Enhanced ONYX System V

ADMINISTRATOR REFERENCE MANUAL

Product Number: 805-02651-001

First Edition

INTRODUCTION

This manual is intended to supplement the information contained in the **Enhanced Onyx System V User Reference Manual** and to provide an easy reference volume for those who must administer a UNIX 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 **Enhanced Onyx System V Programmer Reference** or **User Reference** Manuals.

Section 1: System Maintenance Commands and Application Programs

This section contains system maintenance programs such as **fsck**, **mkfs**, etc., which generally reside in the directory /etc; these entries carry a sub-class designation of "1M" for cross-referencing reasons.

Section 7: Special Files

This section 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: Special Maintenance Procedures

This section describes crash recovery and boot procedures, facility descriptions, etc.

Each section consists of a number of independent entries of a page or so in length. The name of the entry appears in the upper corners of its page. Entries within each section are alphabetized, with the exception of an introductory entry that begins

each section. 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 in the following manner.

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

The **SYNOPSIS** summerizes 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 the appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual. (They are underlined in the typed version of the entries.)

> 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(S)** part gives example(s) or 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 deficiences. Occasionally, the suggested fix is also described.

A table of contents precedes Section 1. On most systems, all entries are available on-line via the **man(1)** command (see Section 1 of the **Enhanced Onyx System V User Reference Manual**).

TABLE OF CONTENTS

8. System Maintenance Procedures

PERMUTED INDEX

Permuted Index

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 UNIX System 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).
UNIX System User's Manual.
UNIX System 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 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 is a suggestion for use in shutdown procedures:

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

FILES

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

SEE ALSO

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

NAME

acctcon1, acctcon2 - connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

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

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

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

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

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

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

EXAMPLES

These commands are typically used as shown below. The file **otmp** 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

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.

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

acctmerg - merge or add total accounting files

SYNOPSIS

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

DESCRIPTION

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

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

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

```
acctmerg -v <file1 >file2
```

Perform edit on file2, then enter:

```
acctmerg -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.

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, remove, 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/remove

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

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

/usr/lib/acct/startup

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

DESCRIPTION

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

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

Dodisk should be invoked by cron to perform the disk accounting functions.

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.

Remove is invoked to remove the previous day's accounting files. It is located in **/lib** and called by startup when the system is brought up each day. This should be invoked only by **/usr/lib/acct/startup** and never by a user or an administrator.

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/pacctincr (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

adduser - add a user to the system

SYNOPSIS

/etc/adduser

DESCRIPTION

Adduser performs all the necessary tasks to add a user account to the system such as setting the login name, account's uid and gid, password, login shell and the account's home directory, automatically. Adduser prompts you for each parameter to be entered; however, adduser requires that you be either looged into the "root" account or become a super-user.

FILES

/etc/adduser /etc/passwd /etc/profile*

NAME

bcopy - interactive block copy

SYNOPSIS

/etc/bcopy

DESCRIPTION

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

The following questions are asked:

to: name the file or device to be copied to.

offset: provide the starting ``to'' block number.

from: name the file or device to be copied from.

offset: provide the starting ``from'' block number.

count: reply with the number of blocks to be copied.

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

Two consecutive carriage returns terminates 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)).

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

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

The powerfail procedure is invoked when the system detects a power failure condition. 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

checkall - faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

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

```
# check the root file system by itself
fsck /dev/dsk/c0d0s1

# dual fsck of drives 0 and 1
dfsck /dev/dsk/c0d0s3 - /dev/dsk/c0d1s1
```

In the above example (where **/dev/dsk/c0d1s1** is 34K blocks and **/dev/dsk/c0d0s3** is 25K blocks), a previous sequential fsck took 3 minutes. The checkall procedure also takes 3 minutes.

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

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

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

```
dfsck file_systems_on_drive_0 - file_systems_on_drive_1
dfsck file_systems_on_drive_2 - file_systems_on_drive_3
  . . .
```

WARNINGS

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

2. On a check that requires a scratch file (the **-t** option), be careful not to use the same temporary file for the two groups (this is sure to scramble the file systems).

3. The dfsck procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).
Setting up the UNIX System in the UNIX System Administrator's Guide.

NAME

chroot - change root directory for a command

SYNOPSIS

/bin/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.

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

/bin/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.

SEE ALSO

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

BUGS

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

NAME

config.68 - configure

SYNOPSIS

/etc/config.68 [-t] [-v file] [-l file] [-c file] [-m file] dfile

DESCRIPTION

Config.68 is a program that takes a description of and generates two files. One file provides information regarding the interface between the hardware and device handlers. The other file is a C program defining the configuration tables for the various devices on the system.

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

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

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

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

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

The user must supply dfile; it must contain device information for the user's system. This file is divided into three parts. The first part contains physical device specifications. The second part contains system-dependent information. The third part contains microprocessor-specific information. The first two parts are required, the third part is optional. Any line with an asterisk (*) in column 1 is a comment.

First Part of dfile

Each line contains four or five fields, delimited by blanks and/or tabs in the following format:

devname vector address bus number

where devname is the name of the device (as it appears in the **/etc/master** device table), vector is the interrupt vector location (hexadecimal), address is the device address (hexadecimal), bus is the bus request level (1 through 7),

and *number* is the number (decimal) of devices associated with the corresponding controller; *number* is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers which may be provided with the system that are actually pseudo-device drivers, that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt *vector*, device *address*, and *bus* request level must all be zero.

Second Part of *dfile*

The second part contains three different types of lines. Note that all specifications of this part are required, although their order is arbitrary.

1. *Root/pipe/dump device specification*

   Three lines of three fields each:

   root *devname* *minor* [,*minor*...]
   pipe *devname* *minor* [,*minor*...]
   dump *devname* *minor* [,*minor*...]

   where *minor* is the minor device number (in octal). For certain Motorola Inc. disk controllers, e.g., the Universal Disk, it is possible, and often desirable, to have a single UNIX System capable of executing on any device on the controller. For such devices, *minor* can be repeated (separated by commas). The first reference to *minor* specifies the **root** (**pipe**, **dump**) to be used for disk 0, the second *minor* for disk 1, etc. The same number of *minor* references must be present for **root**, **pipe**, **dump**, and **swap**.

2. *Swap device specification*

   One line that contains five fields as follows:

   swap *devname* *minor* *swplo* *nswap* [,*minor* *swplo* *nswap*...]

   where *swplo* is the lowest disk block (decimal) in the swap area and *nswap* is the number of disk blocks (decimal) in the swap area. Similar to the **root**, **pipe**, and **dump** specifications above, the **minor**, **swplo**, and **nswap** references can be repeated for certain Motorola Inc. controllers.

3. *Parameter specification*

Several lines of two fields each as follows (number is decimal):

buffers	number
inodes	number
files	number
mounts	number
coremap	number
swapmap	number
calls	number
procs	number
maxproc	number
texts	number
clists	number
hashbuf	number
physbuf	number
power	0 or 1
mesg	0 or 1
sema	0 or 1
shmem	0 or 1

Third Part of dfile

The third part contains lines identified by a keyword. The format of each line differs for each keyword. The ordering of the third part is significant.

1. Microprocessor specification

   One line of two fields:

   mpu number

   where number is 68000, 68010, or 68020. The default mpu number is 68000.

2. Non-unique driver specifications

   Several lines of two fields:

   force identifier

   where identifier is the name of a unique identifier defined within a driver, located in the kernel I/O library file. This forces the correct linking of non-table driven drivers, such as those for the clock, console, and mmu.

3. Memory probe specifications

   Several lines of three fields:

   probe address value

where address is the hexadecimal number specifying a memory-mapped I/O location, which must be reset for to execute properly. The intent is to provide a means by which non-standard (or unsupported) devices can be set to a harmless state. Value is a hexadecimal number (0X00-0XFF) to be written in address, or -1, indicating that the address is to be "read only".

4. Alien handler entry specifications

Several lines of three fields:

alien <vector address> <alien address>

where vector address is the hexadecimal address of the normal exception vector for the alien entry point, and alien address is the hexadecimal entry point for the non-UNIX handler. If no UNIX handler is associated with the vector address, then alien address is entered into the vector. Otherwise, code is produced in low.s so that the alien handler is entered only when the exception occurs in the processor's supervisor state.

5. Multiple handler specifications

Several lines of four or five fields:

dup flag <vector address> handler [argument]

where flag is a bit mask. The bits are interpreted as:

1 - if handler returns 0, go to the normal interrupt return point ("intret").

2 - if handler returns 0, go to the normal trap return point ("alltraps").

4 - if handler returns 0, go to the quick return point ("return").

10 - argument is to be passed to handler.

Vector address is the hexadecimal address of the normal exception vector for the firmware entry point. Handler is the name of an exception handling routine, with the optional argument passed to it. The intent is to provide a means of specifying multiple handlers for a single exception. These handlers are called in the order given in dfile; then, the normal handler is called. If bits 1, 2 or 4 of flag are set and the handler returns zero, then the remainder of the handlers are not called.

6. Memory configuration specifications

Several lines of four of five fields:

ram flag low high [size]

where flag is an octal bit mask, which is interpreted as follows:

1 - memory has no parity check and, therfore, need not be initialized after power up.

2 - a single memory block may exist, ranging from low through high -1.

4 - multiple memory blocks may be located in the range and are of size bytes.

Low and high are hexadecimal memory addresses, and size is a hexadecimal number. The intent is to provide information to about noncontiguous memory. Low specifies the low memory address where memory may be located, and which may extend through high-1. If the range consists of multiple boards, which may or may not be present, they are of size bytes.

For flag 2 ranges, UNIX writes sequential memory locations, starting at low, until a memory fault occurs. For flag 4 ranges, UNIX performs a test for each size-sized subrange. If memory need not be initialized, only the first byte of the range (flag 2) or subrange (flag 4) is tested to determine the presence of the memory.

It is essential that ram lines be ordered in ascending low addresses. If two lows are equal (more than one size memory block may be located within a range), they must be ordered in ascending sizes.

If no **ram** specifier is present, the default is:

ram 2 0 F00000

EXAMPLE

The actual system configuration table would be specified as follows (comments may be inserted by preceding the comment with an asterisk (*)):

```
* System Devices
root    disk    10
swap    disk    17    1    3292
pipe    disk    10
```

```
dump    disk    17
* Tunable Parameters
buffers     50
calls       50
inodes      90
files       90
mounts       8
procs       50
texts       40
clists     150
sabufs       0
maxproc     15
coremap    100
swapmap     75
hashbuf     64
physbuf      4
power        0
mesg         1
sema         1
shmem        1
* See <sys/io.h>
```

FILES

/etc/master default input master device table

m68kvec.s
default output exception vector file for m68k

low.s
default output hardware interface file for m68k

conf.c
default output configuration table file

SEE ALSO

sysdef(1M), master(4).
``Setting up '' in the Administrator's Guide.

DIAGNOSTICS

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

NAME

crash - examine system images

SYNOPSIS

/etc/crash [system] [namelist]

DESCRIPTION

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

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

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

COMMANDS

Input to crash is typically of the form:

command [options] [structures to be printed]

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

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

The current list of commands includes:

user [list of process table entries]
Aliases: **uarea**, **u_area**, **u**.
Print the user structure of the named process as determined by the information contained in the process table entry. If no entry number is given, the information of the last executing process is printed. Swapped processes produce an error message.

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

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

kfp [stack frame pointer]
Aliases: **fp**.
Print the program's idea of the start of the current stack frame (set initially from a fixed location in the dump) if no argument is given, or set the frame pointer to the supplied value.

stack [list of process table entries]
Aliases: **stk**, **s**, **kernel**, **k**.
Format a dump of the kernel stack of a process. The addresses shown are virtual system data addresses rather than true physical locations. If no entry number is given, the information on the last executing process is printed.

proc [-[r]] [list of process table entries]
Aliases: **ps**, **p**.
Format the process table. The **-r** option causes only runnable processes to be printed. The **-** alone generates a longer listing.

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

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

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

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

tty [type] [-] [list of tty entries]
Aliases: **term**, **acia**.
Print the tty structures. The type argument determines which structure is used (such as **acia**; the last type is remembered). The **-** option prints the stty(1)

parameters for the given line.

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

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

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

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

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

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

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

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

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

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

! Escape to shell.

q Exit from crash.

? Print synopsis of commands.

ALIASES

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

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

FILES

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

SEE ALSO

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

BUGS

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

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

NAME

cron - clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times 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.

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:

-sX 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).

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

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

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

-ffsize[:isize]
specify the outputfs file system and inode list sizes (in blocks). If not given, the values from the inputfs are used.

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

SEE ALSO

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

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 bcheckrc(1M)) to construct a mount table entry for the **root** device.

EXAMPLE

The command:

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

produces

```
dsk/c0d0s2 /usr
```

if **/usr** is mounted on **/dev/dsk/c0d0s2.**

FILES

/dev/dsk*
/etc/mnttab

SEE ALSO

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

NAME

df - report number of free disk blocks

SYNOPSIS

df [-t] [-f] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free 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/dsk/c0d0s2**) 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

errdead - extract error records from dump

SYNOPSIS

/etc/errdead dumpfile [namelist]

DESCRIPTION

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

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

FILES

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

DIAGNOSTICS

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

SEE ALSO

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

NAME

errdemon - error-logging daemon

SYNOPSIS

/usr/lib/errdemon [file]

DESCRIPTION

The error logging daemon errdemon collects error records from the operating system by reading the special file **/dev/error** and places them in file. If file is not specified when the daemon is activated, **/usr/adm/errfile** is used. Note that file is created if it does not exist; otherwise, error records are appended to it, so that no previous error data is lost. No analysis of the error records is done by errdemon; that responsibility is left to errpt(1M). The error-logging daemon is terminated by 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. For the EXORmacs, the device for which errors are logged is ud(7). For

3B20S, the devices are DFC, IOP, and MT. For Digital Equipment Corporation machines, the (block) devices for which errors are logged are RP03, RP04, RP05, RP06, RP07, RS03, RS04, TS11, TU10, TU16, TU78, RK05, RK06, RK07, RM05, RM80, and RF11. Additional identifiers are **int** and **mem** which include detailed reports of stray-interrupt and memory-parity type errors respectively.

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

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

FILES

/usr/adm/errfile default error file

SEE ALSO

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 superuser may use errstop.

FILES

/unix default system namelist

DIAGNOSTICS

The diagnostics produced by errstop are self-explanatory.

SEE ALSO

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

NAME

ff - list file names and statistics for a file system

SYNOPSIS

/bin/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.

EXAMPLES

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

```
ff -I /dev/dsk/c0d0s1
```

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

```
ff -m -1 /dev/dsk/c0d0s2 > /log/incbackup/usr/tuesday
```

To obtain the path names for i-nodes 451 and 76 on a specified file system:

```
ff -i 451,76 /dev/dsk/c0d0s1
```

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-tape backup and tapesave is for weekly disk-to-tape.

FILES

/etc/filesave
/etc/tapesave

SEE ALSO

cpio(1).

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.

EXAMPLES

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

```
finc -m -2 /dev/rdsk/c0d0s2 /dev/rmt_rwd
```

SEE ALSO

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

NAME

frec - recover files from a backup tape

SYNOPSIS

/bin/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.

EXAMPLES

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

```
frec /dev/rmt_rwd 1216:junk
```

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

```
frec -p /usr/src/cmd /dev/rmt0 14156:a 1232:b
3141:/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

fsba - file system block analyzer

SYNOPSIS

fsba file system ...

DESCRIPTION

Fsba determines the number of extra sectors (1 sector has 512 bytes) needed when the file system logical block size is increased from 512 bytes per block to 1024 bytes/block. File system should be specified by device name (e.g., /dev/dk11).

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

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

SEE ALSO

fs(4).

NAME

fsck, dfsck - file system consistency check and interactive repair

SYNOPSIS

/etc/fsck [-y] [-n] [-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.

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

-sBlocks-per-cylinder:Blocks-to-skip (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.

-sX Conditionally reconstruct the free list. This option is like -sX 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).

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.

FILES

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

SEE ALSO

checkall(1M), clri(1M), ncheck(1M), checklist(4), fs(4), crash(8).
Setting up the UNIX System in the UNIX System Administrator's Guide.

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.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

-v Convert file system from M68000 to VAX format.

-m Convert file system from VAX to M68000 format.

Ispecial is the name of a special file containing a file system to be converted (e.g., **/dev/rdk10**). The optional ospecial is the name of the special file to receive the results of the conversion. If ospecial is specified, the entire contents of ispecial are copied to ospecial before the conversion is performed. If ospecial is not specified, an in-place conversion of ispecial is performed. The following items should be noted before executing fscv:

1. A file system consistency check (fsck(1M)) should be performed on ispecial immediately prior to executing fscv.

2. Neither ispecial nor the optional ospecial should contain a mounted file system during execution of fscv. Modification to either the input or the output file system while fscv is executing will probably corrupt the converted file system.

3. A backup of ispecial (see volcopy(1M)) is highly recommended if an in-place conversion is to be performed. System crashes, I/O errors, etc., during execution of fscv may destroy the file system contained in ispecial. Also, if the optional ospecial is specified, any data contained in that special file is over written.

4. If the optional ospecial is specified, this special file must be large enough to contain the entire contents of ispecial. See the appropriate special files in section 4.

EXAMPLES

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

```
/etc/fscv -v /dev/rdk00 /dev/rdk10
```

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

```
/etc/fscv -m /dev/rdk10
```

BUGS

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

SEE ALSO

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

NAME

fsdb, fsdb1b - file system debugger

SYNOPSIS

/etc/fsdb special [-]
/etc/fsdb1b special [-]

DESCRIPTION

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

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

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

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

The symbols recognized by fsdb are:

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

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

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

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

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

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

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

min minor device number

Fsdb1b

Fsdb1b is the same as fsdb, except fsdb1b is used for file systems with a 512 byte block size.

EXAMPLES

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

SEE ALSO

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

NAME

fwtmp, wtmpfix - manipulate connect accounting records

SYNOPSIS

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

DESCRIPTION

Fwtmp

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

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

Wtmpfix

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

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

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

FILES

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

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

Getty is a program that is invoked by init(1M). It is the second process in the series, (init-getty-login-shell) that ultimately connects a user with 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 inital 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 searches a set of default directories (/bin, /usr/bin, /etc, /lib, and /usr/lib, in that order) for a file with the same name as file. When the first occurrence is found, install issues a message saying that it is overwriting that file with file, and proceeds to do so. If the file is not found, the program states this and exits without further action.

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

The meanings of the options are:

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

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

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

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

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

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

SEE ALSO

make(1), mk(8).

NAME

killall - kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is 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

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.

SEE ALSO

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

NAME

lpadmin - configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin -pprinter [options]
/usr/lib/lpadmin -xdest
/usr/lib/lpadmin -d[dest]

DESCRIPTION

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

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

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

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

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

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

-cclass inserts printer P into the specified class. Class is created if it does not already exist.

-eprinter 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.

-iinterface establishes a new interface program for P. Interface is the pathname of the new program.

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

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

-rclass removes printer P from the specified class. If P is the last member of the class, then the class is removed.

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

Restrictions.

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

Models.

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

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

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

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

nroff 450 driving table.

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

-c compressed print
-e expanded print

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

EXAMPLES

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

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

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

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

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

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

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

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

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

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

SYNOPSIS

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

DESCRIPTION

Lpsched schedules requests taken by lp(1) for printing online printers.

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

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

mkfs - construct a file system

SYNOPSIS

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

DESCRIPTION

Mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line. If the second argument is given as a string of digits, mkfs builds a file system with a single empty directory on it. The size of the file system is the value of blocks interpreted as a decimal number. This is the number of physical disk blocks the file system 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 new-lines. The first token is the name of a file to be copied onto block zero as the bootstrap program (see unixboot(8)). The second token is a number specifying the size of the created file system in physical disk blocks. 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 blocks/cyl can be specified.

SEE ALSO

dir(4), fs(4), unixboot(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.

NAME

mknod - build special file

SYNOPSIS

/bin/mknod name c | b major minor
/bin/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).

SEE ALSO

mknod(2).

NAME

mount, umount - mount and dismount file system

SYNOPSIS

/bin/mount [special directory [-r]]

/bin/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

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.

SEE ALSO

mkdir(1).

NAME

ncheck - generate names from i-numbers

SYNOPSIS

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

DESCRIPTION

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.

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 group of programs to facilitate an activity study of the operating system.

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

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

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

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

FILES

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

SEE ALSO

prf(7).

NAME

pwck, grpck - password/group file checkers

SYNOPSIS

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

DESCRIPTION

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

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

FILES

/etc/group
/etc/passwd

SEE ALSO

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

DIAGNOSTICS

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

NAME

runacct - run daily accounting

SYNOPSIS

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

DESCRIPTION

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

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

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

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of **chargefee** into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.

MERGETACCT Merge the daily total accounting records in daytacct with the summary total accounting records in **/usr/adm/acct/sum/tacct**.

CMS Produce command summaries.

USEREXIT Include any installation-dependent accounting programs here.

CLEANUP Cleanup temporary files and exit.

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

EXAMPLES

To start runacct.
```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart runacct.
```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart runacct at a specific state.
```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &
```

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), acct(2), acct(4), utmp(4).
``Accounting'' in the Administrator's Guide.

DIAGNOSTICS

The accounting system starts complaining with *****RECOMPILE pnpsplit WITH NEW HOLIDAYS***** after the last holiday of the year. See ``Accounting'' in the Administrator's Guide for

more on how to correct this condition. Other diagnostics are placed in various error and log files.

BUGS

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

runacct mmdd **WTMPFIX**

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

NAME

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 sys -c "/usr/lib/sa/sadc /usr/adm/sa/sadate +%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 sys -c "/usr/lib/sa/sa1"
0 8-17 * * 1-5 su sys -c "/usr/lib/sa/sa1 1200 3"
0 18-7 * * 1-5 su sys -c "/usr/lib/sa/sa1"
```

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

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

```
5 18 * * 1-5 su adm -c "/usr/lib/sa/sa2 -s 8:00 -e 18:01  -i
3600 -A"
```

reports important activities hourly during the working day.

The structure of the binary daily data file is:

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

FILES

/usr/adm/sa/sadd	daily data file
/usr/adm/sa/sardd	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

sag(1G), sar(1), timex(1), "System Activity Package" in Admininstrator's Guide.

NAME

setmnt - establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

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

filesys node

where filesys is the name of the file system's special file (e.g., ``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.

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

sysdef - system definition

SYNOPSIS

/etc/sysdef [opsys [master]]

DESCRIPTION

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

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

FILES

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

SEE ALSO

config(1M), master(4).

BUGS

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

NAME

uuclean - uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [options]

DESCRIPTION

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

The following options are available.

-ddirectory Cleans directory instead of the spool directory.

-ppre Scans for files with pre as the file prefix. Up to 10 -p arguments may be specified. A -p without any pre following causes all files older than the specified time to be deleted.

-ntime Deletes files whose age is more than time hours, if the prefix test is satisfied. (default time is 72 hours)

-wfile Finds files which are older than time hours; however, the files are not deleted. If the argument file is present, the warning is placed in file; otherwise, the warnings go to the standard output.

-ssys Examines only files destined for system sys. Up to 10 -s arguments may be specified.

-mfile Sends mail to the owner of the file when it is deleted. If a file is specified, then an entry is placed in file.

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

FILES

/usr/lib/uucp (directory with commands used by uuclean internally)

/usr/spool/uucp (spool directory)

SEE ALSO

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

NAME

uusub - monitor uucp network

SYNOPSIS

/usr/lib/uucp/uusub [options]

DESCRIPTION

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

-asys Add sys to the subnetwork.
-dsys Delete sys from the subnetwork.
-l Report the statistics on connections.
-r Report the statistics on traffic amount.
-f Flush the connection statistics.
-uhr Gather the traffic statistics over the past hr hours.
-csys 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 -uhr option. Similarly, rfile and rbyte are the numbers of files and bytes received.

The command:

uusub -c all -u 24

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

FILES

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

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

NAME

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),
- **-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**, **usr**, etc.) of the filsystem being copied.

The special should be the physical disk section or tape (e.g.: **/dev/rdsk/c0d0s1**, **/dev/rmt_rwd**, etc.).

The volname is the physical volume name (e.g.: **s1**, **s3**, 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).

EXAMPLES

These examples show how to use volcopy for backing up onto cartridge tape. Use 1000 feet for the size of reel with XL (450 feet/track) cartridge tapes. Use 600 feet for the size of reel with standard (300 feet/track) cartridge tapes. First use labelit to label as many cartridge tapes as you feel will be necessary for backing up (for example, three standard tapes are required to backup a full /usr filesystem)

```
labelit /dev/rmt_rwd usr s3 -n
```

Now execute the volcopy command:

```
volcopy usr /dev/rdsk/c0d0s3 s3 /dev/rmt_rwd -
```

The system will respond with:

```
Enter size of reel in feet for <->:     1000

Tape density (ie, 800 ¦ 1600 ¦ 6250)?   1600

Reel 0, 1000 feet, 1600 BPI
You will need 3 reels.
(    The same size and density is expected for all reels)
From: /dev/rdsk/c0d0s3, to /dev/rmt_rwd?(DEL if wrong)

Writing REEL 1 of 3, VOL = -
```

FILES

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

SEE ALSO

fs(4).

BUGS

Tape record sizes are determined both by density and by drive type.

Both volcopy and labelit must be used with rewind devices; non-rewind devices won't work.

The density 6250 for cartridge tape won't work.

The -n option to labelit puts the wrong date and a minus number for the number of inodes. Volcopy, however, corrects these after it has written the header at the beginning of the cartridge tape.

Labelit reports twice as many blocks as there actually are. All of the UNIX utilities, with the exception of mkfs, report blocks in 512-byte blocks. This is because AT&T does it this way.

NAME

wall - write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

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

Broadcast Message from ...

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

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

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

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

NAME

whodo - who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

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

SEE ALSO

ps(1), who(1).

NAME

intro - introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and 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

cat - phototypesetter interface

DESCRIPTION

Cat provides the interface to a Wang Laboratories, Inc. C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding will not be described here.

Only one process may have this file open at a time. It is write-only.

FILES

/dev/cat

SEE ALSO

troff(1).

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

mem, kmem - core memory

DESCRIPTION

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

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

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

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

FILES

/dev/mem, /dev/kmem

BUGS

Mem does not access addresses outside of physical ram memory; hence, no device registers are available.

NAME

null - the null file

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf - operating system profiler

DESCRIPTION

The file **prf** provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **prf** is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

config(1M), profiler(1M).

NAME

termio - general terminal interface

DESCRIPTION

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:

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

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

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

FILES

/dev/tty*

SEE ALSO

stty(1), ioctl(2).

NAME

trace - event-tracing driver

DESCRIPTION

Trace is a special file that allows event records generated within the UNIX kernel to be passed to a user program so that the activity of a driver or other system routines can be monitored for debugging purposes.

An event record is generated from within a kernel driver or system routine by invoking the trsave function:

```
trsave (dev, chno, buf, cnt)
char dev, chno, *buf, cnt;
```

Dev is a minor device number of the trace driver; chno is an integer between 0 and 15 inclusive that identifies the data stream (channel) to which the record belongs; buf is a buffer containing the data for an event; and cnt is the number of bytes in buf. Calls to trsave will result in data being placed on a queue, provided that some user program has opened the trace minor device dev and has enabled channel chno. Event records consisting of a time stamp (4 bytes), the channel number (1 byte), the count (1 byte), and the event data (cnt bytes) are stored on a queue until a system-defined maximum (TRQMAX) is reached; an event record is discarded if there is not sufficient room on the queue for the entire record. The queue is emptied by a user program reading the trace driver. Each read returns an integral number of event records; the read count must, therefore, be at least equal to cnt plus six.

The trace driver supports open, close, read, and ioctl system calls. The ioctl system call is invoked as follows:

```
#include <sys/vpm.h>
int fildes, cmd,  arg;
ioctl (fildes, cmd, arg);
```

The values for the cmd argument are:

VPMSETC-Enable trace channels. This command enables each channel indicated by a 1 in the bit mask found in arg. The low-order bit (bit 0) corresponds to channel zero, the next bit (bit 1) corresponds to channel 1, etc.

VPMGETC-Get enabled channels. This command returns in arg a bit mask containing a 1 for each channel that is currently enabled.

VPMCLRC-Disable channels. This command disables the channels indicated by a 1 in the bit mask found in arg.

SEE ALSO
vpmsave(1M), vpm(7).

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*

SEE ALSO

dz(7), tn4(7), tn74(7).

NAME

intro - introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions on such topics as boot procedures, recovery from crashes, file backups, etc.

BUGS

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

NOT SUPPORTED AT THIS TIME

NAME

crash.m68 - what to do when the system crashes

DESCRIPTION

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

In restarting after a crash, always bring up the system single-user, as specified in bo(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 the at all, three files (and the directories leading to them) must be intact. First, the initialization program **/etc/init** must be present and executable. For init to work correctly, **/dev/console** and **/bin/sh** must be present. If either does not exist, the symptom is best described as thrashing. Init goes into a fork/exec loop trying to create a shell with proper standard input and output.

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

Repairing disks. The first rule to keep in mind is that a disk in need of repair should be treated gently; it shouldn't be mounted unless necessary, and if it is very valuable yet in quite bad shape, perhaps it should be copied before trying surgery on it.

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

Fsck also reports on incorrect link counts and requests permission to adjust any that are erroneous. In addition, it reconnects any files or directories that are allocated but have no file system references to a ``lost+found'' directory. Finally, if the free list is bad (out of range, missing, or duplicate blocks) fsck constructs a new one, with the operator's concurrence.

Why did it crash?
types a message on the console when a crash occurs. Here is the current list of such messages, with enough information to provide a possible remedy. The message has the form ``panic: ...'', usually accompanied by other information. Left unstated in all cases is the possibility that hardware or software error produced the message in some unexpected way.

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

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

iinit
An I/O error reading the superblock for the root file system during initialization.

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

no imt
Similar to ``no fs'', but produced elsewhere.

no clock

During initialization, neither the line nor programmable clock was found to exist.

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

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

trap
An unexpected trap has occurred within the system. This is accompanied by three numbers: a ``ps'', which is the user's stack pointer; ``pc'', which is the user's program counter; and a ``trap type'' that encodes which trap occurred. The trap types are:

2	bus error
3	address error
4	illegal instruction
5	zero divide fault
6	CHK instruction fault
7	TRAPV instruction fault
8	privileged instruction fault
9	trace trap
10	line 1010 emulator
11	line 1111 emulator
24	spurious interrupt
32	TRAP 0 - system call
33	TRAP 1 - breakpoint
34	TRAP 2 - simulate DEC IOT instruction
35	TRAP 3 - simulate DEC EMT instruction
36	TRAP 4 - floating point exception

In some of these cases it is possible for hexadecimal 200 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, dump the system.

Interpreting dumps. All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file **/usr/tmp/core**; cp(1) can be used. At this point, you should execute **ps -el -c /usr/tmp/core** and **who** to print the process table and a list of the users who were on at the time of the crash.

SEE ALSO

fsck(1M), bo(8), m68kops(8).

NAME

diskconf - configures a hard disk for filesystem use

SYNOPSIS

diskconf -c ctrl# -d drive# [-s s1 [s2...s8]] [-b] [-z]

DESCRIPTION

diskconf sub-divides disks into logical sections, configures disks for booting, and/or zeros disks.

Parameters:

-c ctrl# Specifies the controller address on the SCSI bus to be accessed.

-d drive# Identifies the disk on the specified controller to be configured.

-s s1 [s2...s8] Specifies the logical sub-divisions of the disk. One or more zero values can be used to allocate remaining sectors.

For example:

-s 4000 0 5000

configures a disk of 15000 sectors into three sub-divisions. The first section contains 4000 sectors, the second 6000, and the third 5000 sectors.

-s 4000 0 5000 0

configures the same disk of 15000 sectors into four sub-divisions. The first contains 4000 sectors, the second 3000, the third 5000, and the fourth 3000 sectors.

-b Configures the disk for booting.

-z Zeros the disk.

Note: This option destroys all data on the disk.

diskconf may only be invoked under the stand-alone environment or by the super-user under UNIX.

CAVEATS AND BUGS

The **-z** option has not been implemented.

NAME

format - format disks in preparation for filesystem building

SYNOPSIS

format -c ctrl# -d drive# -s size -t ctrl_type [-l label]

DESCRIPTION

format hard formats disks connected to SCSI compatible controllers. When invoked, commands are sent to the appropriate controller to initiate formatting on the requested drive.

Parameters:

-c ctrl# Specifies the controller address on the SCSI bus to be accessed.

-d drive# Identifies the disk on the specified controller to be formatted.

-s size Identifies the drive type and size (model number is used).

size must be one of:
st506 (6 MB)
5006h (6 MB)
5012h (12 MB)
5018h (18 MB)

-t ctrl_type Specifies the type of controller at the address given by -c ctrl#.

ctrl_type must be one of:
adaptec
adaptec55

-l label An optional 8 character label to be recorded in sector 0 of the hard disk.

Due to the length of time required by the controller to format the disk, format will take awhile to complete.

Note: This program will destroy all data on a disk, so use it with care. None of the data on the disk is saved and format is not limited by the logical sub-divisions of the disk (see diskconf(1M/1SA)). format may only be invoked under the stand-alone environment or by the super-user under UNIX.

DIAGNOSTICS

If an unsupported size or type of controller is specified the supported types and/or sizes will be displayed. A controller number and type, drive number, and disk size must be

specified.

SEE ALSO

diskconf(8)

CAVEATS AND BUGS

If **format** is invoked with an incorrect controller type (ie the controller specified is different than the controller found) the results will be incorrect. The data will be destroyed and there is the possibility of receiving an error from the SCSI strategy routine (stating that the mode select command is incorrect).

Remember, this command will destroy all data on the specified disk, so use it with care!

NOT SUPPORTED AT THIS TIME

NAME

mk - how to remake the system and commands

DESCRIPTION

All source for
is in a source tree distributed in the directory **/usr/src**. This includes source for the operating system, libraries, commands, miscellaneous files necessary to the running system, and procedures to create everything from this source.

The top level consists of the directories **cmd**, **lib**, **uts**, **head**, and **stand** as well as commands to remake each of these directories. These commands are named :mk, which remakes everything, and :mk**dir** where **dir** is the directory to be recreated. Each recreation command makes all or part of the piece; over which it has control. :mk runs each of these commands and thus recreates the whole system.

The **lib** directory contains libraries used when loading user programs. The largest and most important of these is the C library. All libraries are in sub-directories and are created by a makefile or runcom. A runcom is a shell command procedure used specifically to remake a piece of the system. :mklib rebuilds the libraries that are given as arguments. The argument * causes it to remake all libraries.

The **head** directory contains the header files, usually found in **/usr/include** on the running system. :mkhead installs those header files that are given as arguments. The argument * causes it to install all header files.

The **uts** directory contains the source for the operating system. :mkuts (no arguments) invokes a series of makefiles that recreate the operating system.

The **stand** directory contains stand-alone commands and boot programs. :mkstand rebuilds and installs these programs.

The **cmd** directory contains files and directories. :mkcmd transforms source into a command based upon its suffix (**.l**, **.y**, **.c**, **.s**, **.sh**), or its makefile (see make(1)) or runcom. A directory is assumed to have a makefile or a runcom that takes care of creating everything associated with that directory and its sub-directories. Makefiles and runcoms are named command**.mk** and command**.rc** respectively.

:mkcmd recreates commands based upon a makefile or runcom if one of them exists; alternatively commands are recreated in a standard way based on the suffix of the source file. All commands requiring more than one file of source are grouped in sub-directories, and must have a makefile or a runcom. C programs (.c) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (.s) are assembled with **/usr/include/sys.s** which contains the system call definitions. Yacc programs (.y) and lex programs (.l) are processed by yacc(1) and lex(1) respectively before C compilation. Shell programs (.sh) are copied to create the command. Each of these operations leaves a command in **./cmd** which is then installed by using **/etc/install**.

The arguments to :mkcmd are either command names, or subsystem names. The subsystems distributed with
are: **acct**, **graf**, **sccs**, and **text**. Prefacing the :mkcmd instruction with an assignment to the shell variable **$ARGS** causes the indicated components of the subsystem to be rebuilt.

The entire **sccs** subsystem can be rebuilt by:

```
/usr/src/:mkcmd  sccs
```

while the delta component of **sccs** can be rebuilt by:

```
ARGS="delta" /usr/src/:mkcmd sccs
```

The **log** command, which is a part of the **stat** package, which is itself a part of the **graf** package, can be rebuilt by:

```
ARGS="stat log" /usr/src/:mkcmd  graf
```

The argument ***** causes all commands and subsystems to be rebuilt.

Makefiles, both in **./cmd** and in sub-directories, have a standard format. In particular :mkcmd depends on there being entries for install and clobber. Install should cause everything over which the makefile has jurisdiction to be made and installed by **/etc/install**. Clobber should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

Most of the runcoms in **./cmd** (as opposed to sub-directories) relate in particular to a need for separated instruction and data (I and D) space.

Ctime checks the environment (see environ(5)) for the time zone. This results in time zone conversions possible on a per-process basis. **/etc/profile** sets the initial environment for each user, and **/etc/rc** sets it for certain system daemons. These two programs are the only ones which must be modified outside of the eastern time zone.

An effort has been made to separate the creation of a command from source, and its installation on the running system. The command **/etc/install** is used by :mkcmd and most makefiles to install commands in the proper place on the running system. The use of install allows maximum flexibility in the administration of the system. Install makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in install. The object is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles. The minimization of makefiles in most cases, and the site independence of the others should greatly reduce the necessary maintenance, and allow makefiles to be considered part of the standard source.

SEE ALSO

install(1M), make(1).

NAME

spare - replace a bad sector with a good one

SYNOPSIS

spare -c? -d? -s? [<blocklist>]

DESCRIPTION

Spare replaces the specified blocks in <blocklist> with spare blocks. Spare only works on one slice at a time. Spare updates the information in absolute sector 0 and writes a copy to the alternate sector (129).

-c?, -d?, and -s? is the controller, unit, and slice number(s) respectively. These correspond to the numbers in the device name.

<blocklist> is a list of up to 16 slice relative blocks to be spared. If blocklist is not supplied, spare simply reads a valid copy of sector 0 and writes to both sector 0 and the alternate sector (129). This is a convenient way to recover from a damaged sector 0.

If there is not enough spare sectors, spare displays a message. If sector 0 is bad, spare reads the alternate sector (129) and displays a warning message automatically.

EXAMPLE

spare -c0 -d0 -s3 69

This example illustrates replacing bad block 69 in slice 3 of disk 0, controller 0, with a spare sector if one is available.

NAME

sparelist - list the spared sectors associated with a slice

SYNOPSIS

sparelist [device] in UNIX

- or -

sparelist [-c? -d? -s?] in STANDALONE

DESCRIPTION

Sparelist lists the base sector, the total number of sectors in that slice, the number of alternate (spared) sectors allocated, and the sectors that already have been spared. If no device is specified, sparelist lists information for all of the disks on the system.

Use the first form of sparelist if currently running under UNIX, and the second form if running in the standalone environment.

[device] is the device name (e.g., /dev/dsk/c0d0s3).

-c? -d? -s? is the controller, unit, and slice number which correspond to the numbers in the device name.

Sparelist requires the user to become a super-user to envoke it. If sector 0 is bad, spare reads the alternate sector (129) and displays a warning message automatically.

EXAMPLE

sparelist /dev/dsk/c0d0s3

- or -

sparelsit -c0 -d0 -s3

Both forms of the example above, displays the spare sector information for disk controller 0, unit 0, slice 3.