner.

SIZ/DEV Number of bytes in an ordinary file, or major and minor device of special file.

-x Print the text table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

- T *ptrace*(2) in effect
- W text not yet written on swap device
- L loading in progress
- K locked
- w wanted (L flag is on)

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Core address, measured in multiples of core clicks (machine dependent).

SIZE Size of text segment, measured in multiples of core clicks (machine dependent).

IPTR Core location of corresponding inode.

CNT Number of processes using this text segment.

CCNT Number of processes in core using this text segment.

-p Print process table for active processes with these headings:

LOC The core location of this table entry.

S Run state encoded thus:

- 0 no process
- 1 waiting for some event
- 3 runnable

4 being created
5 being terminated
6 stopped under trace

F Miscellaneous state variables, or-ed together:
01 loaded
02 the scheduler process
04 locked
010 swapped out
020 traced
040 used in tracing
0100 locked in by *lock*(2).

PRI Scheduling priority, see *nice*(2).
SIGNAL Signals received (signals 1-16 coded in bits 0-15),
UID Real user ID.
TIM Time resident in seconds; times over 127 coded as 127.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see *nice*(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
ADDR If in core, the physical address of the "u-area" of the process measured in multiples of 64 bytes. If swapped out, the position in the swap area measured in multiples of 512 bytes.
SIZE Size of process image in multiples of 64 bytes.
WCHAN Wait channel number of a waiting process.
LINK Link pointer in list of runnable processes.
TEXTP If text is pure, pointer to location of text table entry.
CLKT Countdown for *alarm*(2) measured in seconds.

-t Print table for terminals (only DH11 and DL11 handled) with these headings:
RAW Number of characters in raw input queue.
CAN Number of characters in canonicalized input queue.
OUT Number of characters in output queue.
MODE See *tty*(4).
ADDR Physical device address.
DEL Number of delimiters (newlines) in canonicalized input queue.
COL Calculated column position of terminal.
STATE Miscellaneous state variables encoded thus:
W waiting for open to complete
O open
S has special (output) start routine
C carrier is on
B busy doing output
A process is awaiting output
X open for exclusive use
H hangup on close
PGRP Process group for which this is controlling terminal.

-u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or

the file used can be a core image and the address 0.

-f Print the open file table with these headings:

LOC The core location of this table entry.

FLG Miscellaneous state variables encoded thus:

R open for reading

W open for writing

P pipe

CNT Number of processes that know this open file.

INO The location of the inode table entry for this file.

OFFS The file offset, see *lseek*(2).

FILES

/unix namelist

/dev/mem default source of tables

EXAMPLE

```
pstat -i
```

displays all the active inodes in a table format with headings.

SEE ALSO

ps(1), stat(2), fs(5)

UNIX Implementation, by K. Thompson.

NAME

pwck, grpck — password/group file checkers

SYNOPSIS

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

DESCRIPTION

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

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

EXAMPLE

```
pwck
```

will list inconsistencies in **/etc/passwd**.

```
grpck
```

will list inconsistencies in **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).
"Setting up the UNIX System" in *UniPlus+ Administrator's Guide.*

DIAGNOSTICS

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

NAME

runacct – run daily accounting

SYNOPSIS

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

DESCRIPTION

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

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

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

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

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up 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. 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.

EXAMPLE

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

starts *runacct.*

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

restarts *runacct.*

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

restarts *runacct* at a specific *state.*

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(4), utmp(4).
"UNIX Accounting System" in the *UniPlus+ Administrator's Guide.*

DIAGNOSTICS

The accounting system will start complaining with ***RECOMPILE pnpsplit WITH NEW HOLIDAYS*** after the last holiday of the year. See "The UNIX Accounting System" for more on how to correct this condition. Other diagnostics are placed in various error and log files.

BUGS

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

runacct *mmdd* **WTMPFIX**

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

NAME

sa1, sa2, sadc − system activity report package

SYNOPSIS

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

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

/usr/lib/sa/sa2 [−**ubdycwaqvm**] [−s time] [−e time] [−i sec]

DESCRIPTION

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

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

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

```
su adm −c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`&"
```

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

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

```
0 * * * 0,6 su adm −c "/usr/lib/sa/sa1"
0 8−17 * * 1−5 su adm −c "/usr/lib/sa/sa1 1200 3"
0 18−7 * * 1−5 su adm −c "/usr/lib/sa/sa1"
```

will produce records every 20 minutes during working hours and hourly otherwise.

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

```
5 18 * * 1−5 su adm −c "/usr/lib/sa/sa2 −s 8:00 −e 18:01 −i 3600 −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 */
        int  szinode;               /* current entries of inode table */
        int  szfile;                /* current entries of file table */
        int  sztext;                /* current entries of text table */
        int  szproc;                /* current entries of proc table */
        int  mszinode;              /* size of inode table */
        int  mszfile;               /* size of file table */
        int  msztext;               /* size of text table */
        int  mszproc;               /* size of proc table */
        long inodeovf;              /* cumul. overflows of inode table */
        long inodeovf;              /* cumul. overflows of file table */
        long textovf;               /* cumul. overflows of text table */
        long procovf;               /* cumul. overflows of proc table */
        time_t ts;                  /* time stamp, seconds */
        long devio[NDEVS][4];       /* device info for up to NDEVS units */
#define IO_OPS              0  /* cumul. I/O requests */
#define IO_BCNT             1  /* cumul. blocks transferred */
#define IO_ACT              2  /* cumul. drive busy time in ticks */
#define IO_RESP             3  /* cumul. I/O resp time in ticks */
};
```

FILES

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

SEE ALSO

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

NAME

setmnt – establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

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

filesys node

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

EXAMPLE

/etc/devnm / | grep -vv swap | grep -v root | /etc/settmnt

will put an entry for the root file system and the device on which it is mounted into the file **/etc/mnttab** (except if it is mounted on a device named "swap" or "root").

FILES

/etc/mnttab

SEE ALSO

mnttab(4).

BUGS

Evil things will happen if *filesys* or *node* are longer than 10 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

shutdown – terminate all processing

SYNOPSIS

/etc/shutdown

DESCRIPTION

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

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

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

All file systems' super blocks are updated before the system is to be stopped (see *sync*(1)). This must be done before re-booting the system, to insure file system integrity. The most common error diagnostic that will occur is *device busy*. This diagnostic happens when a particular file system could not be unmounted.

SEE ALSO

mount(1M), sync(1).

NAME

updater – update files between two machines

SYNOPSIS

updater [**key**] local remote ...

DESCRIPTION

Updater updates files between two machines.

One of the following key letters must be included:

t Take files from the remote machine, updating the local machine.

p Put files from the local machine onto the remote machine, updating the remote machine.

d List the difference between files on the local and remote machines.

The following key letters are optional:

u Update a file only if it exists on both machines; this is the default condition.

r Replace a file if it did not exist on the destination machine.

Local refers to the local directory name.

Remote refers to the remote directory names. Only one remote name can be specified if the **p** (put) key is specified.

ALGORITHM

Open **/dev/tty0** to the remote machine.

Stty the local port and send a *stty* command to the remote machine to condition both ends of the connection.

Send a "cd remote ; sumdir . | sort +2 > /tmp/rXXXXX" to remote machine for each remote system; "cd local ; sumdir . | sort > /tmp/lXXXXX" for local machine.

Wait for remote to complete.

Take /tmp/rXXXXX.

Do a comparison between the local and the union of the remotes:

exists on remote only:
: If both the **t** and **r** keys are specified, take the file; otherwise list the file.

exists on local only:
: If both **p** and **r** keys are specified, put the file; otherwise list the file.

exist on both but different:
: If **t** key is specified, take the file.
: If **p** key is specified, put the file.
: If **d** key is specified, list the file.

same:
: nothing

EXAMPLE

```
updater d . .
```

uses **/dev/tty0** to communicate with a remote machine, and compares directories on the remote and local systems.

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options]

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

–d*directory* Clean *directory* instead of the spool directory.

–p*pre* Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following will cause all files older than the specified time to be deleted.

–n*time* Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)

–w*file* The default action for *uuclean* is to remove files which are older than a specified time (see **–n** option). The **–w** option is used to find those files older than *time* hours, however, the files are not deleted. If the argument *file* is present the warning is placed in *file*, otherwise, the warnings will go to the standard output.

–s*sys* Only files destined for system *sys* are examined. Up to 10 **–s** arguments may be specified.

–m*file* The **–m** option sends mail to the owner of the file when it is deleted. If a *file* is specified then an entry is placed in *file*.

This program is typically started by *cron*(1M).

EXAMPLE

```
uuclean -pT -pRC -n0 -m
```

removes all files in **/usr/spool/uucp** with a prefix of T or RC, and mails notifications to the owners of the removed files.

FILES

/usr/lib/uucp directory with commands used by *uuclean* internally
/usr/spool/uucp spool directory

SEE ALSO

cron(1M), uucp(1C), uux(1C).

NAME

uusub − monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

Uusub defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

- **−a***sys* Add *sys* to the subnetwork.
- **−d***sys* Delete *sys* from the subnetwork.
- **−l** Report the statistics on connections.
- **−r** Report the statistics on traffic amount.
- **−f** Flush the connection statistics.
- **−u***hr* Gather the traffic statistics over the past *hr* hours.
- **−c***sys* Exercise the connection to the system *sys*. If *sys* is specified as **all**, then exercise the connection to all the systems in the subnetwork.

The meanings of the connections report are:

sys #call #ok time #dev #login #nack #other

where *sys* is the remote system name, *#call* is the number of times the local system tries to call *sys* since the last flush was done, *#ok* is the number of successful connections, *time* is the latest successful connect time, *#dev* is the number of unsuccessful connections because of no available device (e.g. ACU), *#login* is the number of unsuccessful connections because of login failure, *#nack* is the number of unsuccessful connections because of no response (e.g. line busy, system down), and *#other* is the number of unsuccessful connections because of other reasons.

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

where *sfile* is the number of files sent and *sbyte* is the number of bytes sent over the period of time indicated in the latest *uusub* command with the **−u***hr* option. Similarly, *rfile* and *rbyte* are the numbers of files and bytes received.

EXAMPLE

uusub −c all −u 24

is typically started by *cron*(1M) once a day.

FILES

/usr/spool/uucp/SYSLOG	system log file
/usr/lib/uucp/L_sub	connection statistics
/usr/lib/uucp/R_sub	traffic statistics

SEE ALSO

uucp(1C), uustat(1C).

NAME

vchk − version checkup

SYNOPSIS

vchk [argument] ...

DESCRIPTION

Vchk is a highly specialized form of *make*(1) designed to check and maintain the modes, ownerships, and versions of a set of files specified in the *description file.* The description file is essentially a "photograph" of what a healthy system (i.e., one with all its components in the correct state) looks like. It contains a list of pathnames (for both files and directories) that should exist and have specific protections and contents. *Vchk* reads the description file, checks each item specified and prints error messages when a file does not match its description. Many problems can be fixed directly by *vchk*, such as incorrect mode and/or owner and missing link names. All other problems involve actually replacing a file, detected by comparing some combination of checksum, length, and/or version number (from the description file) with the value generated from the actual file being checked. When a file needs to be replaced *vchk* invokes the command named by the REMAKE macro (see MACROS below).

Each *argument* is either a definition or an option. Option arguments begin with the character − and consist of a string of letters (called *flags*) from the set **ADIPSabcdefiklmprstvx**. The **f** and **t** flags cause the next argument to be considered specially. The **p** and **P** flags cause the rest of the argument in which they appear to be considered specially. Other arguments are either macro definitions (i.e., *name* = *string* pairs) or simply strings which are saved as numeric macros. Briefly, the *flags* are as follows:

−A*sysid*
: specifies an alternate *sysid* rather than using the one found in **/etc/sys_id**.

−D enables debugging messages.

−I process control lines only.

−P*sysid*
: preprocess the description file; *sysid* is optional and is explained below under PREPROCESSING.

−S suppress printing of non-fatal error messages.

−a checks all lines in the description file. Modifies the **b**, **P**, and **k** options.

−b build a description file for the current directory.

−c print shell commands to fix the file system.

−d suppress re-installation commands and error messages.

−e suppress checks for everything but existence.

−f *filename*
: cause *vchk* to read filename instead of **/etc/vchk_tree**.

−i go interactive: modifies **b**, **c**, and **x** options.

−k perform checksums on files having checksum field.

-l suppress listing of files left in directories.

-m allow multiple copies of files.

-p*pw_file*
force *vchk* to re-evaluate and/or use an alternate password file.

-r allow redundant password entries (user ID1).

-s remain silent about trivial problems.

-v suppress checking of version numbers.

-x execute shell commands to fix the file system (cf. the **-c** option which prints rather than executes).

DESCRIPTION FILE SYNTAX

Lines in the description file are either comments, control lines, specifications, or commands. Control lines provide a simple *ifdef* mechanism for selectively ignoring specification lines. Specification lines describe files and/or directories that need to be checked. Commands are not processed by *vchk* but (in the spirit of *make*(1)) are used when the file specified above them is found to need replacing.

Several conventions are observed to maintain the readability of the description file; for example, a trailing backslash and all leading white space on the following line are ignored when processed. In addition, backslash may be used to delay the expansion of macros (in macro definitions only) and, as described below, to alter the evaluation of parentheses and braces in pathnames.

COMMENTS

Comment lines always start with a '#'. If the second character on a comment line is also a '#', then that line is printed on the standard error when read by *vchk*. Any line may have a trailing comment, which is universally ignored.

CONTROL LINES

Control lines start with a '.' (period). The mechanism is similar to the C language pre-processor except that defined words do not have values associated with them; words are simply defined or not. The control commands supported are as follows:

.define wordlist
where *wordlist* is a list of white-space-separated words to be defined which have no associated values. Note that only the first twelve letters of defined words are significant. Storage for defined words is static. There is a maximum of sixty defined words at any given time.

.ifdef define_expression
where *define_expression* is an infix boolean expression involving defined words and the operators !, &, and |, which mean 'not', 'and', and 'or', respectively. The value of each word evaluates to a boolean "yes" if that word is defined and "no" if not. If the expression evaluates to be false, lines are ignored until the matching *.endif* or *.else* control line is read. *Ifndef* is also supported and reverses the sense of the expression test.

.include filename
is very similar the C pre-processor *include* with the exception that there are no default searching places and that the filename is not en-

closed in double quotes or angle brackets. In addition, if the first character of the filename is an exclamation mark (!), then the rest of line is considered to be a shell command and its standard output is what gets included.

.undef wordlist
undefines each word in the wordlist.

.unset wordlist
frees the storage associated with macro definitions (detailed in a following section) for the given wordlist (which is composed of macro names). Words in wordlist that are already unset (or were never set) are silently ignored.

.chdir directory_name
changes the current directory to the one specified and alters the starting location of pathnames anchored from the current directory.

.exit message
causes *vchk* to print the message and exit immediately.

As an aid to debugging the description file, a single '.' on a line by itself causes *vchk* to print the currently defined control symbols on the standard error when it reads that line.

MACROS

A macro processing facility very similar to the one used by *make*(1) is provided. Macros are defined when a line containing the macro name, an equal sign, and the value is read. The value may be null or include macro invocations. Unlike 'defined' words, macro names are fully significant and are saved in dynamic memory. Macros are invoked by the '$' character. As in *make* scripts, macro names must be surrounded by ()s when they are longer than a single letter. There is a special macro (named '.', thus referenced with '$.') which always expands to the name of the current directory. It is useful in the construction of link names since most files have their links close by.

Except in the definition of a new macro (where interpretation may need to be delayed) and in comments, it is always an error for a macro to be used if it is undefined. Since the '##' comment is printed after macro substitution, it is a useful debugging tool. In keeping with the spirit of the 'dump control words' command (.), a single '$' on a line by itself prints all the currently defined macros and their values.

Note that lines are re-scanned once a macro has been substituted so that a macro may be defined to expand to a control line, comment, or even a macro definition. Note that this degenerates to a recursive loop if the definition of a macro contains a reference to that macro.

Two predefined macros exist. The first, called REMAKE, contains the name (and options) of the program to use to replace damaged or missing files. If the file **/etc/sys_id** exists and is not empty, it is assumed to contain the *UniSoft* code name assigned to your system and *vchk* will setup the REMAKE macro to be the command "take −iN" where N is the name found in **/etc/sys_id**. This allows systems that reside at *UniSoft* to be updated automatically over a direct *tty* line via the *take*(1M) program. If the **/etc/sys_id** file has a single empty line in it, then REMAKE will be set to "take −i". This allows remote systems to be updated automatically over phone lines. If the **/etc/sys_id** file does not exist, *vchk* sets the REMAKE

macro to be "install". Note that the description file may redefine the REMAKE macro at any time.

The second predefined macro is called ARGS and can be set but not referenced. Strings assigned to ARGS are treated as command line options. The − preceding option keyletters is still required, and enables that option. A plus sign, '+', must be used instead to disable a keyletter option. Resetting the **b**, **p**, and **r** is not allowed.

DESCRIPTION LINES

Each line that is not a comment, control line, macro definition, or command is considered to be the specification for a particular file or directory. These have a simple and regular syntax: the first and only mandatory field is the *pathname*, which must begin at the root (/) or the current directory (./). In practice we find that starting all lines with a macro allows easy relocation of the entire tree described and is very readable.

The rest of the line contains optional information about the contents and protection of the file. Contents specifications are separated from the pathname by white space. The entire protection specification is bracketed to separate it from the rest of the line.

PATHNAMES

Pathnames refer to directories (if and only if they end in a '/' character) or files (if they do not end in a '/'). Use of shell metacharacters (*globbing*) is not supported but two mechanisms are provided to allow variable pathnames: braces, {}s, and parentheses. Braces are interpreted just as in *csh*(1); each of the expanded pathnames must exist and must match the description given. Parentheses in pathnames are interpreted similarly, except that *exactly one* of the resulting pathnames must exist. This feature is useful, for example, to allow a program to be in either **/bin** or **/usr/bin**, but not both.

Parentheses and braces are expanded left to right; for example, the construction **(a,b){x,y}** means either **ax** and **ay** must exist or **bx** and **by**. Backslashes may be used to delay or prevent the interpretation of ()s and {}s. For example, **\(a,b\){x,y}** means one of **ax** or **bx** and one of **ay** or **by** must exist. One layer of backslashes are removed for each pass through the pathname and each time an unescaped parenthesis or bracket is detected and expanded, another pass is made.

Note that when alternative paths are used (i.e., parentheses occur in the pathname) the first one is considered the one to be rebuilt in the event that all are missing. For example, the pathname **(/usr)/bin/ls** would look first in **/usr/bin** for 'ls', then in **/bin**, and try to "REMAKE /usr/bin/ls" if both are missing. The reverse is true for **(,/usr)/bin/ls**.

SPECIFICATIONS

Two kinds of specifications are implemented. The first kind deals with the contents of the file or directory and follows the pathname (separated from it by white space). The second kind deals with the files protection; these are enclosed in some type of parentheses to separate them from contents specifications. The kind of parentheses used, ()s, []s, <>s, or {}s, modify the action taken by *vchk* according to the table below:

() Enables checking and replacing of the file.

[] Enables checking but never replacing. If the file is missing, *vchk* will complain but not try to rebuild it.

<> Enables checking (if and only if the file exists or the **-a** command line option is given) and never replacing.

{} Enables checking but not repairing, (i.e., if the file is missing then it will be remade, but if it exists and is incorrect it will not be remade).

Associated with each directory is a default mode and ownership for the files and directories contained within it. Unless explicitly given, each directory inherits its defaults from its parent directory. If unspecified, the uppermost directory (either the root or current directory) sets the mode and ownership of its contents from its own mode and ownership. These defaults may be reset at any time simply by following the directory name with a mode and/or user name.

Regular files have three optional contents specification fields: length, checksum, and version number. These may be specified with a word (either Length, Checksum, or Version), an optional space, or a numeric value. The word may be any prefix, for example, 'Length 34' or 'L34'. The checksums used are the same as those produced by *sum*(1). The length checked is that returned by *stat*(2). These checks do not apply to device files (only block and character devices are supported); thus their contents specification field must begin with either **b** or **c** and must be followed by the major and minor device number (separated by white space). If **x** is used instead of either the major or minor device number, *vchk* will allow the device to have any value.

The protection specification consists of a list of command prefixes separated by semicolons. The commands supported are *chmod, chown,* and *link to*. If angle brackets (i.e., <>s) are used instead of parentheses to enclose the protection specification, the file or directory so referenced is optional and will not generate diagnostics if it is missing. It can be raised to the status of []s by giving the **-a** command line option. If square brackets (i.e., []s) are used, the referenced file cannot be replaced automatically, as for example, the password file. If curly brackets (i.e., {}s) are used, the referenced file will be replaced only if it is missing, not if it exists and is wrong (according to the description file). This is useful for files like **/etc/termcap**.

Any other information in the protection specification is treated as a special comment that is printed with error messages about that file.

OPERATION

In order for *vchk* to check the ownership of files it must map user ID numbers onto login names. The password file is normally used for this mapping but it is too expensive to look up each name every time it is used so *vchk* creates a temporary file (**/etc/vchk_pw**) the first time it is run; whenever its temporary is out of date with respect to the real password file, *vchk* recreates the temporary file.

In the process of reading the password file *vchk* inspects each account and prints diagnostics when it finds questionable data there. These messages are warnings or simply situations which bear reporting; the format of these messages is "Line <number>: message". The word "Error: " is prepended to the warnings for a particular line if *vchk* has decided to ignore that line of the password file.

The **-p**[*pw_file*] option is provided to allow users who do not have write permission in **/etc** to use *vchk*. If specified with a filename after it, *vchk*

will get the saved password information from that file. If the file does not exist, *vchk* will create it. Use of the **-p** option without a filename informs *vchk* to reprocess the password file even if it is not out of date.

Vchk normally expects the description file to be **/etc/vchk_tree**, but if the standard input to *vchk* is a regular file, that file will be read instead.

Instead of redirecting the standard input, the **-f** option can be used to respecify the description file. It is an error to use both.

The best way to build a new description file is to *chdir* to the appropriate directory and run *vchk* with the **-b** option. A description file for the current directory will be produced on the standard output. The **-i** option may also be used, causing *vchk* to ask before descending each directory.

In addition to reporting errors, the **-c** flag prints shell commands to correct the detected error. The **-i** option can be used with the **-c** option to ask before outputting a command.

It is inadvisable to use the **-x** flag until the description file has been used and debugged. This flag allows *vchk* to execute the *chmod*(2), *chown*(2), and *ln*(2) commands internally, saving much time. Re-installation commands (cf. the REMAKE macro) are executed via the *system*(3) call.

PREPROCESSING

The **-P***sysid* command line option provides a means for simplifying a complex description file. Everything except macro substitution is suppressed and after each line is parsed, it is printed on the standard output instead of being used to check the filesystem.

If a *sysid* is given after the **-P** flag, then it is used to lookup a line from the (**/etc/takelist**) file. (See *take*(1m) for a more complete description of the function of the **/etc/takelist** file.)

The lines in **/etc/takelist** are composed of any number of fields (called alternates) separated by colons (:s). The first alternate in a line is a list of system names separated by or bars (|s). The *sysid* above is compared with each of the system names in the first alternate of each line until it is found. If not found, then *vchk* exits with an appropriate error message.

When the line from **/etc/takelist** for the current *sysid* is found, then each of the additional alternates are considered lists of root directories (separated by colons or bars) to be prepended to filenames in the tree file before looking for them.

If a file is found in more or less than exactly 1 place in the list, then an error is reported and that line is not include in the preprocessed output. If it is found, then the checksum, length, and version number are computed from that file and replaced in the preprocessed output.

EXAMPLE

Following are some excerpts from a typical description file.

```
B = (/usr)/bin              # programs can be in /bin or /usr/bin

/           bin 755         (chmod 755)
$B/ar       Version 1.0
$B/awk      Version 1.3
$B/more     Version 1.0     (link to $./page)
$B/sccsdiff C54686 L1253    (shell script)
$B/su       Version 1.0     (chown root; chmod 4755)
```

```
/etc/          root 644
/etc/passwd                    [password file]
/etc/group                     [group file]
/etc/init      Version 1.0     (chmod 700)
/etc/update    Version 1.1     (chmod 700)
/etc/ddate                     <dump dates>
```

The first line of the above example defines a macro, *B*, to be the string **(/usr)/bin**. This macro is then invoked on lines 3 through 7 of the example to allow the programs mentioned to be in either **/usr/bin** or **/bin**.

The second line specifies that the root directory (/)**should**have**mode** that the default mode and owner for files found in it be 755 and **bin.**

The third line specifies that the *ar* program should be version 1.0, owned by **bin** and have mode 755. The mode and owner are implied in the following way. Each directory inherits its mode and ownership from its parent. Thus **/bin** inherits the owner of root (which is unspecified in the example and thus defaults to whatever the owner of the root (/) is when the example is run). The mode of the root directory is specified as 755.

FILES

/etc/vchk_pw	the file where *vchk* saves the password file summary.
/etc/passwd	the password file.
/etc/vchk_tree	the default description file.
/dev/tty	where *vchk* prints questions and gets the responses (when the **−i** option is used).
<standard error>	used to print all diagnostics.
<standard output>	used to print shell commands and the newly built description file (when using the **−b** option).
<standard input>	considered the default description file if it is a regular file.

SEE ALSO

chmod(1), ln(1), chown(1M).

BUGS

There is no way (except tediously via the **−i** option) to exclude directories from inspection when building a new description file. There is also no way to automatically update an existing description file (i.e., to tell *vchk* to fix the description file instead of the filesystem).

NAME

volcopy, labelit – copy file systems with label checking

SYNOPSIS

/etc/volcopy [options] fsname special1 volname1 special2 volname2

/etc/labelit special [fsname volume [**–n**]]

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

- **–a** invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made,
- **–s** (default) invoke the **DEL if wrong** verification sequence.

Other *options* are used only with tapes:

- **–bpi**density bits-per-inch (i.e., **800/1600/6250**),
- **–feet**size size of reel in feet (i.e., **1200/2400**),
- **–reel**num beginning reel number for a restarted copy,
- **–buf** use double buffered I/O.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives.

The *fsname* argument represents the mounted name (e.g.: **root**, **u1**, etc.) of the filsystem being copied.

The *special* should be the physical disk section or tape (e.g.: **/dev/rdsk15**, **/dev/rmt0**, etc.).

The *volname* is the physical volume name (e.g.: **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be – to use the existing volume name.

Special1 and *volname1* are the device and volume from which the copy of the file system is being extracted. *Special2* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6];**).

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, *labelit* prints current label values. The **–n** option provides for initial labeling of new tapes only (this destroys previous contents).

EXAMPLE

```
volcopy newsys /dev/rrp15 1 /dev/rfd0 1
```

copies volume 1 of the file system labeled *newsys* mounted on **/dev/rrp15** onto volume 1 of **/dev/rfd0**.

```
labelit /dev/rfd0 oldsys save
```

relabels the file system mounted on **/dev/rfd0** with a new *fsname* of *oldsys* and a new *volname* of save.

FILES

/etc/log/filesave.log a record of file systems/volumes copied

SEE ALSO

fs(4).

BUGS

Only device names beginning **/dev/rmt** are treated as tapes. Tape record sizes are determined both by density and by drive type. Records are 5,120 bytes long at 800 and 1600 bits-per-inch, and 25,600 bytes long at 6250 bits-per-inch.

NAME

wall – write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message to all currently logged in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see *mesg*(1)).

EXAMPLE

wall

will broadcast the standard input to all users who are not protected against receiving messages by the *mesg* command.

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

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

NAME

whodo — who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces merged, reformatted, and dated output from the *who*(1) and *ps*(1) commands.

EXAMPLE

/etc/whodo

will return something like the following:

```
UNIX
co  root    13:52
co     60   0:01  sh
co     61   0:01  ps
co     62   0:00  sh
```

SEE ALSO

ps(1), who(1).

NAME

intro — introduction to special files

DESCRIPTION

This section describes special files that refer to specific hardware peripherals and UNIX System device drivers. The names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX System device driver are discussed where applicable.

BUGS

While the names of the entries *generally* refer to vendor hardware names, in certain cases these names are seemingly arbitrary for various historical reasons.

NAME

aliases – aliases file for delivermail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user ID aliases that will be used by **/etc/delivermail**. It is formatted as a series of lines of the form

name:addr1,addr2,...addrn

The *name* is the name to alias, and the *addri* are the addresses to send the message to. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops cannot occur since no message will be sent to any person more than once.

SEE ALSO

delivermail(8N).

NAME

err — error-logging interface

DESCRIPTION

Minor device 0 of the *err* driver is the interface between a process and the system's error-record collection routines. The driver may be opened only for reading by a single process with super-user permissions. Each read causes an entire error record to be retrieved; the record is truncated if the read request is for less than the record's length.

FILES

/dev/error special file

SEE ALSO

errdemon(1M).

NAME

/etc/hosts -- host table for bnet

DESCRIPTION

The BNET host table is organized as follows:

<inet-addr> <delim> <host-name> [<delim> <host-nicname>]

where

< inet-addr >

is a 32 bit type-a internet address, composed of one byte of "network number" followed by 3 bytes of local network address. Each byte is delimited by a period. For example,

0x27.0.1.1

refers to local-address 0x11 on network 0x27. Fields can be specified in decimal or hexadecimal notation. Normally, an installation will choose one network number for all hosts on the bnet (ethernet). Official type-a network numbers are assigned by the US DOD. Any network number less than 127 may be actually chosen if a particular installation is not planning to connect to the DCN network in the near future. Future releases of BNET will support type-b (16-bit) and type-c (24-bit) network numbers. See DCN/NIC RFC-790 for discussion of network numbers. Local-addresses are the low-order three bytes of the ethernet board address. The current release assumes a 3com ethernet controller is installed, so the upper three bytes of the ethernet address is determined. This restriction will be removed in the next release of BNET.

< delim >

is a SINGLE SPACE. (!).

< host-name >

is the official name of the host. For the local host, this field should be exactly the same as the contents of **/usr/lib/uucp/SYSTEMNAME**, though no checks are made anywhere for this equivelance at this time. Since *uucp* limits the length of hostnames to seven characters, so does BNET, but this restriction will dissapear in the next release of BNET.

< host-nicname >

is a nicname or alias for the officially named host. Hosts can have several nicnames.

LOCAL HOST

The local host, i.e., the loopback driver, has a distinguished entry in the host table:

127.0.0.1 myself

That is, network number 127, host number 1.

EXAMPLE

The following is a short host table for a network with two hosts.

```
39.0.1.14 jeff j
127.0.0.1 bill b
```

The local host is named "bill" or "b". The remote host is named "jeff" or "j", If the remote host has a 3com ethernet controller, then it would have ethernet number 0x02608C00010D. The 0x02608C is 3com's manufacturer's ethernet number.

NAME

mem, kmem — core memory

DESCRIPTION

Mem is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in *mem* are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file *kmem* is the same as *mem* except that kernel virtual memory rather than physical memory is accessed.

FILES

/dev/mem
/dev/kmem

NAME

null – the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

termio – general terminal interface

DESCRIPTION

This section describes both a particular special file and the general nature of the terminal interface.

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, the character **#** erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character **@** kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control-\ or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF (Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) is an additional line delimiter, like NL. It is not normally used.

STOP (Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

START (Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hangup* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hangup signal is ignored, any subsequent read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive.

If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define  NCC      8
struct   termio {
         unsigned short  c_iflag;     /* input modes */
         unsigned short  c_oflag;     /* output modes */
         unsigned short  c_cflag;     /* control modes */
         unsigned short  c_lflag;     /* local modes */
         char            c_line;      /* line discipline */
         unsigned char   c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	INTR	DEL
1	QUIT	FS
2	ERASE	#
3	KILL	@
4	EOF	EOT
5	EOL	NUL
6	reserved	
7	reserved	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all bits clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2 four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A

EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible

value (e.g. 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use*:
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA — Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

TCSETA — Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW — Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.

TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).

TCXONC Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.

TCFLSH If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FIONREAD Return the number of characters currently in a terminal's input buffer into the integer pointer rg. ICANNON mode must be set for this option to work.

FILES

/dev/tty
/dev/tty*
/dev/console

SEE ALSO

stty(1), ioctl(2).

NAME

tty – controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section outlines procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on the topics of boot procedures and recovery from crashes.

BUGS

No manual can take the place of good, solid experience.

NAME

boot – startup procedures

DESCRIPTION

A 68000 UNIX system is typically started by a two-stage process. The first is a primary bootstrap which is used to read in the system itself.

The primary bootstrap, when read into memory and executed, sets up memory management if necessary, and types a prompt message on the console. Then it reads from the console a device specification (see below) followed immediately by a pathname. This program finds the corresponding file on the given device, loads that file into the proper memory location, and then transfers control of the program. Normal line editing characters can be used.

Conventionally, the name of the current version of the system is '/unix'. Then, the recipe is:

1) Load the boot program by fiddling with the console keys and crt as appropriate for your hardware.

2) When the ":" prompt is given, type [for example]
fpy(0,0)unix
or
hd(0,0)unix

   depending on whether you are loading from floppy or hard disk, respectively. The first 0 indicates the physical unit number; the second indicates the block number of the beginning of the logical file system (device) to be searched. (See below).

   When asked for the device name, a list of valid device names can be obtained by typing a "?" followed by a carriage return. A carriage return by itself boots the UNIX system on the default device.

When the system is running, it types a "#" prompt. After doing any file system checks via *fsck*(1) and setting the date (*date*(1)), the system can be brought up for standard operation by typing **init 2** in response to the "#" prompt, then an EOT (control-d) when the system requests it.

Device Specifications

A device specification has the following form:

device(unit,offset)

where *device* is the type of the device to be searched, *unit* is the unit number of the device, and *offset* is the block offset of the file system on the device. *Device* specifications vary according to which 68000 UNIX system you are using. Check manufacturer's instructions for the device specifications.

For example, the specification

hp(1,7000)

would indicate an HP disk, unit 1, and the file system found starting at block 7000.

ROM Programs

Programs to call the primary bootstrap may be installed in read-only memories or manually keyed into main memory. Each program is position-independent but should be placed well above location 0 so it will

not be overwritten. See manufacturer's instructions for a manually keyed-in ROM boot program, should one become necessary.

FILES

/unix — system code

NAME

crash − what to do when the system crashes

DESCRIPTION

This entry gives at least a few clues about how to proceed if the system crashes. It can't pretend to be complete.

In restarting after a crash, always bring up the system single-user, as specified in *boot*(8) as modified for your particular installation. Then perform an *fsck*(1M) on all file systems which could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user.

To even boot UNIX at all, certain files (and the directories leading to them) must be intact. First, the initialization program **/etc/init** must be present and executable. For *init* to work correctly, **/dev/console**, **/bin/sh** and **/bin/env** must be present. If one of these does not exist, the symptom is best described as thrashing. *Init* will go into a *fork/exec* loop trying to create a Shell with proper standard input and output. The file **/etc/rc** should also be there and be executable; the system will come up but will not be fully initialized without it.

If you cannot get the system to boot, a runnable system must be obtained from a backup medium. The root file system may then be doctored as a mounted file system as described below. If there are any problems with the root file system, it is probably prudent to go to a backup system to avoid working on a mounted file system.

Repairing disks. The first rule to keep in mind is that an addled disk should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it. This is an area where experience and informed courage count for much.

Fsck(1M) is adept at diagnosing and repairing file system problems. It first identifies all of the files that contain bad (out of range) blocks or blocks that appear in more than one file. Any such files are then identified by name and *fsck* requests permission to remove them from the file system. Files with bad blocks should be removed. In the case of duplicate blocks, all of the files except the most recently modified should be removed. The contents of the survivor should be checked after the file system is repaired to ensure that it contains the proper data. (Note that running *fsck* with the **−n** option will cause it to report all problems without attempting any repair.)

Fsck will also report on incorrect link counts and will request permission to adjust any that are erroneous. In addition, it will reconnect any files or directories that are allocated but have no file system references to a "lost+found" directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) *fsck* will, with the operators concurrence, construct a new one.

Why did it crash? UNIX types a message on the console typewriter when it voluntarily crashes. Here is the current list of such messages, with enough information to provide a hope at least of the remedy. The message has the form "panic: ...", possibly accompanied by other information. Left unstated in all cases is the possibility that hardware or software error

produced the message in some unexpected way.

blkdev
The *getblk* routine was called with a nonexistent major device as argument. Definitely hardware or software error.

devtab
Null device table entry for the major device used as argument to *getblk*. Definitely hardware or software error.

dpfrelse
The list of processes currently mapped into the memory management unit has been lost (68451 only).

iinit
An I/O error reading the super-block for the root file system during initialization.

interrupt stack overflow
The kernel ran out of stack space on an interrupt. Subroutine depth is too great or too many local variables.

kernel memory management error
Bus error or address error in supervisor mode. Can be a software or hardware problem.

no fs
A device has disappeared from the mounted-device table. Definitely hardware or software error.

no imt
Like "no fs", but produced elsewhere.

no clock
During initialization, neither the line nor programmable clock was found to exist.

no procs
Process table has been destroyed.

I/O error in swap
An unrecoverable I/O error during a swap. Really shouldn't be a panic, but it is hard to fix.

oops!!! syscall
The interrupt vector for system calls is missing.

out of swap space
A program needs to be swapped out, and there is no more swap space. It has to be increased. This really shouldn't be a panic, but there is no easy fix.

timeout table overflow
The timeout table overflowed. The timeout table is not large enough or some routine is starting up too many timeouts.

trap
An unexpected trap has occurred within the system. This is accompanied by the following information:

trap type
2 bus error
3 address error

4 illegal instruction
5 divide by zero
6 CHK instruction
7 TRAPV instruction
8 privilege violation
9 trace
10 1010 emulator
11 1111 emulator
12-255 unexpected interrupt

virtual address (for bus/address errors only)
physical address
instruction register
function code

mmu dump

program counter
status register
program id

registers

In some of these cases it is possible for hex 1000 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred.

SEE ALSO

adb(1), fsck(1M), boot(8).

NAME

delivermail − deliver mail to arbitrary people

SYNOPSIS

/etc/delivermail [−[**fr**] *address*] [−**a**] [−**e**[**empqw**]] [−**n**] [−**m**] [−**s**] [−**i**] [−**h** *N*] address ...

DESCRIPTION

Delivermail delivers a letter to one or more people, routing the letter over whatever networks are necessary. *Delivermail* will do inter-net forwarding as necessary to deliver the mail to the correct place.

Delivermail is not intended as a user interface routine; it is expected that other programs will provide user-friendly front ends, and *delivermail* will be used only to deliver pre-formatted messages.

Delivermail reads its standard input up to a control-D or a single dot and sends a copy of the letter found there to all of the addresses listed. If the −**i** flag is given, single dots are ignored. It determines the network to use based on the syntax of the addresses. Addresses containing the character "@" or the word "at" are sent to BNET; and addresses containing "!" are sent to the UUCP net. Other addresses are assumed to be local.

Local addresses are looked up in the file **/usr/lib/aliases** and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash or using the −**n** flag. Normally the sender is not included in any alias expansions, e.g., if "john" sends to "group", and "group" includes "john" in the expansion, then the letter will not be delivered to "john". The −**m** flag disables this suppression.

Delivermail computes the person sending the mail by looking at your login name. The "from" person can be explicitly specified by using the −**f** flag; or, if the −**a** flag is given, *delivermail* looks in the body of the message for a "From:" or "Sender:" field in ARPANET format. The −**f** and −**a** flags can be used only by the special users *root* and *network*, or if the person you are trying to become is the same as the person you are. The −**r** flag is entirely equivalent to the −**f** flag; it is provided for ease of interface only.

The −e*x* flag controls the disposition of error output, as follows:

e Print errors on the standard output, and echo a copy of the message when done. It is assumed that a network server will return the message back to the user.

m Mail errors back to the user.

p Print errors on the standard output.

q Throw errors away; only exit status is returned.

w Write errors back to the user's terminal, but only if the user is still logged in and write permission is enabled; otherwise errors are mailed back.

If the error is not mailed back, and if the mail originated on the machine where the error occurred, the letter is appended to the file "dead.letter" in the sender's home directory.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *delivermail* from supressing

the blanks from between arguments.

The message is normally edited to eliminate "From" lines that might confuse other mailers. In particular, "From" lines in the header are deleted, and "From" lines in the body are prepended by ">". The -s flag saves "From" lines in the header.

The -h flag gives a "hop-count", i.e., a measure of how many times this message has been processed by *delivermail* (presumably on different machines). Each time *delivermail* processes a message, it increases the hop-count by one; if it exceeds 30 *delivermail* assumes that an alias loop has occurred and it aborts the message. The hop-count defaults to zero.

Delivermail returns an exit status describing what it did. The codes are defined in *mailexits.h*:

0	EX_OK	Succesful completion on all addresses.
2	EX_NOUSER	User name not recognized.
3	EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
4	EX_SYNTAX	Syntax error in address.
5	EX_SOFTWARE	Internal software error, including bad arguments.
6	EX_OSERR	Temporary operating system error, such as "cannot fork".
7	EX_NOHOST	Host name not recognized.

FILES

/usr/lib/aliases	to alias names
/bin/mail	to deliver local mail
/etc/netmailer	to deliver BNET mail
/bin/mail	to deliver UUCP mail (/bin/mail knows how...)
/tmp/mail*	temp file
/tmp/xscript*	saved transcript
/dev/log	to log status (optional)

SEE ALSO

mail(1), aliases(7N), netmailer(8N).

BUGS

Delivermail sends one copy of the letter to each user; it should send one copy of the letter to each host and distribute to multiple users there whenever possible.

Delivermail assumes the addresses can be represented as one word. This is incorrect according to the ARPANET mail protocol RFC 733 (NIC 41952), but is consistent with the real world.

NAME

netmail – the bnet network mail system

DESCRIPTION

The bnet network mail system consists of the following programs:

/etc/bnetmaild
a simple mail daemon run by crontab or by hand. Looks in the *mail spool directory* (**/usr/spool/netmail**) for files to send out onto the network. Uses *remsh* to send mail to remote hosts. Deletes mail if the mail is apparently successfully sent. Deletes mail found lying around which is more than one week old; apparently the destination host in this case is off the net. Accepts no arguments.

/etc/delivermail
exec'd by **/bin/mail** to deliver mail to users or networks depending on the contents of the address of the mail. If the address contains an

@ then deliver to the bnet network, else if the address has a

! then deliver to the uucp network,

else deliver locally.

/etc/netmailer
exec'd by **/etc/delivermail** to "deliver" netmail. Mail is actually deposited in **/usr/spool/netmail** with appropriate network mail headers prepended. **/etc/bnetmaild** actually sends the mail to the network. See *netmailer*(8N) for a description of flag arguments.

/bin/mail
has been modified to *exec* **/etc/delivermail** (*delivermail*(8N)) which does aliasing and re-routing of mail destined for the b-network.

FILES

/usr/spool/netmail
directory for network mail

/usr/spool/netmail/bnetXXXXXX
actual mail file(s), XXXXXX = pid.

SEE ALSO

mail(1), remsh(1N), delivermail(8N), netmailer(8N).

BUGS

Many, no doubt; for example, lots of work should be done on **/etc/bnetmaild**, i.e., if a piece of mail is deleted due to its being old or otherwise undeliverable, notification should be sent to the originator. Soon, however, *bnetmaild* will be replaced by *sendmail*.

A front-end is needed for mail, such as *Mail* (ucb).

NAME

netmailer – deliver mail to BNET

SYNOPSIS

/etc/netmailer *from-address to-host to-user*

DESCRIPTION

Netmailer queues the letter found on its standard input for delivery to the host and user specified. The actual delivery will be performed by the BNET mailer daemon (**/etc/bnetmaild**).

If the letter does not appear to have a full BNET header, *netmailer* will insert "Date:" and "From:" fields in the proper format. The "From:" person is determined by the *from-address* argument, with colons translated to periods and "@<local-host>" appended. The "<local-host>" is obtained from the file **/usr/lib/uucp/SYSTEMNAME**.

FILES

/usr/spool/netmail/*

SEE ALSO

delivermail(8N), netmail(8N).