-

NAME

ac − login accounting

SYNOPSIS

ac [−**w** wtmp] [−**p**] [−**d**] people

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. −**w** is used to specify an alternate *wtmp* file. −**p** prints individual totals; without this option, only totals are printed. −**d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login.* Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init (VIII), login (I), wtmp (V).

BUGS

-

NAME

boot procedures − UNIX startup

DESCRIPTION

How to start UNIX. UNIX is started by placing it in core at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape.

The *tp* command places a bootstrap program on the otherwise unused block zero of the tape. The DECtape version of this program is called *tboot,* the magtape version *mboot.* If *tboot* or *mboot* is read into location zero and executed there, it will type '=' on the console, read in a *tp* entry name, load that entry into core, and transfer to zero. Thus one way to run UNIX is to maintain the system code on a tape using *tp.* Caution: the file /usr/mdec/tboot (DECtape) or /usr/mdec/mboot (magtape) must be present when the tape is made! When a boot is required, execute (somehow) a program which reads in and jumps to the first block of the tape. In response to the '=' prompt, type the entry name of the system on the tape (we use plain 'unix'). It is strongly recommended that a current version of the system be maintained in this way, even if it is usually booted from disk.

The standard DEC ROM which loads DECtape is sufficient to read in *tboot,* but the magtape ROM loads block one, not zero. If no suitable ROM is available, magtape and DECtape programs are presented below which may be manually placed in core and executed.

The system can also be booted from a disk file with the aid of the *uboot* program. When read into location 0 and executed, *uboot* reads a single character (either **p** or **k** for RP or RK, both drive 0) to specify which device is to be searched. Then it reads a UNIX pathname from the console, finds the corresponding file on the given device, loads that file into core location zero, and transfers to it. *Uboot* operates under very severe space constraints. It supplies no prompts, except that it echoes a carriage return and line feed after the **p** or **k.** No diagnostic is provided if the indicated file cannot be found, nor is there any means of correcting typographical errors in the file name except to start the program over. If it fails to find the file, however, it jumps back to its start, so another try can be attempted, starting again with the **p** or **k.** Notice that *uboot* will only load a file from drive 0, and the file system it searches must start at the beginning of the disk. *Uboot* itself usually resides in the otherwise unused block 0 of the disk, so it can be loaded by ROM program; *mkfs* can be used to put it there when the file system is created. It can also be loaded from a *tp* tape as described above.

The switches. The console switches play an important role in the use and especially the booting of UNIX. During operation, the console switches are examined 60 times per second, and the contents of the address specified by the switches are displayed in the display register. (This is not true on the 11/40 since there is no display register on that machine.) If the switch address is even, the address is interpreted in kernel (system) space; if odd, the rounded-down address is interpreted in the current user space.

If any diagnostics are produced by the system, they are printed on the console only if the switches are non-zero. Thus it is wise to have a non-zero value in the switches at all times.

During the startup of the system, the *init* program (VIII) reads the switches and will come up single-user if the switches are set to 173030.

It is unwise to have a non-existent address in the switches. This causes a bus error in the system (displayed as 177777) at the rate of 60 times per second. If there is a transfer of more than 16ms duration on a device with a data rate faster than the bus error timeout (about 10μs) then a permanent disk non-existent-memory error will occur.

ROM programs. Here are some programs which are suitable for installing in read-only memories, or for manual keying into core if no ROM is present. Each program is position-independent but should be placed well above location 0 so it will not be overwritten. Each reads a block from the beginning of a device into core location zero. The octal words constituting the program are listed on the left.

-

DECtape (drive 0) from endzone:

```
012700        mov    $tcba,r0
177346
010040        mov    r0,−(r0)        / use tc addr for wc
012710        mov    $3,(r0)         / read bn forward
000003
105710   1:   tstb   (r0)            / wait for ready
002376        bge    1b
112710        movb   $5,(r0)         / read (forward)
000005
000777        br     .               / loop; now halt and start at 0
```

DECtape (drive 0) with search:

```
012700   1:   mov    $tcba,r0
177346
010040        mov    r0,−(r0)        / use tc addr for wc
012740        mov    $4003,−(r0)     / read bn reverse
004003
005710   2:   tst    (r0)
002376        bge    2b              / wait for error
005760        tst    −2(r0)          / loop if not end zone
177776
002365        bge    1b
012710        mov    $3,(r0)         / read bn forward
000003
105710   2:   tstb   (r0)            / wait for ready
002376        bge    2b
112710        movb   $5,(r0)         / read (forward)
000005
105710   2:   tstb   (r0)            / wait for ready
002376        bge    2b
005007        clr    pc              / transfer to zero
```

Caution: both of these DECtape programs will (literally) blow a fuse if 2 drives are dialed to zero.

Magtape from load point:

```
012700        mov    $mtcma,r0
172526
010040        mov    r0,−(r0)        / usr mt addr for wc
012740        mov    $60003,−(r0)    / read 9−track
060003
000777        br     .               / loop; now halt and start at 0
```

RK (drive 0):

```
012700        mov    $rkda,r0
177412
005040        clr    −(r0)           / rkda cleared by start
010040        mov    r0,−(r0)
012740        mov    $5,−(r0)
000005
105710   1:   tstb   (r0)
002376        bge    1b
005007        clr    pc
```

-

RP (drive 0)

```
012700          mov     $rpmr,r0
176726
005040          clr     -(r0)
005040          clr     -(r0)
005040          clr     -(r0)
010040          mov     r0,-(r0)
012740          mov     $5,-(r0)
000005
105710    1:    tstb    (r0)
002376          bge     1b
005007          clr     pc
```

FILES

/unix − UNIX code
/usr/mdec/mboot − *tp* magtape bootstrap
/usr/mdec/tboot − *tp* DECtape bootstrap
/usr/mdec/uboot − file system bootstrap

SEE ALSO

tp (I), init (VIII)

NAME

chgrp − change group

SYNOPSIS

chgrp group file ...

DESCRIPTION

The group-ID of the files is changed to *group.* The group may be either a decimal GID or a group name found in the group-ID file.

Only the super-user is allowed to change the group of a file, in order to simplify as yet unimplemented accounting procedures.

SEE ALSO

chown (VIII)

FILES

/etc/group

BUGS

NAME

chown − change owner

SYNOPSIS

chown owner file ...

DESCRIPTION

The user-ID of the files is changed to *owner.* The owner may be either a decimal UID or a login name found in the password file.

Only the super-user is allowed to change the owner of a file, in order to simplify as yet unimplemented accounting procedures.

FILES

/etc/passwd

SEE ALSO

chgrp (VIII)

BUGS

NAME

clri − clear i-node

SYNOPSIS

clri i-number [filesystem]

DESCRIPTION

Clri writes zeros on the 32 bytes occupied by the i-node numbered *i-number.* If the *file system* argument is given, the i-node resides on the given device, otherwise on a default file system. The file system argument must be a special file name referring to a device containing a file system. After *clri,* any blocks in the affected file will show up as ‘‘missing’’ in an *icheck* of of the file system.

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.

BUGS

Whatever the default file system is, it is likely to be wrong. Specify the file system explicitly.

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

-

NAME

crash − what to do when the system crashes

DESCRIPTION

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

How to bring it back up. If the reason for the crash is not evident (see below for guidance on 'evident') you may want to try to dump the system if you feel up to debugging. At the moment a dump can be taken only on magtape. With a tape mounted and ready, stop the machine, load address 44, and start. This should write a copy of all of core on the tape with an EOF mark. Caution: Any error is taken to mean the end of core has been reached. This means that you must be sure the ring is in, the tape is ready, and the tape is clean and new. If the dump fails, you can try again, but some of the registers will be lost. See below for what to do with the tape.

In restarting after a crash, always bring up the system single-user. This is accomplished by following the directions in *boot procedures* (VIII) as modified for your particular installation; a single-user system is indicated by having a particular value in the switches (173030 unless you've changed *init)* as the system starts executing. When it is running, perform a *dcheck* and *icheck* (VIII) on all file systems which could have been in use at the time of the crash. If any serious file system problems are found, they should be repaired. When you are satisfied with the health of your disks, check and set the date if necessary, then come up multi-user. This is most easily accomplished by changing the single-user value in the switches to something else, then logging out by typing an EOT.

To even boot UNIX at all, three files (and the directories leading to them) must be intact. First, the initialization program */etc/init* must be present and executable. If it is not, the CPU will loop in user mode at location 6. For *init* to work correctly, */dev/tty8* and */bin/sh* must be present. If either does not exist, the symptom is best described as thrashing. *Init* will go into a *fork/exec* loop trying to create a Shell with proper standard input and output.

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

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

The problems reported by *icheck* typically fall into two kinds. There can be problems with the free list: duplicates in the free list, or free blocks also in files. These can be cured easily with an *icheck −s.* If the same block appears in more than one file or if a file contains bad blocks, the files should be deleted, and the free list reconstructed. The best way to delete such a file is to use *clri* (VIII), then remove its directory entries. If any of the affected files is really precious, you can try to copy it to another device first.

Dcheck may report files which have more directory entries than links. Such situations are potentially dangerous; *clri* discusses a special case of the problem. All the directory entries for the file should be removed. If on the other hand there are more links than directory entries, there is no danger of spreading infection, but merely some disk space that is lost for use. It is sufficient to copy the file (if it has any entries and is useful) then use *clri* on its inode and remove any directory entries that do exist.

Finally, there may be inodes reported by *dcheck* that have 0 links and 0 entries. These occur on the root device when the system is stopped with pipes open, and on other file systems when the system stops with files that have been deleted while still open. A *clri* will free the inode, and an *icheck -s* will recover any missing blocks.

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

-

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

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

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

out of inodes
A mounted file system has no more i-nodes when creating a file. Sorry, the device isn't available; the *icheck* should tell you.

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

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

no inodes
The in-core inode table is full. Try increasing NINODE in param.h. Shouldn't be a panic, just a user error.

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

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

unlink – iget
The directory containing a file being deleted can't be found. Hardware or software.

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

out of text
A pure procedure program is being executed, and the table for such things is full. This shouldn't be a panic.

trap
An unexpected trap has occurred within the system. This is accompanied by three numbers: a 'ka6', which is the contents of the segmentation register for the area in which the system's stack is kept; 'aps', which is the location where the hardware stored the program status word during the trap; and a 'trap type' which encodes which trap occurred. The trap types are:

0	bus error
1	illegal instruction
2	BPT/trace
3	IOT
4	power fail
5	EMT
6	recursive system call (TRAP instruction)
7	11/70 cache parity, or programmed interrupt
10	floating point trap
11	segmentation violation

In some of these cases it is possible for octal 20 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred. If you wish to examine the stack after such a trap, either dump the system, or use the console switches to examine core; the required address mapping is described below.

Interpreting dumps. All file system problems should be taken care of before attempting to look at dumps. The dump should be read into the file */usr/sys/core; cp* (I) will do. At this point, you should execute *ps −alxk* and *who* to print the process table and the users who were on at the time of the crash. You should dump (*od* (I)) the first 30 bytes of */usr/sys/core.* Starting at location 4, the registers R0, R1, R2, R3, R4, R5, SP and KDSA6 (KISA6 for 11/40s) are stored. If the dump had to be restarted, R0 will not be correct. Next, take the value of KA6 (location 22(8) in the dump) multiplied by 100(8) and dump 1000(8) bytes starting from there. This is the per-process data associated with the process running at the time of the crash. Relabel the addresses 140000 to 141776. R5 is C's frame or display pointer. Stored at (R5) is the old R5 pointing to the previous stack frame. At (R5)+2 is the saved PC of the calling procedure. Trace this calling chain until you obtain an R5 value of 141756, which is where the user's R5 is stored. If the chain is broken, you have to look for a plausible R5, PC pair and continue from there. Each PC should be looked up in the system's name list using *db* (I) and its ':' command, to get a reverse calling order. In most cases this procedure will give an idea of what is wrong. A more complete discussion of system debugging is impossible here.

SEE ALSO

clri, icheck, dcheck, boot procedures (VIII)

BUGS

-

NAME

cron − clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

Cron executes commands at specified dates and times according to the instructions in the file /usr/lib/crontab. 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* (VIII).

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Crontab is examined by *cron* every hour. Thus it could take up to an hour for entries to become effective. If it receives a hangup signal, however, the table is examined immediately; so 'kill −1 ...' can be used.

FILES

/usr/lib/crontab

SEE ALSO

init(VIII), sh(I), kill (I)

DIAGNOSTICS

None − illegal lines in crontab are ignored.

BUGS

A more efficient algorithm could be used. The overhead in running *cron* is about one percent of the machine, exclusive of any commands executed.

-

NAME

dcheck − file system directory consistency check

SYNOPSIS

dcheck [**−i** numbers] [filesystem]

DESCRIPTION

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

Currently, /dev/rrk2 and /dev/rrp0 are the default file systems.

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.

SEE ALSO

icheck (VIII), fs (V), clri (VIII), ncheck (VIII)

BUGS

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

NAME

df − disk free

SYNOPSIS

df [filesystem]

DESCRIPTION

Df prints out the number of free blocks available on a file system. If the file system is unspecified, the free space on all of the normally mounted file systems is printed.

FILES

/dev/rf?, /dev/rk?, /dev/rp?

SEE ALSO

icheck (VIII)

BUGS

NAME

dpd − data phone daemon

SYNOPSIS

/etc/dpd

DESCRIPTION

Dpd is the 201 data phone daemon. It is designed to submit jobs to the Honeywell 6070 computer via the GRTS interface.

Dpd uses the directory */usr/dpd.* The file *lock* in that directory is used to prevent two daemons from becoming active. After the daemon has successfully set the lock, it forks and the main path exits, thus spawning the daemon. The directory is scanned for files beginning with **df.** Each such file is submitted as a job. Each line of a job file must begin with a key character to specify what to do with the remainder of the line.

S directs *dpd* to generate a unique snumb card. This card is generated by incrementing the first word of the file */usr/dpd/snumb* and converting that to three-digit octal concatenated with the station ID.

L specifies that the remainder of the line is to be sent as a literal.

B specifies that the rest of the line is a file name. That file is to be sent as binary cards.

F is the same as **B** except a form feed is prepended to the file.

U specifies that the rest of the line is a file name. After the job has been transmitted, the file is unlinked.

M is followed by a user ID; after the job is sent, the snumb number and the first line of information in the file is mailed to the user to verify the sending of the job.

Any error encountered will cause the daemon to drop the call, wait up to 20 minutes and start over. This means that an improperly constructed *df* file may cause the same job to be submitted every 20 minutes.

While waiting, the daemon checks to see that the *lock* file still exists. If it is gone, the daemon will exit.

FILES

/dev/dn0, /dev/dp0, /usr/dpd/*

SEE ALSO

opr (I)

-

NAME

dump − incremental file system dump

SYNOPSIS

dump [key [arguments] filesystem]

DESCRIPTION

Dump makes an incremental file system dump on magtape of all files changed after a certain date. The *key* argument specifies the date and other options about the dump. *Key* consists of characters from the set **abcfiu0hds.**

- **a** Normally files larger than 1000 blocks are not incrementally dump; this flag forces them to be dumped.
- **b** The next argument is taken to be the maximum size of the dump tape in blocks (see **s**).
- **c** If the tape overflows, increment the last character of its name and continue on that drive. (Normally it asks you to change tapes.)
- **f** Place the dump on the next argument file instead of the tape.
- **i** the dump date is taken from the entry in the file /etc/dtab corresponding to the last time this file system was dumped with the **-u** option.
- **u** the date just prior to this dump is written on /etc/dtab upon successful completion of this dump. This file contains a date for every file system dumped with this option.
- **0** the dump date is taken as the epoch (beginning of time). Thus this option causes an entire file system dump to be taken.
- **h** the dump date is some number of hours before the current date. The number of hours is taken from the next argument in *arguments.*
- **d** the dump date is some number of days before the current date. The number of days is taken from the next argument in *arguments.*
- **s** the size of the dump tape is specified in feet. The number of feet is taken from the next argument in *arguments.* It is assumed that there are 9 standard UNIX records per foot. When the specified size is reached, the dump will wait for reels to be changed. The default size is 2200 feet.

If no arguments are given, the *key* is assumed to be **i** and the file system is assumed to be /dev/rp0.

Full dumps should be taken on quiet file systems as follows:

```
dump 0u /dev/rp0
ncheck /dev/rp0
```

The *ncheck* will come in handy in case it is necessary to restore individual files from this dump. Incremental dumps should then be taken when desired by:

```
dump
```

When the incremental dumps get cumbersome, a new complete dump should be taken. In this way, a restore requires loading of the complete dump tape and only the latest incremental tape.

DIAGNOSTICS

If the dump requires more than one tape, it will ask you to change tapes. Reply with a new-line when this has been done. If the first block on the new tape is not writable, e.g. because you forgot the write ring, you get a chance to fix it. Generally, however, read or write failures are fatal.

FILES

/dev/mt0 magtape
/dev/rp0 default file system
/etc/dtab

-

SEE ALSO

restor (VIII), ncheck (VIII), dump (V)

BUGS

-

NAME

getty − set typewriter mode

SYNOPSIS

/etc/getty [char]

DESCRIPTION

Getty is invoked by *init* (VIII) immediately after a typewriter is opened following a dial-up. It reads the user's name and invokes the *login* command (I) with the name as argument. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the typewriter line. Arguments other than '0' can be used to make *getty* treat the line specially. Normally, it sets the speed of the interface to 300 baud, specifies that raw mode is to be used (break on every character), that echo is to be suppressed, and either parity allowed. It types the "login:" message, which includes the characters which put the Terminet 300 terminal into full-duplex and return the GSI terminal to non-graphic mode. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the "break" ("interrupt") key. The speed is then changed to 150 baud and the "login:" is typed again, this time including the character sequence which puts a Teletype 37 into full-duplex. If a subsequent null character is received, the speed is changed back to 300 baud.

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 *stty* (II)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, 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 argument.

SEE ALSO

init (VIII), login (I), stty (II), ttys (V)

BUGS

NAME

glob − generate command arguments

SYNOPSIS

/etc/glob command [arguments]

DESCRIPTION

Glob is used to expand arguments to the shell containing “*”, “[”, or “?”. It is passed the argument list containing the metacharacters; *glob* expands the list and calls the indicated command. The actions of *glob* are detailed in the Shell writeup.

SEE

sh (I)

BUGS

NAME

icheck – file system storage consistency check

SYNOPSIS

icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

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 number of blocks missing; i.e. not in any file nor in the free list,
The number of special files,
The total number of files,
The number of large and huge files,
The number of directories,
The number of indirect blocks, and the number of double-indirect blocks in huge files,
The number of blocks used in files,
The number of free blocks.

The **–s** flag 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** flag causes the normal output reports to be suppressed.

Following the **–b** flag 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

Currently, /dev/rrk2 and /dev/rrp0 are the default file systems.

SEE ALSO

dcheck (VIII), ncheck (VIII), fs (V), clri (VIII), restor(VIII)

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.

-

NAME

init − process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is invoked inside UNIX as the last step in the boot procedure. Generally its role is to create a process for each typewriter on which a user may log in.

First, *init* checks to see if the console switches contain 173030. (This number is likely to vary between systems.) If so, the console typewriter **/dev/tty8** is opened for reading and writing and the Shell is invoked immediately. This feature is used to bring up a single-user system. When the system is brought up in this way, the *getty* and *login* routines mentioned below and described elsewhere are not used. If the Shell terminates, *init* starts over looking for the console switch setting.

Otherwise, *init* invokes a Shell, with input taken from the file */etc/rc.* This command file performs housekeeping like removing temporary files, mounting file systems, and starting daemons.

Then *init* reads the file */etc/ttys* and forks several times to create a process for each typewriter specified in the file. Each of these processes opens the appropriate typewriter for reading and writing. These channels thus receive file descriptors 0 and 1, the standard input and output. Opening the typewriter will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. Then */etc/getty* is called with argument as specified by the last character of the *ttys* file line. *Getty* reads the user's name and invokes *login* (q.v.) to log in the user and execute the Shell.

Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal #1) and interprets it to mean that the switches should be examined as in a reboot: if they indicate a multi-user system, the */etc/ttys* file is read again. The Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use "kill −1 1."

FILES

/dev/tty?, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login (I), kill (I), sh (I), ttys (V), getty (VIII)

NAME

lpd − line printer daemon

SYNOPSIS

/etc/lpd

DESCRIPTION

Lpd is the line printer daemon (spool area handler) invoked by *opr.* It uses the directory */usr/lpd.* The file *lock* in that directory is used to prevent two daemons from becoming active simultaneously. After the daemon has successfully set the lock, it scans the directory for files beginning with ‘‘df.’’ Lines in each *df* file specify files to be printed in the same way as is done by the data-phone daemon dpd (VIII).

FILES

/usr/lpd/* spool area
/dev/lp printer

SEE ALSO

dpd (VIII), opr (I)

BUGS

-

NAME

mkfs − construct a file system

SYNOPSIS

/etc/mkfs special proto

DESCRIPTION

Mkfs constructs a file system by writing on the special file *special* according to the directions found in the prototype file *proto.* The prototype file contains tokens separated by spaces or new lines. The first token is the name of a file to be copied onto block zero as the bootstrap program (see boot procedures (VIII)). The second token is a number specifying the size of the created file system. Typically it will be the number of blocks on the device, perhaps diminished by space for swapping. The next token is the i-list size in blocks (remember there are 16 i-nodes per block). The next set of tokens comprise the specification for the root file. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

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

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

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

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

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

If the prototype file cannot be opened and its name consists of a string of digits, *mkfs* builds a file system with a single empty directory on it. The size of the file system is the value of *proto* interpreted as a decimal number. The i-list size is the file system size divided by 43 plus the size divided by 1000. (This corresponds to an average size of three blocks per file for a 4000 block file system and six blocks per file at 40,000.) The boot program is left uninitialized.

A sample prototype specification follows:

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

SEE ALSO

file system (V), directory (V), boot procedures (VIII)

BUGS

It is not possible to initialize a file larger than 64K bytes.
The size of the file system is restricted to 64K blocks.
There should be some way to specify links.

NAME

mknod − build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file. The first argument is the *name* of the entry. 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).

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

SEE ALSO

mknod (II)

BUGS

NAME

mount – mount file system

SYNOPSIS

/etc/mount special file [**–r**]

DESCRIPTION

Mount announces to the system that a removable file system is present on the device corresponding to special file *special* (which must refer to a disk or possibly DECtape). The *file* must exist already; it becomes the name of the root of the newly mounted file system.

Mount maintains a table of mounted devices; if invoked without an argument it 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.

SEE ALSO

mount (II), mtab (V), umount (VIII)

BUGS

Mounting file systems full of garbage will crash the system.

NAME

ncheck – generate names from i-numbers

SYNOPSIS

ncheck [–**i** numbers] [–**a**] [filesystem]

DESCRIPTION

Ncheck with no argument generates a pathname vs. i-number list of all files on a set of default file systems. The –**i** flag reduces the report to only those files whose i-numbers follow. –**a** allows printing of the names '.' and '..', which are ordinarily suppressed. A file system may be specified.

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

SEE ALSO

dcheck (VIII), icheck (VIII), sort (I)

BUGS

NAME

restor − incremental file system restore

SYNOPSIS

restor key [arguments]

DESCRIPTION

Restor is used to read magtapes dumped with the *dump* command. The *key* argument specifies what is to be done. *Key* is a character from the set **trxw.**

t The date that the tape was made and the date that was specified in the *dump* command are printed. A list of all of the i-numbers on the tape is also given.

r The tape is read and loaded into the file system specified in *arguments.* This should not be done lightly (see below).

x Each file on the tape is individually extracted into a file whose name is the file's i-number. If there are *arguments,* they are interpreted as i-numbers and only they are extracted.

c If the tape overflows, increment the last character of its name and continue on that drive. (Normally it asks you to change tapes.)

f Read the dump from the next argument file instead of the tape.

i All read and checksum errors are reported, but will not cause termination.

w In conjunction with the **x** option, before each file is extracted, its i-number is typed out. To extract this file, you must respond with **y.**

The **x** option is used to retrieve individual files. If the i-number of the desired file is not known, it can be discovered by following the file system directory search algorithm. First retrieve the *root* directory whose i-number is 1. List this file with *ls −fi 1.* This will give names and i-numbers of sub-directories. Iterating, any file may be retrieved.

The **r** option should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape onto this. Thus

```
/etc/mkfs /dev/rp0 40600
restor r /dev/rp0
```

is a typical sequence to restore a complete dump. Another *restor* can be done to get an incremental dump in on top of this.

A *dump* followed by a *mkfs* and a *restor* is used to change the size of a file system.

FILES

/dev/mt0

SEE ALSO

ls (I), dump (VIII), mkfs (VIII), clri (VIII)

DIAGNOSTICS

There are various diagnostics involved with reading the tape and writing the disk. There are also diagnostics if the i-list or the free list of the file system is not large enough to hold the dump.

If the dump extends over more than one tape, it may ask you to change tapes. Reply with a new-line when the next tape has been mounted.

BUGS

There is redundant information on the tape that could be used in case of tape reading problems. Unfortunately, *restor's* approach is to exit if anything is wrong.

-

NAME

sa − Shell accounting

SYNOPSIS

sa [−**abcjlnrstuv**] [file]

DESCRIPTION

When a user logs in, if the Shell is able to open the file */usr/adm/sha,* then as each command completes the Shell writes at the end of this file the name of the command, the user, system, and real time consumed, and the user ID. *Sa* reports on, cleans up, and generally maintains this and other accounting files. To turn accounting on and off, the accounting file must be created or destroyed externally.

Sa is able to condense the information in */usr/adm/sha* into a summary file */usr/adm/sht* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system *sha* can grow by 100 blocks per day. The summary file is read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; *sha* is the default. There are zillions of options:

a Place all command names containing unprintable characters and those used only once under the name ‘‘***other.’’

b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

c Besides total user, system, and real time for each command print percentage of total time over all commands.

j Instead of total minutes time for each category, give seconds per call.

l Separate system and user time; normally they are combined.

n Sort by number of calls.

r Reverse order of sort.

s Merge accounting file into summary file */usr/adm/sht* when done.

t For each command report ratio of real time to the sum of user and system times.

u Superseding all other flags, print for each command in the accounting file the day of the year, time, day of the week, user ID and command name.

v If the next character is a digit *n,* then type the name of each command used *n* times or fewer. Await a reply from the typewriter; if it begins with ‘‘y’’, add the command to the category ‘‘**junk**.’’ This is used to strip out garbage.

FILES

/usr/adm/sha	accounting
/usr/adm/sht	summary

SEE ALSO

ac (VIII)

BUGS

NAME

su − become privileged user

SYNOPSIS

su

DESCRIPTION

Su allows one to become the super-user, who has all sorts of marvelous (and correspondingly dangerous) powers. In order for *su* to do its magic, the user must supply a password. If the password is correct, *su* will execute the Shell with the UID set to that of the super-user. To restore normal UID privileges, type an end-of-file to the super-user Shell.

The password demanded is that of the entry ‘‘root’’ in the system’s password file.

To remind the super-user of his responsibilities, the Shell substitutes ‘#’ for its usual prompt ‘%’.

SEE ALSO

sh (I)

NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. See sync (II) for details.

SEE ALSO

sync (II)

BUGS

NAME

umount – dismount file system

SYNOPSIS

/etc/umount special

DESCRIPTION

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

SEE ALSO

mount (VIII), umount (II), mtab (V)

FILES

/etc/mtab mounted device table

DIAGNOSTICS

It complains if the special file is not mounted or if it is busy. The file system is busy if there is an open file on it or if someone has his current directory there.

BUGS

-

NAME

update − periodically update the super block

SYNOPSIS

update

DESCRIPTION

Update is a program that executes the *sync* primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file. See sync (II) for details.

SEE ALSO

sync (II), init (VIII)

BUGS

With *update* running, if the CPU is halted just as the *sync* is executed, a file system can be damaged. This is partially due to DEC hardware that writes zeros when NPR requests fail. A fix would be to have *sync* temporarily increment the system time by at least 30 seconds to trigger the execution of *update.* This would give 30 seconds grace to halt the CPU.

-

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

The sender should be super-user to override any protections the users may have invoked.

FILES

/dev/tty?

SEE ALSO

mesg (I), write (I)

DIAGNOSTICS

“Cannot send to ...” when the open on a user’s tty file fails.

BUGS