ICON/UXV Administrator Reference

ICON INTERNATIONAL

764 East Timpanogos Parkway
Orem, Utah 84057
(801) 225-6888

ADMINISTRATOR REFERENCE MANUAL

ICON/UXV Operating System

Order No. 172-036-002 A0 (Manual Assembly)
Order No. 171-063-002 A0 (Manual Pages only)

This manual was set on an IMAGEN 8/300 laser printer driven by the IROFF formatter operating under the ICON/UXV system.

Trademarks

Change Record Page

ICON/UXV Administrator Reference Manual

Manual Pages Part No. 171-063-002

Date	Revision	Description	Pages Affected
Apr. 1988	A0	Initial production release	All
Aug. 1988	A1	Add man pages to sections 1M and 7	TOC, badsect, dkfmt, doscopyd, doskisk, dosprint, dumpfs, fdump, ff, finc, frec, getty, restore, sadp, trenter, volcopy, intro (7), and ace (7)

INTRODUCTION

This manual is intended to supplement the information contained in the *ICON/UXV User Reference Manual* and to provide an easy reference volume for those who must administer a ICON/UXV 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 *ICON/UXV Programmer Reference Manual* or the *ICON/UXV User Reference Manual*.

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

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

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

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

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

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

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

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

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

A vertical bar ¦ between arguments indicates a selection argument, i.e. only one of the arguments separated by vertical bars is to be used.

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) of usage, where appropriate.

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

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

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

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

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

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

TABLE OF CONTENTS

1. System Maintenance Commands and Application Programs

7. Special Files

8. System Maintenance Procedures

PERMUTED INDEX

NAME

intro — introduction to system maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *ICON/UXV User Reference Manual* and Sections 2, 3, 4, and 5 of the *ICON/UXV Programmer Reference Manual.* References to other manual entries not of the form *name*(1M), *name*(7) or *name*(8) refer to entries of the above manuals.

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).
ICON/UXV User Reference Manual.
ICON/UXV Programmer Reference Manual.
ICON/UXV Administrator 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

badsect — create files to contain bad sectors

SYNOPSIS

/etc/badsect bbdir sector ...

DESCRIPTION

Badsect makes a file to contain a bad sector. Normally, bad sectors are made inaccessible by the standard formatter, which provides a forwarding table for bad sectors to the driver. If a driver supports the bad blocking standard it is much preferable to use that method to isolate bad blocks, since the bad block forwarding makes the pack appear perfect, and such packs can then be copied with *dd*(1). The technique used by this program is also less general than bad block forwarding, as *badsect* can't make amends for bad blocks in the i-list of file systems or in swap areas.

Badsect is used on a quiet file system in the following way: First mount the file system, and change to its root directory. Make a directory BAD there. Run *badsect* giving as argument the BAD directory followed by all the bad sectors you wish to add. (The sector numbers must be relative to the beginning of the file system, but this is not hard as the system reports relative sector numbers in its console error messages.) Then change back to the root directory, unmount the file system and run *fsck*(1M) on the file system. The bad sectors should show up in two files or in the bad sector files and the free list. Have *fsck* remove files containing the offending bad sectors, but **do not** have it remove the BAD/*nnnnn* files. This will leave the bad sectors in only the BAD files.

Badsect works by giving the specified sector numbers in a *mknod*(2) system call, creating an illegal file whose first block address is the block containing bad sector and whose name is the bad sector number. When it is discovered by *fsck* it will ask "HOLD BAD BLOCK"? A positive response will cause *fsck* to convert the inode to a regular file containing the bad block.

SEE ALSO

fsck(1M), dkfmt(1M)

DIAGNOSTICS

Badsect refuses to attach a block that resides in a critical area or is out of range of the file system. A warning is issued if the block is already in use.

BUGS

If more than one sector which comprise a file system fragment are bad, you should specify only one of them to *badsect,* as the blocks in the bad sector files actually cover all the sectors in a file system fragment.

NAME

binstl – program to install bootloader on disk

DESCRIPTION

Binstl is a program executable only from standalone mode. It is used to install the boot loader, *bload* (1M), in the beginning sectors of a disk (hard or floppy). This must be done before the disk can be used by the system.

For a detailed description on how to install the loader refer to the *Administrator Reference Manual*

FILES

/stand/binstl

SEE ALSO

bload(1M), standalone(8)

NAME

bload — program to load standalone programs

DESCRIPTION

Bload is a program executable only from standalone mode. It is used to load and run standalone programs.

Bload is actually the heart of standalone mode. Bload is run when the automatic reboot procedure is overridden. The user is given the option of which device to load from, and *bload* is loaded from that device and run. *Bload* issues its prompt:

```
ICON loader -- Version 1.0
Load:
```

From here the standalone programs are run, single user mode is entered, and multi-user mode is entered.

Arguments to load may be:

- [device]:[partition] (e.g. ct0:2 or is0:a)
- >standalone program (e.g. stand/fsck,stand/mkfs)
- -s (single user mode)

FILES

/stand/bload

SEE ALSO

standalone(8)

NAME

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

SYNOPSIS

/etc/brc

/etc/bcheckrc

/etc/rc

DESCRIPTION

These shell procedures are executed via entries in **/etc/inittab** by *init*(1M) when the system is changed out of *SINGLE USER* mode.

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

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

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

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

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

SEE ALSO

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

NAME

captoinfo — convert a termcap description into a terminfo description

SYNOPSIS

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

DESCRIPTION

captoinfo looks in *file* for *termcap* descriptions. For each one found, an equivalent *terminfo*(4) description is written to standard output, along with any comments found. A description which is expressed as relative to another description (as specified in the *termcap tc=* field) will be reduced to the minimum superset before being output.

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

The following options are recognized:

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

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

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

-w change the output to width characters.

FILES

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

SEE ALSO

tic(1M).
curses(3X), terminfo(4) in the *ICON/UXV Programmer Reference Manual.*
The chapter on curses in the *ICON/UXV Programmer Guide.*

NAME

checkall – file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

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

```
# check the root file system by itself
fsck /dev/is0a

# check the root file system and the user file system.
fsck /dev/is0a /dev/is0g
```

SEE ALSO

dfsck(1M), fsck(1M).

Setting Up the ICON/UXV System in the *ICON/UXV Administrator Guide*.

NAME

chroot — change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

The given command is executed *relative to the new root*. The meaning of any initial slashes (/) in 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

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

DESCRIPTION

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

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

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

SEE ALSO

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

BUGS

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

NAME

cpset — install object files in binary directories

SYNOPSIS

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

DESCRIPTION

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

If the user of *cpset* has administrative permissions (that is, the user's numerical ID is less than 100), the following defaults are provided:

mode — 0755

owner — bin

group — bin

If the user is not an administrator, the default, owner, and group of the destination file will be that of the invoker. An optional argument of **—o** will force *cpset* to move *object* to **OLD***object* in the destination directory before installing the new object. For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo All the examples above have the same effect (assuming the user is an administrator). The file **echo** will be copied into **/bin** and will be given **0755, bin, bin** as the mode, owner, and group, respectively. *Cpset* utilizes the file **/usr/src/destinations** to determine the final destination of a file. The locations file contains pairs of path names separated by spaces or tabs. The first name is the "official" destination (for example: **/bin/echo**). The second name is the new destination. For example, if *echo* is moved from **/bin** to **/usr/bin**, the entry in **/usr/src/destinations** would be:

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

Cross Generation

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

SEE ALSO

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

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files; users can submit their own crontab file via the *crontab* command. Commands which are to be executed only once may be submitted via the *at* command. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file **/etc/rc** (see *init*(8)). *Cron* only examines crontab files and at command files during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

FILES

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

SEE ALSO

at(1), crontab(1), sh(1), init(1M).

DIAGNOSTICS

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

NAME

dcheck — file system directory consistency check

SYNOPSIS

/etc/dcheck [**—i** numbers] [filesystem]

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(1M).

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The **—i** flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(1M), icheck(1M), fs(4), clri(1M), ncheck(1M)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

Since *dcheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

Dcheck is obsoleted by *fsck* and remains for historical reasons.

NAME

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

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

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

EXAMPLE

The command:

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

produces

```
is0g /usr
```

if **/usr** is mounted on **/dev/is0g**.

FILES

/dev/is*, /dev/sc*, /dev/hs*,
/etc/mnttab

SEE ALSO

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

NAME

df — report number of free disk blocks

SYNOPSIS

df [**—t**] [**—f**] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., **/dev/is0g**) 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/is*
/etc/mnttab

SEE ALSO

fs(4), mnttab(4).

NAME

diskusg — generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

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

uid login #blocks where

uid - the numerical user ID of the user.

login - the login name of the user; and

#blocks - the total number of disk blocks allocated to this user. *Diskusg* normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices. *Diskusg* recognizes the following options:

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

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

-i *fnmlist* ignore the data on those file systems whose file system name is in *fnmlist. Fnmlist* is a list of file system names separated by commas or enclose within quotes. *Diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit(1M)).*

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

-u *file* write records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID. The output of *diskusg* is normally the input to *acctdisk* (see *acct*(1M)) which generates total accounting records that can be merged with other accounting records. *Diskusg* is normally run in *dodisk* (see *acctsh*(1M)).

EXAMPLES

The following will generate daily disk accounting information:

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

FILES

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

SEE ALSO

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

NAME

dkfmt — standalone disk formatter

DESCRIPTION

Dkfmt is a program executable only from standalone mode. It is used to format the hard disk(s) at installation time.

SYNOPSIS

>stand/dkfmt [-f] [-b] [-h] [-p] [-Uunix_users] [-Ppick_users]

DESCRIPTION

Dkfmt is invoked from the standalone functions. This version is used with integrated SCSI disks (is0-is2). Refer to *dkfmt.hsmd* for use with HSMD type disks.

The following options are available: -f suppresses formatting, and is used to update the partitioning without reformatting the disk. If it is desired to re-write the boot block, the -b option should be specified with the -f option to force this. The -h option is used to hand-remap blocks which have been reported bad. The block number required is the absolute (zero-relative) block number (in hex) as reported from the SCSI driver. *Dkfmt* will attempt to read the bad block, then reassign it. The -p flag supresses the physical format, but otherwise does all other formatting, including write/read testing. The -U and -P options may be used to pre-specify values needed in estimating the swap partition size. If they are not specified, the user will be prompted for these values later on.

FILES

/stand/dkfmt

SEE ALSO

standalone(1M), dkfmt.hsmd(1M), and the current ICON/UX Release Note.

NAME

doscopyd — ICON/DOS file copy daemon

SYNOPSIS

/etc/doscopyd [line]

DESCRIPTION

Doscopyd is a server process which should be started in the */etc/rc.local* file. It provides support for copying files to and from ICON/DOS. The line argument may be specified to override the default data stream, which is /dev/mtty7. It is not normally necessary to specify this parameter.

Please refer to the *ICON/UXV Administrator Manual* for a full description of *doscopyd* and its ICON/DOS clients, *UCOPY* and *TAR*.

SEE ALSO

dosc(1),
ICON/UXV Administrator Manual

NAME

dosdisk — program to create and display information for ICON/DOS vdisks

SYNOPSIS

/etc/dosdisk [-n] [-v volname] [-c clustersize] [-r #rootdirents] [path] [size]

DESCRIPTION

Dosdisk is used by the system administrator to add vdisks for use by the ICON/DOS environment, or to display the vdisks currently defined. If *dosdisk* is entered without parameters, it will list all currently defined vdisks. If parameters are specified, there are two types of vdisks which can be identified to the system. The first is a "dos partition" type vdisk, which is supported for backward compatibility. In this case the path must be either /dev/sc0d or /dev/sc1d, and none of the other parameters can be specified. The path name is simply added to /etc/dosdisks. The other type of vdisk is an ICON/UXV file to be used as a vdisk. In this case, **path** specifies the pathname of a file which will be created to serve as the vdisk. This file cannot currently exist. The size must also be specified, and may be any value from 512K to 512M. (See NOTE.) The size may be specified as a number, a number followed by "k or K" which multiplies the value given by 1024, or a number followed by "m or M" which multiplies the value given by 1024*1024. The -v, -c and -r options are summarized below:

-v volname
Specifies the name to be used in initializing the disk.

-c clustersize
Specifies the cluster size in megabytes, with a maximum of 32.

-r #rootdirents
Number of entries allowed in the DOS root directory.

-n Must be specified for vdisks used for Novell® file servers

Please refer to the *Technical Note on the Implementation of Dosc and Proc/286 Software Support* for a full description of ICON/DOS vdisk support.

FILES

/etc/dosdisks vdisk description file

SEE ALSO

dosc(1), *Technical Note on the Implementation of Dosc and PROC286 Software Support*

NOTES

Please note that in release 3.00 of the ICON/UXV operating system, not all sizes of vdisks have been tested. The following sizes of vdisks have been tested and appear to work successfully:

512K through 256M
500M

NAME

dosprint — ICON/DOS spooler daemon

SYNOPSIS

/etc/dosprint [line [delay]]

DESCRIPTION

Dosprint is a server process which should be started in the */etc/rc.local* file. It provides spooled printer support for up to eight virtual printers to ICON/DOS users. The optional arguments are to override the default input stream (*/dev/mtty6*), and the default timeout delay (10 seconds). If only the delay is to be changed, */dev/mtty6* must be specified as the first parameter. It should not normally be necessary to specify either of these parameters.

Please refer to the *ICON/UXV Administrator Manual* for a full description of ICON/DOS spooled printer support.

FILES

/etc/dosprinters MS-DOS printer description file

SEE ALSO

dosc(1)
ICON/UXV Administrator Manual

NAME

dumpfs — dump file system information

SYNOPSIS

dumpfs *filesys|device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(4), disktab(4), tunefs(1M), newfs(1M), fsck(1M)

NAME

fdump — incremental file system dump

SYNOPSIS

/etc/fdump [key [*argument* ...] filesystem]

DESCRIPTION

Fdump copies to magnetic tape all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the fdump. *Key* consists of characters from the set **0123456789fusdW.**

0—9 This number is the 'dump level'. All files modified since the last date stored in the file */etc/dumpdates* for the same filesystem at lesser levels will be fdumped. If no date is determined by the level, the beginning of time is assumed; thus the option **0** causes the entire filesystem to be fdumped.

f Place the fdump on the next *argument* file instead of the tape. If the name of the file is "—", *fdump* writes to standard output.

u If the fdump completes successfully, write the date of the beginning of the fdump on file */etc/dumpdates*. This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people, consisting of one free format record per line: filesystem name, increment level and *ctime(3)* format fdump date. */etc/dumpdates* may be edited to change any of the fields, if necessary.

s The size of the fdump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *fdump* will wait for reels to be changed. The default tape size is 2300 feet. The c flag must be set to zero to enable the s flag.

d The density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per reel. The default is 1600. The c flag must be set to zero to enable the d flag.

c The capacity of the tape, specified in megabytes (1,000,000 bytes, not 1024 X 1024 bytes), is taken from the next *argument*. This is useful for cartridge and cassette media because without this option, the *fdump* command would make assumptions about inter-record gaps and recording density which would be incorrect for these media. For example,

fdump 0cf 60 /dev/qic24 /dev/sc0g

specifies a "0" fdump, on a 60 MB tape (/dev/qic24), of the filesystem on /dev/sc0g. The default fdump assumes a CS20 (21 MB), so for half-inch tape, it is necessary to specify a capacity of zero in order to enable the density and length options. For example,

fdump 0cfsd 0 /dev/mt0 2400 1600 /dev/sc0g

specifies a "0" level fdump, on a 2400 foot tape at 1600 bpi density, of the file system /dev/sc0g.

W *Fdump* tells the operator what file systems need to be dumped. This information is gleaned from the file */etc/dumpdates*. The **W** option causes *fdump* to print out, for each file system in */etc/dumpdates* the most recent fdump date and level, and highlights those file systems that should be fdumped. If the **W** option is set, all other options are ignored, and *fdump* exits immediately.

w Is like W, but prints only those filesystems which need to be dumped.

If no arguments are given, the
key is assumed to be **9u** and a default file system is dumped to the default tape.

Fdump requires operator intervention on these conditions: end of tape, end of fdump, tape write error, tape open error or disk read error (if there are more than a threshold of 32). *Fdump* interacts with the operator on *fdump's* control terminal at times when *fdump* can no longer proceed, or if something is grossly wrong. All questions *fdump* poses **must** be answered by typing "yes" or "no", appropriately.

Since making an fdump involves a lot of time and effort for full fdumps, *fdump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *fdump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Fdump tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change. The output is verbose, so that others know that the terminal controlling *fdump* is busy, and will be for some time.

Now a short suggestion on how to perform fdumps. Start with a full level 0 fdump

fdump 0un

Next, fdumps of active file systems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of fdump levels:

3 2 5 4 7 6 9 8 9 ...

For the daily fdumps, a set of 10 tapes per dumped file system is used on a cyclical basis. Each week, a level 1 fdump is taken, and the daily Hanoi sequence repeats with 3. For weekly fdumps, a set of 5 tapes per dumped file system is used, also on a cyclical basis. Each month, a level 0 fdump is taken on a set of fresh tapes that is saved forever.

FILES

/dev/sc0a default filesystem to dump from

/dev/ct0	default tape unit to dump to
/etc/dumpdates	new format dump date record

SEE ALSO

restore(1m),

DIAGNOSTICS

Many, and verbose.

Fdump exits with zero status on success. Startup errors are indicated with an exit code of 1; abnormal termination is indicated with an exit code of 3.

BUGS

Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

Fdump with the **W** or **w** options does not report filesystems that have never been recorded in /etc/dumpdates.

It would be nice if *fdump* knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running *restore*.

NAME

ff — list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

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

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

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

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

EXAMPLES

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

```
ff -I /dev/diskroot
```

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

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

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

```
ff -i 451,76 /dev/is0g
```

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 ICON/UXV system file system backup

SYNOPSIS

/etc/filesave.?
/etc/tapesave

DESCRIPTION

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

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

SEE ALSO

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

NAME

finc — fast incremental backup

SYNOPSIS

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

DESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-tape* . The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit* (see *volcopy*(1M)). The selection is controlled by the *selection-criteria*, accepting only those i-nodes/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 cassette tape consisting of all files from file-system **/usr** , mounted on /dev/is0g, modified in the last 48 hours:

```
finc —m —2 /dev/is0g /dev/ct0
```

SEE ALSO

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

NAME

flfmt — format floppy disks

SYNOPSIS

/etc/flfmt *special* [*passes*]

DESCRIPTION

The *flfmt* program formats a floppy disk with the format associated with the special device *special*. *Special* is normally /dev/flh, for high density floppy disks, and /dev/fld, for double density floppy disks. *Passes* may be supplied to specify the number of verify passes performed on the floppy. If *passes* is not specified, no verify is performed. High density format is compatible with the IBM AT standard diskette format (512 bytes/sector). Double density format is compatible with the IBM PC standard diskette format (512 bytes/sector). Although the formatting is compatible, file system information is not placed on the diskette. Additional file system information may be needed to allow a particular utility on an IBM machine to read the diskette.

Before formatting a diskette *flfmt* prompts for the user to insert the diskette. If the user does not want to format the diskette currently in the drive he may replace it or abort the operation. (note that formatting a diskette will destroy any existing data). Formatting is done by the hardware.

FILES

/dev/flh /dev/fld

SEE ALSO

fl(4)

AUTHOR

Mark Clement

BUGS

Use of the flfmt utility seriously degrades system performance.

NAME

frec – recover files from a backup tape

SYNOPSIS

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

DESCRIPTION

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

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

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

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

EXAMPLES

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

```
frec /dev/ct0 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/ct0 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 path name) *frec* can only recover i-node fields for those directories contained on the tape and requested for recovery.

NAME

fsck — file system consistency check and interactive repair

SYNOPSIS

/etc/fsck **—p** [filesystem ...]
/etc/fsck [**—b** block#] [**—y**] [**—n**] [filesystem] ...

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. Only partitions in fstab that are mounted "rw" or "rq" and that have non-zero pass number are checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies that *fsck* with the **—p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If sent a QUIT signal, *fsck* will finish the file system checks, then exit with an abnormal return status that causes the automatic reboot to fail. This is useful when you wish to finish the file system checks, but do not want the machine to come up multiuser.

Without the **—p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for

concurrence before each correction is attempted. It should be noted that some of the corrective actions which are not correctable under the **—p** option 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 on the file system *fsck* will default to a **—n** action.

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

The following flags are interpreted by *fsck.*

—b Use the block specified immediately after the flag as the super block for the file system. Block 32 is always an alternate super block.

—y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

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

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:
   Directory size not of proper format.
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 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. The name assigned is the inode number. If the *lost+found* directory does not exist, it is created. If there is insufficient space its size is increased.

SEE ALSO

newfs(1M), mkfs(1M), reboot(1M)

BUGS

There should be some way to start a **fsck** **–p** at pass *n*.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

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

DESCRIPTION

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

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

EXAMPLES

fuser –ku /dev/is1?
: will terminate all processes that are preventing disk drive one from being unmounted if typed by the super-user, listing the process ID and login name of each as it is killed.

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

fuser –ku /dev/is1? –u /etc/passwd
: will do both of the above examples in a single command line.

FILES

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

SEE ALSO

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

NAME

fwtmp, wtmpfix — manipulate connect accounting records

SYNOPSIS

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

DESCRIPTION

Fwtmp

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

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

Wtmpfix

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

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

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

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), ed(1), 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 ICON/UXV 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* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character). The default *speed* is 300 baud. The optional third argument, *type*, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* understands all types of terminals defined in */usr/lib/terminfo*. The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. 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, new-line 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
/etc/issue

SEE ALSO

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

BUGS

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

NAME

halt — stop the processor

SYNOPSIS

/etc/halt [**—n**] [**—q**] [**—y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor.

The **—n** option prevents the sync before stopping. The **—q** option causes a quick halt, no graceful shutdown is attempted. The **—y** option is needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(1M), shutdown(1M)

NAME

icheck — file system storage consistency check

SYNOPSIS

/etc/icheck [**—s**] [**—b** numbers] [filesystem]

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(1M).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; i.e. not in any file nor in the free list.

The **—s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **—s** option causes the normal output reports to be suppressed.

Following the **—b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(1M), dcheck(1M), ncheck(1M), fs(4), clri(1M)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

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 ICON/UXV 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 collocated with the processor.

When *init* prompts for the new *run-level*, the operator may enter only 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 does not 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.

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

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

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

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

FILES

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

SEE ALSO

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

DIAGNOSTICS

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

NAME

install — install commands

SYNOPSIS

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

DESCRIPTION

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

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

If one or more directories (*dirx ...*) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

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

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

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

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

—o If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a normally text busy file such as

/bin/sh or **/etc/getty**, where the existing file cannot be removed. May be used alone or with any other options other than **–c**.

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

SEE ALSO

cpset(1S), make(1), mk(8).

NAME

kickdosdisk — program to start DOS processing

DESCRIPTION

Kickdosdisk is a program executed at boot time to kick the disk processor and enable it to configure the DOS option if it exists. Determination of DOS configuration is dependant on which file systems are mounted, so ***kickdosdisk*** must be run after all filesystems related to DOS are mounted. It actually just modifies a location on the disk processor with **/dev/dmem.**

FILES

/etc/kickdosdisk

SEE ALSO

dmem (7)

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

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

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

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

FILES

/etc/shutdown

SEE ALSO

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

NAME

link, unlink — exercise link and unlink system calls

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

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

SEE ALSO

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

NAME

loadpcp — program to load the PCP software

DESCRIPTION

Loadpcp is a program executed at boot time to load the software for the Peripheral Communication Processor (PCP). It loads the file **pcpimage** into PCP local memory using the special device **/dev/kmem.** If there is a file named **pcpimage** in the directory where the kernel was loaded from, *loadpcp* will download this file. Otherwise *loadpcp* will use the file **/etc/pcpimage** if it exists.

FILES

/pcpimage

SEE ALSO

kmem (7), pcp (7).

DIAGNOSTICS

PCP #%d not responding .

The PCP is not running properly. This may have happened because the program **/etc/loadpcp** failed for some reason. There may have been a corrupted **pcpimage** or there may be hardware problems with the PCP board.

NAME

lpadmin — configure the LP spooling system

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

–l indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*(1M), disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.

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

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

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

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

Models.
Model printer interface programs are supplied with the LP software. They are shell procedures which interface between *lpsched (1M)* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **—m**. Models should have 644 permission if owned by lp & bin, or 664 permission if owned by bin & bin. 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** do not use the *450*(1) filter. The output has been pre-processed by either *450*(1) or the *nroff (1)* 450 driving table.

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

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

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

EXAMPLES

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

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

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

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

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

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

4. An *nroff (1)* 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*(1).

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

makedev — make system special files

SYNOPSIS

/dev/MAKEDEV *device...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form *device-name*? where *device-name* is one of the supported devices listed in section 4 of the manual and "?" is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; e.g. /dev/console, /dev/tty.

local Create those devices specific to the local site. This request causes the shell file */dev/MAKEDEV.local* to be executed. Site specific commands, such as those used to setup dialup lines as "ttyd?" should be included in this file.

Since all devices are created using *mknod*(1M), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use "sh -x MAKEDEV" in case of trouble.

SEE ALSO

intro(4), config(1M), mknod(1M)

BUGS

When more than one piece of hardware of the same "kind" is present on a machine (for instance, a dh and a dmf), naming conflicts arise.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

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

FILES

/usr/spool/lp/*

SEE ALSO

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

NAME

mkfs — construct a file system

SYNOPSIS

/etc/mkfs [–N] special size [nsect [ntrack [blksize [fragsize [ncpg [minfree [rps [nbpi [opt]]]]]]]]]

DESCRIPTION

N.B.: file systems are normally created with the *newfs*(1M) command.

Mkfs constructs a file system by writing on the special file *special* unless the –N flag has been specified. The numeric size specifies the number of sectors in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory. (see *fsck*(1M)) The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(1M).)

The optional arguments allow fine tune control over the parameters of the file system. **Nsect** specify the number of sectors per track on the disk. **Ntrack** specify the number of tracks per cylinder on the disk. **Blksize** gives the primary block size for files on the file system. It must be a power of two, currently selected from 4096 or 8192. **Fragsize** gives the fragment size for files on the file system. The **fragsize** represents the smallest amount of disk space that will be allocated to a file. It must be a power of two currently selected from the range 512 to 8192. **Ncpg** specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32. **Minfree** specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10%. If a disk does not revolve at 60 revolutions per second, the **rps** parameter may be specified. If a file system will have more or less than the average number of files the **nbpi** (number of bytes per inode) can be specified to increase or decrease the number of inodes that are created. Space or time optimization preference can be specified with **opt** values of "s" for space or "t" for time. Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

SEE ALSO

dir(4), fsck(1M), newfs(1M), tunefs(1M)

M. McKusick, W. Joy, S. Leffler, R. Fabry, "A Fast File System for UNIX", *ACM Transactions on Computer Systems 2*, 3. pp 181-197, August 1984.

BUGS

There should be some way to specify bad blocks.

NAME

mknod — build special file

SYNOPSIS

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

DESCRIPTION

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

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

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

SEE ALSO

mknod(2).

NAME

mount, umount — mount and dismount file system

SYNOPSIS

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

/etc/umount special

DESCRIPTION

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

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

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

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

FILES

/etc/**mnttab** mount table

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 moves 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

/etc/ncheck [**—i** numbers] [**—a**] [**—s**] filesystem

DESCRIPTION

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

Ncheck with no argument generates a pathname 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 must be specified.

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

SEE ALSO

sort(1), dcheck(1M), fsck(1M), icheck(1M)

DIAGNOSTICS

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

NAME

newfs — construct a new file system

SYNOPSIS

/etc/newfs [**—v**] [**mkfs-options**] **filesystem**

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(1M) program. *Newfs* will calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs*. The **filesystem** argument is the partition on the device on which you wish to create the new file system. For example, */dev/is0g*.

If the **—v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

—s size The size of the file system in sectors.

—b block-size
The block size of the file system in bytes.

—f frag-size
The fragment size of the file system in bytes.

—t #tracks/cylinder

—c #cylinders/group
The number of cylinders per cylinder group in a file system. If not given, a per drive type value is calculated.

—m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

—o optimization preference ("space" or "time")
The file system can either be instructed to try to minimize the time spent allocating blocks, or to try to minimize the space fragmentation on the disk. If the value of minfree (see above) is less than 10%, the default is to optimize for space; if the value of minfree greater than or equal to 10%, the default is to optimize for time.

—r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

—S sector-size
The size of a sector in bytes (almost never anything but 512).

—i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller

number should be given.

EXAMPLE

To create a filesystem on */dev/hs00*, you would type the following,

newfs -c 16 /dev/hs00

FILES

/etc/mkfs to actually build the file system

SEE ALSO

fsck(1M), mkfs(1M), tunefs(1M)

M. McKusick, W. Joy, S. Leffler, R. Fabry, "A Fast File System for UNIX", *ACM Transactions on Computer Systems 2*, 3. pp 181-197, August 1984.

NAME

park — program to park the hard disk heads

DESCRIPTION

Park is a program executable only from standalone mode. It is used to park the heads on the hard disk(s), so that they will not damage the disk if the machine is jarred while moving it.

It is always a good idea to park the disk heads, no matter how short a distance the machine may be moved.

FILES

/stand/park

SEE ALSO

standalone(8)

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 ICON/UXV System in the *ICON/UXV Administrator 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 ICON/UXV System in the *ICON/UXV Administrator Guide.*

DIAGNOSTICS

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

NAME

reboot – ICON/UXV bootstrapping procedures

SYNOPSIS

/etc/reboot [**–n**] [**–q**]

DESCRIPTION

Rebooting a running system. When a ICON/UXV is running and a reboot is desired, *shutdown*(1M) is normally used. If there are no users then **/etc/reboot** can be used usually. Reboot causes the disks to be synced, and then a reboot (as described below) is initiated. This causes a system to be booted.

Options to reboot are:

–n option avoids the sync. It can be used if a disk or the processor is on fire.

–q reboots quickly and ungracefully, without shutting down running processes first.

Power fail and crash recovery. Normally, the system will reboot itself at power-up. When the system crashes, it will hang up with an error message, generally, being printed on the console. The system must then be rebooted by turning the keyswitch to the reset position (this is a spring loaded position).

FILES

/vmunix virtual memory system code
/m1unix main system code
/dcunix disk cache code

SEE ALSO

crash(8), **fsck(1M)**, **init**(1M), rc(1M), shutdown(1M), halt(1M), newfs(1M)

NAME

restore — incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

Restore reads tapes dumped with the *fdump*(1m) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete fdump tape onto a clear file system or to restore an incremental fdump tape after a full level zero restore. Thus

```
/etc/newfs /dev/isl1g eagle
/etc/mount /dev/isl1g /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete fdump. Another *restore* can be done to get an incremental fdump in on top of this. Note that *restore* leaves a file *restoresymtab* in the root directory to pass information between incremental restore passes. This file should be removed when the last incremental tape has been restored.
A *fdump*(1m) followed by a *newfs*(1m) and a *restore* is used to change the size of a file system.

R *Restore* requests a particular tape of a multi volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, then the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified.

i This mode allows interactive restoration of files from a fdump tape. After reading in the directory information from the tape, *restore* provides a shell like

interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

ls [arg] — List the current or specified directory. Entries that are directories are appended with a "/". Entries that have been marked for extraction are prepended with a "*". If the verbose key is set the inode number of each entry is also listed.

cd arg — Change the current working directory to the specified argument.

pwd — Print the full pathname of the current working directory.

add [arg] — The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, then it and all its descendents are added to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a "*" when they are listed by **ls**.

delete [arg] — The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, then it and all its descendents are deleted from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete those files that are not needed.

extract — All the files that are on the extraction list are extracted from the fdump tape. *Restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

setmodes — All the directories that have been added to the extraction list have their owner, modes, and times set; nothing is extracted from the tape. This is useful for cleaning up after a restore has been prematurely aborted.

verbose — The sense of the **v** key is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* to print out information about each file as it is extracted.

help — List a summary of the available commands.

quit — Restore immediately exits, even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired.

b The next argument to *restore* is used as the block size of the tape (in kilobytes). If the **-b** option is not specified, *restore* tries to determine the tape block size dynamically.

f The next argument to *restore* is used as the name of the archive instead of /dev/rmt?. If the name of the file is "–", *restore* reads from standard input. Thus, *fdump*(1m) and *restore* can be used in a pipeline to fdump and restore a file system with the command

```
fdump 0f - /usr | (cd /mnt; restore xf -)
```

v Normally *restore* does its work silently. The **v** (verbose) key causes it to type the name of each file it treats preceded by its file type.

y *Restore* will not ask whether it should abort the restore if gets a tape error. It will always try to skip over the bad tape block(s) and continue as best it can.

m *Restore* will extract by inode numbers rather than by file name. This is useful if only a few files are being extracted, and one wants to avoid regenerating the complete pathname to the file.

h *Restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

s The next argument to *restore* is a number which selects the file on a multi-file fdump tape. File numbering starts at 1.

DIAGNOSTICS

Complaints about bad key characters.

Complaints if it gets a read error. If **y** has been specified, or the user responds "y", *restore* will attempt to continue the restore.

If the fdump extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** key has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are given below.

Converting to new file system format.
A fdump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

<filename>: not found on tape
The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a fdump tape created on an active file system.

expected next file <inumber>, got <inumber>
A file that was not listed in the directory showed up. This can occur when using a fdump tape created on an active file system.

Incremental tape too low
When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
When doing incremental restore, a tape that does not begin its coverage where

the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>
Tape read error while skipping over inode <inumber>
Tape read error while trying to resynchronize
A tape read error has occurred. If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> blocks
After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/ct0	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtable	information passed between incremental restores.

SEE ALSO

fdump(1m), newfs(1m), mount(1m), mkfs(1m)

BUGS

Restore can get confused when doing incremental restores from fdump tapes that were made on active file systems.

A level zero fdump must be done after a full restore. Because restore runs in user code, it has no control over inode allocation; thus a full restore must be done to get a new set of directories reflecting the new inode numbering, even though the contents of the files is unchanged.

NAME

runacct — run daily accounting

SYNOPSIS

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

DESCRIPTION

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

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

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

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

CLEANUP Cleanup temporary files and exit.

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

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), mail(1), acct(2), acct(4), utmp(4).

ICON/UXV System Accounting System in the *ICON/UXV Administrator Guide.*

BUGS

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

runacct *mmdd* WTMPFIX

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

NAME

sadp — disk access profiler

SYNOPSIS

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

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. Disk activity is continuously monitored during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

The valid values of *device* are **sc**, **is** and **hs0** through **hs3**. *Drive* specifies the disk drives and it may be:

a drive number in the range supported by *device*,
two numbers separated by a minus (indicating an inclusive range),

or

a list of drive numbers separated by commas.

Up to 8 disk drives may be reported. The **—d** option may be omitted, if only one *device* is present.

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

EXAMPLE

The command:

sadp —d sc —0 -d hs0 —0,1 900 4

will generate 4 tabular reports, each describing cylinder usage and seek distance of sc drive 0 and hs0 drives 1 and 2 during a 15-minute interval.

FILES

/dev/kmem
/dev/dmem

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 [**–ubdycwaqvwprA**] [**–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 interprocess communications.

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

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

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

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

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

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

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

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

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

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
        struct minfo mi;  /* defined in /usr/include/sys/sysinfo.h */
        int  szinode;       /* current size of inode table  */
        int  szfile;        /* current size of file table  */
        int  szproc;        /* current size of proc table  */
        int   szlckf;       /* current size of file record header table */
        int   szlckr;       /* current size of file record lock table */
        int  mszinode;      /* size of inode table  */
        int  mszfile;       /* size of file table  */
        int  mszproc;       /* size of proc table  */
        int   mszlckf;     /* maximum size of file record header table */
        int   mszlckr;      /* maximum size of file record lock table */
        long  inodeovf;     /* cumulative overflows of inode table  */
        long  fileovf;      /* cumulative overflows of file table  */
        long  procovf;      /* cumulative overflows of proc table  */
        time_t  ts;         /* time stamp  */
        int   apstate;
        long  devio[NDEVS][4];      /* device unit information */
#define IO_OPS          0           /* cumulative I/O requests  */
#define IO_BCNT         1           /* cumulative blocks transferred */
#define IO_ACT          2           /* cumulative drive busy time in ticks  */
#define IO_RESP         3           /* cumul. I/O resp time in ticks since boot  */
};
```

FILES

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

SEE ALSO

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

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

FILES

/etc/mnttab

SEE ALSO

mount(1M), mnttab(4).

BUGS

Evil things will happen if *filesys* or *node* are longer than 32 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries. This upper limit is 20 entries.

NAME

shutdown — terminate all processing

SYNOPSIS

/etc/shutdown

DESCRIPTION

Shutdown is part of the ICON/UXV 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

smilecopyd – SMILE file copy daemon

SYNOPSIS

/etc/smilecopyd

DESCRIPTION

Smilecopyd is a server process which should be started in the rc.local file. It provides support for copying files to and from computers connected to SMILE.

Please refer to the "Technical Note on SMILE" for a full description of *smilecopyd* and its SMILE utilities, *UCOPY* and *TAR*.

SEE ALSO

"Technical Note on SMILE"

NAME

smiledisk — program to create and display information for SMILE vdisks

SYNOPSIS

/etc/smiledisk [-v volumename] [-c clustersize] [-r #rootdirents] path [size] label description

/etc/smiledisk [-l]

DESCRIPTION

Smiledisk is used by the system administrator to add *vdisks* for use by computers connected to SMILE or to display the vdisks currently defined. If *smiledisk* is entered without parameters, it will list all currently defined vdisks. The -l option will list the number of cylinders, heads, sectors, and sector size for each vdisk. If other parameters are specified, there are two types of vdisks which can be identified to the system. The first is a "DOS partition" type vdisk, which is supported for backward compatibility. In this case **path** must be either /dev/sc0d or /dev/sc1d, and label and description are the only other parameters allowed. The other type of vdisk is a ICON/UX file to be used as a vdisk. In this case, **path** specifies the pathname of a file which will be created to serve as the vdisk. This file cannot currently exist. The size must also be specified, and may be any value from 512K to 512M. (See NOTE.) The size may be specified as a number, a number followed by "k" which multiplies the value given by 1024, or a number followed by "m" which multiplies the value given by 1024*1024. **Label** is a string (up to 5 characters) that is used to identify disks in the local configuration files. **Description** is a string to give information about the drive. It can be up to 45 characters in length. The -v, -c and -r options allow specification of the volume name, cluster size, and number of directory entries in the root directory for the newly created vdisk. Users should not normally specify the clustersize for vdisks larger than 32M.

Please refer to the "Technical Note on SMILE" for a full description of SMILE vdisk support.

FILES

/etc/smiledisks	vdisk description file
/etc/smiledisks_00	vdisk local configuration files
...	
/etc/smiledisks_05	

SEE ALSO

"Technical Note on SMILE"

NOTES

Please note that in release MPS/UX release 2.15, not all sizes of vdisks have been tested. The following sizes of vdisks have been tested and appear to work successfully:

512K through 256M
500M

The next release may support up to 1G vdisks, and all sizes up to the max will be supported. Users needing sizes which have not been tested are welcome to try them; they should work.

NAME

smileprint – SMILE spooler daemon

SYNOPSIS

/etc/smileprint [line [delay]]

DESCRIPTION

Smileprint is a server process which should be started in the rc.local file. It provides spooled printer support for up to eight virtual printers to each computer attached to SMILE. The optional arguments are to override the default input stream (/dev/smlp00), and the default timeout delay (10 seconds). If only the delay is to be changed, /dev/smlp00 must be specified as the first parameter. It should not normally be necessary to specify either of these parameters.

Please refer to the "Technical Note on SMILE" for a full description of SMILE spooled printer support.

FILES

/etc/smileprinters SMILE printer description file

SEE ALSO

"Technical Note on SMILE"

NAME

tic — terminfo compiler

SYNOPSIS

tic [**–v**[*n*]] file ...

DESCRIPTION

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

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

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

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

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

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

SEE ALSO

curses(3X), terminfo(4).

BUGS

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

NAME

trenter — enter a trouble report

SYNOPSIS

trenter [**–s**]

DESCRIPTION

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

Fields and Values

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

The first nine fields identify the originator of the report.

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

The next two fields identify the processor on which the problem occurred.

- CPU Serial Number (CPUNO) (*)

- Machine type (MACH)

The following fields identify the area in which the problem occurred.

- Trouble Report Type (TYPE)
  Valid responses: **doc** (documentation), **enh** (enhancement), **cs** (customer support), **fw** (firmware), **hdw** (hardware), **sw** (software), **unk** (unknown).

- Icon Product Name (PROD)
  Examples: ICON/UXV, sdb

- Operating System Release (OS_REL) (*)
  The release of ICON/UX on which the problem occurred.

The remaining fields define the body of the trouble report.

- Severity (SEV)
  The severity of the problem (1-4).

- Required Date (RDATE)
  If the severity of the report is 2, the required date for the fix is prompted. The date given must be at least one week from the date of the trouble report.

- Abstract (ABS)
  One-line description of the problem.

- Description (DESC)
  Full description of the problem. Note that description input will not be passed through *nroff*; however, *trenter* will recognize the macros **.ES** and **.EE** (example start, example end), indicating an indented example (these may be nested).

- Attachments (yes or no) (ATT)

If ? is given in response to a prompt, a message explaining the field will be printed.

If *trenter* receives an interrupt during prompting, the trouble report will be aborted.

After a trouble report has been completed, the user is given an opportunity to edit any data that has been supplied. Next, a reprint of the trouble report just entered may be requested. Finally, the user is asked whether another report is to be entered. If so, the values for the starred items in the field table above will be carried over from the first report.

Editing Field Values

In order to provide editing while responding to prompts, the following *escapes* are recognized on input:

- *–field*
  Return to a field for which data has previously been supplied. If the field name is not specified, return to the previous field. The value already assigned to the field is printed and the user may enter either new data or another editing command.

- **!e**
  Invoke the editor *ed*(1) with any text already supplied for the current prompt in the edit buffer (an alternate editor can be specified; refer to "Specifying Default Values" below).

- **>**
  Move down to the first unfilled field. This is useful, for example, when the - command has been used to fix a single field near the top of the report and the user wishes to quickly return to the point where he/she left off.

- **=field**
  Print the value currently assigned to the given field.

- **??**
  Print a summary of editing functions.

Editing commands are only recognized when they appear at the beginning of the input line; they may be escaped using a backslash ().

Specifying Default Values

Users may provide default values for any fields marked with an asterisk (*) above. These values are specified in a file **.trdef** in the user's home directory. Entries in this file are of the form:

field=value

where *field* is a field name from the table above.

The editor to be used for field editing can be overridden with a **.trdef** entry be assigning the name of the desired program to the field EDITOR.

During prompting, *trenter* will print any values supplied for fields from a **.trdef** file. By default, it will stop at each such field and wait for either a carriage return (indicating confirmation) , an edit command, or new data. If invoked with a -s option, *trenter* will print the supplied values, but will not stop for confirmation.

Default values specified in **.trdef** files may be changed, on a per-report basis, using the editing functions described above.

Administration

When an ordinary user enters a report with *trenter*, the report is spooled in the /usr/spool/trenter directory. To send the report to the ICON Customer Service machine (iconserv) someone must login to the *adm* account and run *trenter*. *Trenter* then enters admin mode and allows one to: delete, edit, print, enter, or send trouble reports. Typing '?' at the prompt will display the following message:

Options:
- d delete TRs
- e edit TRs
- l list known TR numbers
- n enter TRs (does not imply send)
- p print TRs
- q exit trenter
- s send TRs to Customer Support
- ? this message

Listing the trouble reports will display something like:

icon.66780 ($0) icon.68621 ($1) icon.69363 ($2)
icon.25403 ($3)

The numbers in parentheses (including the '$') are used to designate the particular trouble report you wish to address with each command. Thus

Request: e $2

will enable you to edit trouble report icon.69363.

The **p** command prints the desired trouble report to the screen. The **n** allows the administrator to enter his own trouble report.

FILES

/usr/*homedir*/.trdef	default value file
/usr/spool/trenter	spool directory
/usr/spool/trenter/.trdef	default value file

NAME

tunefs — tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special|filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

—a maxcontig
This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see —d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

—d rotdelay
This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

—e maxbpg
This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

—m minfree
This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(1M), mkfs(1M)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

This program should work on mounted and active file systems. Because the superblock is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

You can tune a file system, but you can't tune a fish.

NAME

uuclean — uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**options**]

DESCRIPTION

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

The following options are available.

—d*directory* Clean *directory* instead of the spool directory. If *directory* is not a valid spool directory it cannot contain "work files" i.e., files whose names start with "C.". These files have special meaning to **uuclean** pertaining to **uucp** job statistics.

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

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

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

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

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

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

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(1M) defines a *uucp* subnetwork and monitors the connection and traffic among the members of the subnetwork. The following options are available:

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

The meanings of the connections report are:

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

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

The meanings of the traffic statistics are:

sfile sbyte rfile rbyte

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

The command:

uusub —c all —u 24

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

FILES

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

SEE ALSO

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

NAME

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:

—bpidensity	bits-per-inch (i.e., **800/1600/6250**),
—feetsize	size of reel in feet (i.e., **1200/2400**),
—reelnum	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. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (e.g.,: *labelit*) and return to *volcopy* by exiting the new shell.

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

The *special* should be the physical disk section or tape (e.g.,: **/dev/is0g**, **/dev/ct0**, etc.).

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

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

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

Labelit can be used to provide initial labels for unmounted disk or tape file systems. With the optional arguments omitted, *labelit* prints current label values. The **–n** option provides for initial labeling of new tapes only (this destroys previous contents). If *Labelit* is given arguments, it will display the new label and wait for ten seconds before writing the label out to the device. During this ten second waiting period, you can decide not to label the device by pressing the DEL key, which will cause *Labelit* to stop without writing the labels.

FILES

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

SEE ALSO

sh(1), fs(4).

BUGS

Only device names **/dev/[r]ct0, /dev/[r]mt[0-9], /dev/[r]qic24, /dev/[r]qic11,** are treated as tapes.

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.

FILES

etc/passwd

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.

The following table relates the device names in /dev to the corresponding manual pages.

DEVICE	MANUAL PAGE
/dev/acebrg	ace(7)
/dev/ct0	ct(7)
/dev/rct0	ct(7)
/dev/dcs[0-3]	dcs(7)
/dev/tty[0-3][1-f][0-f]	dcs(7)
/dev/cu[0-3][1-f][0-f]	dcs(7)
/dev/lp[0-3][1-f]	dcs(7)
/dev/flh	fl(7)
/dev/fld	fl(7)
/dev/hs[0-7][0-3][a-h]	hs(7)
/dev/rhs[0-7][0-3][a-h]	hs(7)
/dev/is[0-2][a-h]	is(7)
/dev/ris[0-2][a-h]	is(7)
/dev/klog	klog(7)
/dev/lp	lp(7)
/dev/rlp	lp(7)
/dev/nfflp	lp(7)
/dev/dmem	mem(7)
/dev/kmem	mem(7)
/dev/mba	mem(7)
/dev/console	mot(7)
/dev/tty01	mot(7)
/dev/mt0	mt(7)
/dev/rmt0	mt(7)
/dev/hmt0	mt(7)
/dev/mtty[0-7]	mtty(7)
/dev/null	null(7)
/dev/tty[a-d][0-9,a-f]	pcp(7)
/dev/lp[a-d]	pcp(7)
/dev/rlp[a-d]	pcp(7)
/dev/nfflp[a-d]	pcp(7)
/dev/pty[p-r][0-9a-f]	pty(7)
/dev/tty[p-r][0-9a-f]	pty(7)
/dev/qic24	qic(7)
/dev/rqic24	qic(7)

/dev/qic11	qic(7)
/dev/rqic11	qic(7)
/dev/sc[0-1][a-h]	sc(7)
/dev/rsc[0-1][a-h]	sc(7)
/dev/sfi0[0-5]	sfi(7)
/dev/mb_[0-9,a-f]	ssi(7)
/dev/ttys[0-5][0-9,a-f]	sti(7)
/dev/sxt/??[0-7]	sxt(7)
/dev/tty	tty(7)

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

ace – AUSTEC COBOL special semaphore driver

DESCRIPTION

The file acebrg is used to implement semaphores for the AUSTEC COBOL driver.

FILES

/dev/acebrg special file

NAME

ct — cassette tape interface

DESCRIPTION

The file **/dev/ct0** refers to the normal ICON cassette tape interface. When opened for reading or writing, the tape is assumed to be positioned as desired. Data written to this device is organized in 512 byte blocks, and ICON/UXV buffers 128 of these blocks per physical write. Therefore, the optimal block size for reads and writes is 64K bytes. If the length of the data to be written is not a multiple of 512 when the device is closed, the last block is padded with zero bytes before it is written. When the device is closed after writing, one file mark is written at the end of the data. After closing, the tape is rewound. The EOF is returned as a zero-length read. Seeks are ignored.

The file **/dev/rct0** refers to the non-rewind on close ICON cassette tape interface. This device is exactly like **/dev/ct0** except that the rewind on close is suppressed.

CS20 Cassette Tape Drive

This drive is normally shipped as the standard drive for ICON systems. It provides approximately 22 MB of storage on a standard 500 foot cassette, and approximately 26.5 MB on a 600 foot tape. The data is stored in a serpentine fashion with four recording tracks. Data is stored in fixed length blocks of 512 bytes, similar to the QIC formats. The media is available from ICON and large computer equipment suppliers. This drive provides an inexpensive, reliable backup capability, but it does not enjoy the media interchangeability of more standard devices.

CS50 Cassette Tape Drive

This drive is similar to the CS20, but it has nine tracks instead of four. This gives it approximately 50 MB on a 500 foot tape and 60 MB on a 600 foot tape. The recording area is narrower than the CS20 in order to accommodate the extra tracks, but the new tracks are interspersed between the four tracks used by the CS20. This allows the CS50 to read tapes created on an CS20. We have seen no problems with a CS20 reading the first four tracks of a tape written on an CS50, although the drive manufacturer does not guarantee this direction of compatibility. The CS20 and the CS50 both use the same type of cassette tape media.

PECULIARITIES

There are several peculiarities with the cassette tape drives. First, there are only two places data can be written: at beginning of tape (after rewinding), and at end of recorded data (after 'mt fseof'). It is not sufficient to use the 'mt fsf' command to move to the end of recorded data; 'mt fseof' must be used.

Users may notice that during a space forward operation (space backward is not supported) system operation may seem suspended. This is because the current system (by virtue of the device controller) does not support SCSI disconnect/reconnect.

FILES

/dev/ct0
/dev/rct0

SEE ALSO

intro(7), mt (1).

NAME

dcs – Distributed Communications Subsystem (DCS) driver

DESCRIPTION

The DCS provides from 8 to 128 asynchronous serial lines by interconnecting ICON DCS8, DCS9 and DCS16 cluster controllers to one or more DCS1 host adapter boards. The DCS9 cluster provides one Centronics parallel printer port in addition to eight serial lines.

The discussion of typewriter I/O given in *termio*(7) applies to DCS serial line ports with a few exceptions and limitations.

The special devices associated with *DCS* ports are named in the form,

/dev/tty*xyz*

where *x* is the DCS controller number (0-3), *y* is the DCS cluster controller address (0-f hexidecimal, set with two miniature rotary switches on each cluster controller), and *z* is the port number on the cluster (0-f hexidecimal, labeled 0-15 on the cluster controller). For example, the special file */dev/tty03a* refers to port number 10 (port a) on the cluster controller addressed at 3 and connected to the first DCS host adapter.

A special file of the form,

/dev/lp*xy*

is present for each possible parallel printer port in the DCS, where *x* is the DCS host adapter number (0-3) and *y* is the DCS9 cluster address (0-f hexidecimal).

A port must be "on-line" for an open to complete on it, which means that the cluster controller the port resides on is connected to the DCS coax and is powered-on and functional. DCS cluster controllers can be connected to or removed from a running system. If a cluster controller loses power, or is disconnected from the coax, the only effect is that any processes with ports open on it will receive a hang-up signal.

Data-Terminal-Ready/Data-Carrier-Detect modem control and Request-To-Send/Clear-To-Send handshaking are available on all serial ports, but are disabled by default. How to enable these signals is explained below.

CONFIGURATION

The DCS configuration, by default, allows the connection of any combination of DCS8, DCS9, and DCS16 cluster controllers, up to a maximum of 64 ports total. 128 minor devices are addressable in all; the devices that are active depends on the configuration.

The */etc/uxrc* variable **dcs*x*_config** specifies the basic default configuration for a DCS host adapter, where *x* is the DCS host adapter number. If this variable is set to 0, or is not present in */etc/uxrc*, DCS will allow any combination of DCS8, DCS9, and DCS16 cluster controllers, at all cluster addresses from 1 to f.

If the variable is set to 1, only DCS8 and DCS9 clusters may be used, with cluster addresses 1 through 8 reserved for DCS8 clusters, and 8 through f reserved for DCS9 clusters. This mode of configuration is intended to support the older DCS cluster controller firmware.

The *uxrc* variable **dcs*xy*_cctype** allows you to override the default configuration on a cluster-by-cluster basis, in either setting of the **dcs*x*_config** variable. Again, *x* is the DCS host adapter number, and *y* is the cluster controller address. Using this variable is necessary only with older cluster controllers.

The DCS driver also supports modem controls and/or hardware handshake protocols. The *uxrc* variables **dcs*xy*_modem** and **dcx*xy*_handshake** enable these functions on a port-by-port basis. See *uxrc(8)* for details.

For each port on which modem support is enabled, a second minor device for the port becomes usable. This second device entry is referred to as the "call-unit" side of the port, and is named with the prefix *cu*. The minor device number for the call-unit is formed by adding 128 to the minor device number of the "tty" device.

A special open protocol is implemented between the two "sides" allowing a modem to be used both as a login port for dialing in, and also as a outbound dialer for *uucico, kermit (1),* or other utilities. The basic protocol is as follows: (1) Only one side may be open a time. (2) Opens on the tty side will block until carrier is present, or until the call-unit is opened. (3) Opens on the call-unit side will complete in the absence of carrier, and will cause all waiting or subsequent open attempts on the tty side to return EBUSY. This allows communication with a modem to initiate a call. (4) When carrier drops, SIGHUP will be sent to the process group of the port, if any.

The first open call on either side causes DTR to be asserted on the port. The last close call on either side causes DTR to be negated if the HUP flag is set (see *ioctl*(2)).

FILES

/dev/dcs[0-3]	down-load special files	/dev/tty[0-3][1-f][0-f]	tty special files
/dev/cu[0-3][1-f][0-f]	call-unit special files		
/dev/lp[0-3][1-f]	line printer ports		

/etc/uxrc boot time configuration file

SEE ALSO

tty(7), uxrc(8), termio(7).

DIAGNOSTICS

"tty%d%x%x:overrun". The cluster controller's input buffer for the port was overrun before the host could retrieve it. Since the cluster controllers perform input flow control by using XON/XOFF and/or RTS/CTS flow control, this error only occurs when high baud-rate raw input is occurring on a port that is not using RTS/CTS handshaking.

BUGS

Certain ICON/UX line edit features normally present on PCP16 or Main CPU serial ports are not supported. For example, reprint line does not work, hardware tabs do not rub out properly, and control characters do not echo as they should.

NAME

flh, fld – ICON OMTI floppy disk interface

DESCRIPTION

The *flh, fld* devices provide access to an ICON OMTI floppy disk. The device uses 5 1/4-inch, double-sided floppy disks in high or double density format with 512 byte sectors.

ICON high density floppy disks contain 160 tracks, each with 15 sectors (for a total of 2400 sectors, 1228800 byte formatted capacity). ICON double density floppy disks contain 80 tracks, each with 9 sectors (for a total of 720 sectors, 368640 byte formatted capacity). ICON high density floppy disks (minor device 0) are compatible with IBM AT diskettes and double density ICON disks (minor device 1) are compatible with IBM PC diskettes.

In addition to normal i/o, the driver supports formatting of disks for either density.

Two jumpers on the TEAC FD-55GFV-25-U floppy disk must be moved to change between high and double density modes. Consult the Peripheral Options section of the reference manual for jumper settings.

One *ioctl*(2) call currently applies to the *fl* device. The *ioctl*(2) call has the form

```
#include <sys/iorequest.h>
ioctl(fildes, code, arg)
int *arg;
```

The applicable code is:

IOCFORMATDEV
: Format the diskette. The density to use is specified by the minor device associated with the *fildes* argument, minor device zero gives high density while 1 gives double density.

ERRORS

The following errors may be returned by the driver:

[EIO] Drive not ready; usually because no disk is in the drive or the drive door is open. A physical error other than "not ready" may also cause this error.

[EBUSY] Drive is in already open. The floppy driver allows only one open at a time.

FILES

/dev/flh (High Density)
/dev/fld (Double Density)

SEE ALSO

flfmt(1M), tar(1)

BUGS

Use of the flh, fld devices seriously degrades system performance on older hardware.

NAME

hs – HSMD disk interface

DESCRIPTION

The device names associated with the HSMD disk subsystem are of the form, **hs***cdp* where *c* is the controller number (0-7), *d* is the drive number (0-3), and *p* is the partition (a-h). For example: *hs01g* means partition "g" of drive number 1 on the first HSMD controller.

Only two major device numbers are used with HSMD, one for the block interface and one for the raw interface. The raw devices are named with the prefix "rhs". Accesses via the raw interface bypass the disk cache. Each controller uses 32 minor devices; 8 per drive.

DISK SUPPORT

The disk format utility, *dkfmt.hsmd*(1m), formats and labels HSMD disks. The label block written by dkfmt.hsmd describes the partitioning of the drive.

FILES

/dev/hs[0-7][0-3][a-h]	block files
/dev/rhs[0-7][0-3][a-h]	raw files

DIAGNOSTICS

Informational or warning messages:

hs%d: drive %d s=%d-%d-%d soft ecc. A data error was corrected. The controller, drive, cylinder, head, and sector numbers are displayed.

hs%d: drive %d (port %d, %d-%d-%d-%d) online. The specified drive is online. The controller, drive, and controller port numbers are shown along with the drive geometry.

hs%d: drive %d WARNING: confidence test failed, msg="%s". The specified drive failed the HSMD controller confidence test. The confidence test consists of a seek test, a write/read test, and an ECC correction test.

Fatal Error messages:

Fatal error messages from the HSMD subsystem are of the form,

hs%d%d: lsn%d <OPERATION>: err %04X: <messages>

The controller, drive number and logical sector number are displayed, as well as the requested operation, a hexidecimal error code, and symbolic messages describing the error bits.

hs%d%d: lsn%d <OPERATION>: err %04X: harderr. An unrecoverable error has occured. Exaustive retries and ECC correction failed.

hs%d%d: lsn%d <OPERATION>: err %04X: reject. This diagnostic should only occur during formatting if the drive is found to have too many defects.

hs%d%d: lsn%d <OPERATION>: err %04X: wr lock. A write request was attempted on a write locked drive.

NAME

is – integrated SCSI disks

DESCRIPTION

The device names associated with the integrated SCSI disks are of the form, **/dev/is[0-2]p** where p is the partition (a-h). For example: **/dev/is1g** is associated with partition "g" of drive number 1. A maximum of 3 disks may be configured in this manner.

The raw devices are named with the prefix "ris". For example: **/dev/ris1g** selects the raw device corresponding to partition "g" of drive number 1.

SOFTWARE SUPPORT

The disk format utility, *dkfmt*(1A), formats and labels integrated SCSI disks. The label block written by dkfmt describes the partitioning of the drive. The new file system utility, *newfs*(1A), creates a new file system on the partition. The default sizes for these partitions are contained in /etc/fstab.

GENERAL INFORMATION

For more information on configuring drives, consult the *System Reference Manual* for your machine.

ICON HD180/CDC 94161 "Wren III"

Disk Controller Type	
Sectors per Track	18
Number of Heads	9
Number of Cylinders	967
Cylinder Size	162
Capacity	180 Mbytes

ICON HD180/CDC 94161 with all three Partitions (default)

	Partition Description	Start	Length	End
	Useable Disk	90	156564	156653
	Remapping Tracks	0	0	0
a	Root Partition	162	7942	8103
b	Swap Partition	8262	15876	24137
c	Disk Minus Remap	0	156654	156653
g	Main Partition	24138	132516	156653

ICON HD350/CDC 94171 "Wren IV"

Disk Controller Type	Integrated SCSI
Sectors per Track	20 (variable)
Number of Heads	11
Number of Cylinders	1365
Cylinder Size	220
Capacity	350 Mbytes

ICON HD350/CDC 94171 with all three Partitions (default)

Partition	Description	Start	Length	End
	Useable Disk	90	300210	300299
	Remapping Tracks	0	0	0
a	Root Partition	90	7942	8031
b	Swap Partition	8032	15884	23915
c	Disk Minus Remap	0	300300	300299
g	Main Partition	23916	276384	300299

ICON HD180/Toshiba MK156

ICON HD180/Toshiba MK156 General Information

Disk Controller Type	Integrated SCSI
Sectors per Track	18
Number of Heads	10
Number of Cylinders	825
Defects per Surface	n/a
Variable Sectors per Track:	no
Cylinder Size	180
Capacity	180 Mbytes

ICON HD180/Toshiba MK156 with all three Partitions (default)

Partition	Description	Start	Length	End
	Usable Disk	90	148410	148499
	Remapping Tracks	0	0	0
a	Root Partition	180	7942	8121
b	Swap Partition	8280	15840	24119
c	Disk Minus Remap	0	148500	148499
g	Main Partition	24120	124380	148499

FILES

/dev/is[0-2][a-h]	block files
/dev/ris[0-2][a-h]	raw files

SEE ALSO

newfs (1M), dkfmt (1M), fstab (4), fsck (1M)

NAME

klog – kernel logging device

DESCRIPTION

Klog is a special file that allows log messages generated within the UNIX system kernel to be passed to a user program. This device is generally used by ICON/UXB utilities.

FILES

/dev/klog

NAME

lp – line printer

DESCRIPTION

The special device **/dev/lp** provides an interface to any of the standard Centronics-compatible parallel printers.

When opened or closed, a form feed is generated. In half-ASCII mode, lower case letters are turned into upper case, and certain characters are escaped according to the following table:

{	(
}	)
‘	\
|	!
~	^

The driver correctly interprets carriage returns, backspaces, tabs, and form-feeds. A new-line that extends over the end of a page is turned into a form-feed.

Several different options are available which modify the behavior of the driver. If the device is opened with the FNDELAY bit on in the mode field, the device will perform non-blocking I/O. The device **/dev/rlp** does not perform any output processing and is useful for printers which expect non-ascii characters. The device **/dev/nfflp** performs all the normal output processing except that it does not output a form-feed on opens and closes.

FILES

/dev/lp, /dev/rlp, /dev/nfflp,

SEE ALSO

lpr(1).

NAME

dmem, kmem, mba — main memory

DESCRIPTION

Dmem is a special file that is an image of the disk cache processors physical memory. **Kmem** is a special file that is an image of the main memory of the computer. **Mba** is a special file that is an image of multibus adaptor cards. These devices may be used to download into system memory, to examine (and even to patch) the system.

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

Kernel virtual addresses range from 0x40000000 to 0xffffffff. Disk cache addresses are in the range of zero to the amount of disk cache memory on the system. Accesses to **/dev/mba** are performed one byte at a time to avoid byte swapping problems on multibus cards.

FILES

/dev/dmem
/dev/kmem
/dev/mba

BUGS

Examining and patching device registers is likely to lead to unexpected results.

NAME

mot — MC68681 asynchronous interface

DESCRIPTION

The discussion of typewriter I/O given in *termio*(7) applies to these devices. Modem control and RTS/CTS hadshaking may also be set up using the **/etc/uxrc** file.

FILES

/dev/console, /dev/tty01,

SEE ALSO

init(1M), tty(7), termio(7), uxrc(7) .

NAME

mt – 9–Track ½–Inch Tape Drive

DESCRIPTION

The file **/dev/mt0** refers to the normal ICON 9–track tape interface. This device interfaces to a streaming half inch nine track tape drive which is compatible with industry standard 1600 bpi PE format tapes. The drive can write/read blocks up to 48000 bytes long, at either 25 inches per second or 100 inches per second. The size of the tape blocks for *write* system calls is determined by the length passed. For reads, if the buffer length is greater than or equal to the size of the record read, the entire record is passed and the number of bytes actually read is returned. If the buffer is smaller than the block from tape, the record is truncated and the excess data will be lost; the next *read* will result in another physical I/O. When opened for reading or writing, the tape is assumed to be positioned as desired. When the device is closed after writing, two file marks are written at the end of the data. After closing, the tape is rewound. The EOF is returned as a zero-length read. Seeks are ignored.

The file **/dev/rmt0** refers to the non-rewind on close ICON 9–track tape interface. This device is exactly like **/dev/mt0** except that the rewind on close is suppressed and the tape is positioned before the second EOF mark.

The file **/dev/hmt0** is exactly like /dev/rmt0 except that the tape is operated at 100 ips (high speed). This mode is especially useful for spacing operations or if the data blocks are long.

OTHER POSSIBLE DEVICES

The major device number for the driver is (in decimal) 12, and the minor device number is encoded from the following bit string:

zzrdduuu

where *zz* is always zero, *r* is a no-rewind-on-close flag, *dd* is the density and transport speed selector, and *uuu* is the drive unit. Currently, *uuu* must be zero, as only one MT16 tape drive per system is supported. The following table shows the meaning of the different density (*dd*) values:

MT16 Density Options

Value of "*dd*"	Tape Density	Transport Speed
00	800 bpi	25 ips
01	1600 bpi	25 ips
10	3200 bpi	25 ips
11	1600 bpi	100 ips

For example, the minor device number for an MT16 using 800 bpi tapes with no rewind-on-close would be (in binary) 00100000, or (in decimal) 32. New device entries can be made using *mknod(8)* once the major and minor device numbers have been

ascertained.

FILES

/dev/mt0
/dev/rmt0
/dev/hmt0

SEE ALSO

intro(7), mt (1).

NAME

mtty – Multi-link terminal driver

DESCRIPTION

The *mtty* driver provides a link between the ICON PROC/286 board and a UNIX terminal. Through the PROC/286 processor and the Multi-link software package, up to four UNIX system users can simultaneously access the MPS/DOS operating system.

FILES

/dev/mtty[0-7]

SEE ALSO

SYSTEM REFERENCE MANUAL.

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

pcp – Peripheral Communication Processor

DESCRIPTION

The Peripheral Communication Processor (PCP/16) is a port expansion board in the **ICON** Architecture. The discussion of typewriter I/O given in *termio*(7) applies to serial ports on the PCP.

Each PCP/16 option provides the **ICON** system with 16 serial communications ports and one Centronics-compatible parallel printer port. Of the sixteen serial ports, two are selectable for RS-232C or RS-422 mode, asynchronous or synchronous protocols; two are RS-232C/RS-422 selectable, asynchronous only, and the remaining 12 ports are RS-232C asynchronous only. The device names associated with these PCP serial ports are of the form **/dev/tty[a-d][0-9,a-f]**. For example **/dev/ttya2** refers to the port 2 on the first PCP board.

Each PCP board has a parallel port which functions in the same manner as **/dev/lp**. The device names associated with these PCP parallel ports are of the form **/dev/lp[a-d]**. For example **/dev/lpb**, **/dev/rlpb** and **/dev/nfflpb** refer to the parallel port on the second PCP board.

The RTS/CTS handshaking and modem control are configurable through **/etc/uxrc**. The software that the PCP runs is contained in **/etc/pcpimage** and is downloaded with the program **/etc/loadpcp**.

FILES

/dev/tty[a-d][0-9,a-f] /dev/lp[a-d], /dev/rlp[a-d], /dev/nfflp[a-d], /pcpimage

SEE ALSO

tty(7), termio(7), uxrc(8), loadpcp(1a), lp(7).

DIAGNOSTICS

PCP #%d not responding .

The PCP is not running properly. This may have happened because the program **/etc/loadpcp** failed for some reason. There may have been a corrupted **pcpimage** or there may be hardware problems with the PCP board.

NAME

pty — pseudo terminal driver

SYNOPSIS

pseudo-device pty [count]

DESCRIPTION

The *pty* driver provides support for a device-pair termed a *pseudo terminal*. A pseudo terminal is a pair of character devices, a *master* device and a *slave* device. The slave device provides processes an interface identical to that provided by a physical device. However, whereas all other devices which provide this kind of interface have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

In configuring, if an optional "count" is given in the specification, that number of pseudo terminal pairs are configured; the default count is 32.

Source level support for pseudo terminals is provided through ICON/UXB releases.

FILES

/dev/pty[p-r][0-9a-f]	master pseudo terminals
/dev/tty[p-r][0-9a-f]	slave pseudo terminals

DIAGNOSTICS

None.

NAME

qic – Quarter-Inch Cartridge Tape Drive

DESCRIPTION

This drive supports both the QIC-11 and QIC-24 quarter inch cartridge standards. These cartridges store up to 60 MB.

The file **/dev/qic24** refers to the MTS-1 QIC drive in QIC-24 format. When opened for reading or writing, the tape is assumed to be positioned as desired. Data written to this device is organized in 512 byte blocks, and ICON/UXV buffers 128 of these blocks per physical write. Therefore, the optimal block size for reads and writes is 64K bytes. If the length of the data to be written is not a multiple of 512 when the device is closed, the last block is padded with zero bytes before it is written. When the device is closed after writing, one file mark is written at the end of the data. After closing, the tape is rewound. The EOF is returned as a zero-length read. Seeks are ignored.

The file **/dev/rqic24** is the non-rewind on close version. This device is exactly like **/dev/qic24** except that the rewind on close is suppressed.

The files **/dev/qic11** and **/dev/rqic11** can be used to read and write QIC-11 format tapes. If it is necessary to read a tape when the format is unknown, try QIC-24 first, then if that does not appear to work, try QIC-11.

FILES

/dev/qic24
/dev/rqic24
/dev/qic11
/dev/rqic11

SEE ALSO

intro(7), mt (1).

NAME

sc – ST-506 on OMTI SCSI

DESCRIPTION

The device names associated with the ST-506 SCSI disks are of the form, **/dev/sc[0-1]p** where p is the partition (a-h). For example: **/dev/sc1g** is associated with partition "g" of drive number 1. A maximum of 3 disks may be configured in this manner.

The raw devices are named with the prefix "rsc". For example: **/dev/rsc1g** selects the raw device corresponding to partition "g" of drive number 1. Up to 2 ST-506 disks may be attached to the OMTI controller board.

SOFTWARE SUPPORT

The disk format utility, *dkfmt*(1M), formats and labels ST-506 SCSI disks. The label block written by dkfmt describes the partitioning of the drive. The new file system utility, *newfs*(1M), creates a new file system on the partition. The default sizes for these partitions are contained in /etc/fstab.

GENERAL INFORMATION

For more information on configuring drives, consult the *System Reference Manual* for your machine.

ICON HD190/Maxtor XT-2190

ICON HD190/Maxtor XT-2190 General Information

Disk Controller Type	ST-506 on OMTI SCSI
Sectors per Track	9
Number of Heads	15
Number of Cylinders	1224
Defects per Surface	13
Cylinder Size	135
Capacity	190 Mbytes

ICON HD190/Maxtor XT-2190 with all three Partitions (default)

	Partition Description	Start	Length	End
	Useable Disk	90	163395	163484
	Remapping Tracks	163485	1755	165239
a	Root Partition	135	7942	8076
b	Swap Partition	8100	15795	23894
c	Disk Minus Remap	0	163485	163484
g	Main Partition	23895	139590	163484

ICON HD80/Toshiba MK56

ICON HD80/Toshiba MK56 General Information

Disk Controller Type	ST-506 on OMTI SCSI
Sectors per Track	9
Number of Heads	10
Number of Cylinders	830
Defects per Surface	10
Cylinder Size	90
Capacity	80 Mbytes

ICON HD80/Toshiba MK56 with all three Partitions (default)

	Partition Description	Start	Length	End
	Useable Disk	90	73710	73799
	Remapping Tracks	73800	900	74699
a	Root Partition	90	7942	8031
b	Swap Partition	8100	7920	16019
c	Disk Minus Remap	0	73800	73799
g	Main Partition	16020	57780	73799

ICON HD80/Fujitsu M2243

ICON HD80/Fujitsu M2243 General Information

Disk Controller Type	ST-506 on OMTI SCSI
Sectors per Track	9
Number of Heads	11
Number of Cylinders	754
Defects per Surface	18
Cylinder Size	99
Capacity	80 Mbytes

ICON HD80/Fujitsu M2243 with all three Partitions (default)

	Partition Description	Start	Length	End
	Usable Disk	90	72774	72863
	Remapping Tracks	72864	1782	74645
a	Root Partition	99	7942	8040
b	Swap Partition	8118	7920	16037
c	Disk Minus Remap	0	72864	72863
g	Main Partition	16038	52826	72863

ICON HD80/MiniScribe

ICON HD80/MiniScribe General Information

Disk Controller Type	ST-506 on OMTI SCSI
Sectors per Track	9
Number of Heads	8
Number of Cylinders	1024
Defects per Surface	10
Cylinder Size	72
Capacity	80 Mbytes

ICON HD80/MiniScribe with all three Partitions (default)

Partition	Description	Start	Length	End
	Usable Disk	90	72918	73007
	Remapping Tracks	73008	720	73727
a	Root Partition	144	7942	8085
b	Swap Partition	8136	7920	16055
c	Disk Minus Remap	0	73008	73007
g	Main Partition	16056	56952	73007

FILES

/dev/sc[0-1][a-h]	block files
/dev/rsc[0-1][a-h]	raw files

SEE ALSO

newfs (1M), dkfmt (1M), fstab (4), fsck (1M)

NAME

sfi — SMILE file interface

DESCRIPTION

SMILE allows PCs, XTs, ATs, compatibles and 386 computers with an AT-bus to use resources on the ICON host computer. The SMILE file interface provides Virtual Disk functionality between DOS and ICON/UX.

Two types of disk I/O support are provided under SMILE. The first is support for logical DOS drives accessible to the PC via SMILE. The second type is a file transfer interface to the ICON/UX file system.

Logical DOS Drives In order to access logical DOS Disks on the ICON host through the SMILE interface, a DOS device driver (SMDISK.DEV) must be loaded in the target computer at boot time. This device driver assigns logical drives which are mapped to files in the ICON/UX file system. These logical DOS drives (or files) are defined when the ICON host is booted by the information in the ICON/UX file */etc/smiledisks*. Each port has an */etc/smiledisks_0X* file which shows its particular drive mappings and read/write privilidges, which are communicated to the device driver on boot up.

When a disk access is made this driver writes a message in a buffer in the 64K of dual-port RAM on the SMILE Target card and interrupts the Icon DCP to request service. Upon being interrupted the ICON DCP deciphers which SMILE port interrupted it and looks in the appropriate dual-port memory for a message of which sectors to retrieve from disk. It retrieves the requested data and writes it to the dual-port memory of the SMILE Target requesting such. The DCP also looks ahead and stores additional sectors in its cache memory, so that on the following request the data is in memory rather than on the disk. The target computer is polling its dual-port SMILE memory and when the data arrives it completes its "*disk access*".

Note that the current software release only allows a logical drive to be read/write on one target computer at any given moment. However, all target computers may have read-only access to any logical drive if they are so configured. File Transfer

The second type of disk I/O supported is a file transfer interface to the ICON/UX file system. This interface is used by the UCOPY and TAR utilities in DOS.

Using this interface UCOPY provides the ability to transfer files between DOS and ICON/UX file system. UCOPY writes to the SMILE memory and interrupts the Icon DCP. A smilecopyd deamon monitors the interrupts and invokes the ICON/UX driver when a request for a file transfer is made. This utility allows UNIX files to be read and written from the DOS environment.

TAR is a DOS utility that opens a tape device in the ICON host computer and backs up DOS files to that tape device. It uses the same interface as UCOPY and is

serviced by the same smilecopyd daemon. TAR is similar to the tar utility in ICON/UX.

FILES

/dev/sfi0[0-5]

SEE ALSO

smilecopyd(1m), smiledisk(1m), SMILE USERS MANUAL.

NAME

ssi – IBM 3274 emulation

DESCRIPTION

The devices configured for 3274 emulation provide download, control and terminal support to allow normal terminals to emulate the IBM 3274 protocol. The actual connection is made through a multibus adaptor board.

FILES

/dev/mb_[0-9,a-f], /dev/DOWNLOAD, /dev/CONTROL.

SEE ALSO

mba(7).

NAME

sti — SMILE terminal interface

DESCRIPTION

SMILE allows PCs, XTs, ATs, compatibles and 386 computers with an AT-bus to use resources on the ICON host computer.

Terminal Devices The SMILE terminal interface allows the target computers connected through the SMILE interface to open Unix sessions with the ICON host. Provided with the SMILE target card is a terminal emulator, TERM.EXE, that communicates with the terminal device driver on the ICON host computer, thus allowing a target computer to act as a normal ICON/UX login terminal. Each target computer can support up to 15 simultaneous terminal device connections to ICON/UX through the packet queues.

Print Spooler Printers on the ICON host are also available through this character device connection. One channel of character I/O is reserved for redirecting target computer print output to the ICON/UX print spooler. DOS printers are redirected through the SMDRV.DEV driver and sent to channel 0 of the character interface. The smileprint deamon in ICON/UX monitors channel 0 of the character device connections and redirects print data to the ICON/UX spooler to be printed on the specified printer.

The device names will have the form **/dev/ttys[0-5][0-9,a-f]** where the first number refers to the SMILE port. For example **/dev/ttys02** refers to the second port on SMILE port number 0. The discussion of typewriter I/O given in *termio*(7) applies to these devices.

FILES

/dev/ttys[0-5][0-9,a-f]

SEE ALSO

smileprint(1m), termio(7), SMILE USERS MANUAL.

NAME

sxt — pseudo-device driver

DESCRIPTION

Sxt is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the *sxt* driver. *Sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The *sxt* driver is currently only used by the *shl*(1) command. Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing. Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked. There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL Set *exclusive use* mode: no further opens are permitted until the file has been closed.

TIOCNXCL Reset *exclusive use* mode: further opens are once again permitted.

The second group are directives to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

- EINVAL The argument is out of range.
- ENOTTY The command was not issued from a real tty.
- ENXIO *linesw* is not configured with *sxt*.
- EBUSY An SXTIOCLINK command has already been issued for this real *tty*.
- ENOMEM There is no system memory available for allocating the virtual tty structures.
- EBADF Channel 0 was not opened before this call.

SXTIOCSWTCH Set the controlling channel. Possible errors include:

- EINVAL An invalid channel number was given.
- EPERM The command was not executed from channel 0.

SXTIOCWF	Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL*, if an invalid channel number is given.
SXTIOCUBLK	Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.
SXTIOCSTAT	Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.
SXTIOCTRACE	Enable tracing. Tracing information is written to */dev/osm* on the 3B 20 computer or to the console on the VAX. This command has no effect if tracing is not configured.
SXTIOCNOTRACE	Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7]	Virtual tty devices
/usr/include/sys/sxt.h	Driver specific definitions.

SEE ALSO

shl(1), stty(1), ioctl(2), open(2), termio(7).

NAME

termio — general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. 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 keystroke 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.

SWTCH (Control-z or ASCII SUB) is used by the job control facility, *shl,* to change the current layer to the control layer.

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

When the carrier signal from the data-set drops, a *hang-up* 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 hang-up 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	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	SWTCH	

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.
LOBLK	0010000	Block layer output.

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.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

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:

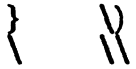
ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable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

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, 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 between characters. 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, switch, and interrupt characters will not be done.

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

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

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

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

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

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

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1), fork(2), ioctl(2), setpgrp(2), signal(2).

NAME

tty — controlling terminal interface

DESCRIPTION

The file **/dev/tty** is, 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

termio(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, adding new user accounts, etc.

BUGS

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

NAME

adduser — procedure for adding new users

DESCRIPTION

A new user must choose a login name, which must not already appear in */etc/passwd*. An account can be added by editing a line into the passwd file; this can be done with the editor *vi(1)*.

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. Thus at a university, for example, you might have groups for system staff, faculty, graduate students, and a few special groups for large projects.

A skeletal account for a new user "ernie" would look like:

ernie::235:20:Kovacs:/mnt/grad/ernie:/bin/csh

The first field is the login name "ernie". The next field is the encrypted password which is not given and must be initialized using *passwd*(1). The next two fields are the user and group id's. Traditionally, users are in group 30.

The next field is the name field. It may contain installation specific information as well.

The final two fields give a login directory and a login shell name.

The login shell will default to "/bin/sh" if none is given.

FILES

/etc/passwd	password file

SEE ALSO

passwd(1), passwd(4)

NAME

crash — what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes.

When the system crashes voluntarily it prints a message of the form

panic: why i gave up the ghost

on the console, then sits there waiting for someone to hit reset. When reset is hit, by turning the system keyswitch to reset, the bootstrap loader starts executing reboot. Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure the system will then resume multi-user operations, if so configured.

The system has a large number of internal consistency checks; if one of these fails, then it will panic with a very short message indicating which one failed.

There are two basic types of system failures, those resulting from hardware failure, and those resulting from internal kernel inconsistency. Hardware errors are the most common (which is not to say that they are common), and can manifest themselves in a number of different ways.

System crashes should be rare occurences. Regular or frequent crashes could indicate a persistent hardware problem. Contact your customer service representative in such instances.

SEE ALSO

reboot(1M)

NAME

standalone — definition of standalone operation mode

DESCRIPTION

Standalone mode is a special single user mode used at initial system configuration time, or subsequently to run special standalone programs, e.g. format a floppy, or check the integrity of the file system.

Standalone mode centers around the loader, bload(1M), and is entered by overriding the automatic boot procedure, when prompted. You will be asked to choose a boot device:

```
Device Name
    sc0     SCSI Controller Hard Disk
    is0     Integrated SCSI Hard Disk
    ct0     Cassette Tape
    qic24   Quarter Inch Cartridge Tape
    mt0     Half Inch Magnetic Tape
Enter boot device name
```

Normally you will choose '0',to boot from the system's default hard disk drive. The loader program will then start running and issue the prompt:

```
Boot loader -- Version 3.20

Load: □
```

To run a standalone program you would type:

>stand/(program name) [options]

The '>' tells the loader that this is a standalone program. The loader assumes that the base directory is called "/". Thus "stand/(program name)" is a program in the directory "stand", which is in the directory "/".

SEE ALSO

bload(1M), binstl(1M), copy(1M), dkfmt(1M), fsck(1M), mkfs(1M), park(1M)

NAME

uxrc – ICON/UXB run-time configuration file

DESCRIPTION

Uxrc is the configuration file which is used to set kernel configuration variables at boot time.

When a reboot is in progress, */etc/uxrc* is read to change the variables from their default state. These kernel variables can be changed by creating a file called */etc/uxrc* and adding the appropriate configuration statements.

Using the *uxrc* file, the system administrator can control serial port characteristics and several other features. For a complete list of options, and a description of the file format, see *uxrc*(4).

SEE ALSO

uxrc(4)

Printed in the U.S.A. **171-063-002 A1**