Unix Administration

David Malone

March 20, 2001

Contents

1 The Function of the Unix Administrator

The task of the Unix Administrator is difficult to define. Broadly speaking it involves ensuring that the Unix system provides the service it was intended for, and can continue to do so for as long as it is needed. These tasks vary from situation to situation, but will usually include the following.

- Adding and removing users.
- Adding and removing hardware.
- Adding and removing software.
- Performing backups.
- Monitoring the system to ensure correct operation.
- Troubleshooting.
- Documentation.
- Auditing security.
- Helping users.

While carrying out these tasks It is important to remember that the Unix system exists to provide some service, and that the administrator should be aiming to support that service — not hinder it.

2 Basic Unix Objects

From the technical side the Unix Administrators job is to work with the basic objects within Unix, so it is important to understand what these are.

files Files serve many functions in Unix. They are where both data and programs are stored. Every file has certain attributes: 3 dates, a user and group who own it and a set of permissions.

There are many types of special file on Unix: directories, links, devices, FIFOs and sockets.

File manipulation commands include: `ls, rm, touch, chmod, chown, chgrp, find, ln, mkdir`.

processes A process is the basic unit of program execution in Unix. Process may be in several states — running, ready to run, waiting for data or suspended. The "load" on a Unix box is the average number of processes running or ready to run.

Each process has a user and group to which it belongs. This determines two things: first what files the process may access and second who may send signals to the process. Signals can kill, suspend, restart a process or notify it of some event.

Typically processes on a Unix machine are either providing some service (dæmons), or belong to users. Many processes are created during the boot procedure. These process then create any other needed processes by forking.

If a process is connected to a terminal then certain key presses can cause signals. The most common of these are `ctrl-C` to kill, `ctrl-Z` to suspend and `ctrl-\` to abort.

Process manipulation commands include: `ps, kill, nice, renice`.

devices One type of Unix special file is the device. Devices correspond to services within the Unix kernel, and often represent disks, tapes and serial ports.

These special device files usually reside under the `/dev` directory, but can be made anywhere in the file system. Each device has two numbers which connect it with the service in the kernel - these are referred to as the major and minor device numbers. The naming of devices within `/dev` is not standard across different versions of Unix.

Device files have attributes just like normal files, and their permissions should be set accordingly.

Section 12 of the DRS/NX manual talks a little about devices, their naming and the major and minor device numbers.

users & groups Users and groups are not really fundamental Unix objects, beyond each process and file having an associated user and group. The Unix kernel knows nothing of user names and group names and treats each user and group as a number — the user ID (uid) and group ID (gid).

However Unix utilities such as `ls` or `ps` have used user names instead of numbers — as this is easier for people to deal with.

The mapping of names to numbers is usually done in the `/etc/passwd` and `/etc/group` files. These files also provide other information which are useful to Unix utilities (eg. Home directory, shell) and to people (eg. Users real name). For the normal running of a Unix system an uncorrupted `/etc/passwd` is essential.

file system layout The standard Unix file system layout is also not a fundamental Unix object, but is essential for the normal operation of a Unix system. This layout is not completely standard across all Unix systems, but there is usually a strong degree of agreement.

`/bin`	Essential commands
`/dev`	Devices
`/etc`	Configuration files and boot scripts
`/sbin`	Essential system commands
`/home`	Home directories for users and projects
`/mnt`	Tempery file system mount point
`/proc`	Special process monitoring file system
`/tmp`	Scratch/Scratch files
`/usr/bin`	Non-essential user commands
`/usr/include`	Header files for compilers
`/usr/lib`	Shared and static libraries
`/usr/local`	Like /usr, but for local software
`/usr/sbin`	Non-essential system commands
`/usr/share`	Files that can be shared between machines
`/usr/share/man`	Manual pages (sometimes `/usr/man`)
`/var`	Files specific to this machines running
`/var/spool`	Spool space for mail, printers, periodic jobs

On older systems `/var` and `/usr/share` may be merged into `/usr` and `sbin` directories merged into `etc` and `bin` directories.

privilege The privilege system in Unix is quite simple. Generally a process may only access a file if it is in the running as the user or is in the correct group. The only exception is if the uid is zero. This usually corresponds to the user name `root`.

Root has other privileges — the ability to send a signal to any process, the ability to run certain system calls (eg. reboot), the ability to change to any uid and the ability to make certain network connections.

Privilege is gained in Unix by running a program with the set user ID attribute (suid). These programs usually perform whatever privileged action is needed and then discard their privilege by switching to another uid.

To get a shell which is running as root, the `su` program is usually used. This is considered better than logging straight in as root, as it logs a message saying who became root.

3 Useful Tools for the administrator

The most useful tool for the Unix administrator is the `man` command — which allows you to look through the on line manual pages. These are often more up to date than the printed manual and more quickly searched through.

Below are some commands which are useful in day to day administration. These are the basic tools of the Unix administrator.

grep Grep allows you to search files for lines which contain a certain set of characters. The most basic use might be to find a user in the password file.

```
grep user /etc/passwd
```

A more complicated use might involve finding boot up file starts a certain service.

```
grep -l serviced /etc/* /etc/*/*
```

You could look for people using `su` to change to users other than root.

```
grep "su:" /var/log/messages | grep -v "to root"
```

awk Awk allows you to print out certain columns in a file. For instance to print a list of user names.

```
awk -F: '{ print $1 }' /etc/passwd
```

Note that the forward quotes are needed, as otherwise your shell would fill in for `$1`. Awk can perform more complicated tasks too. To add up the number of bytes sent by a daemon (the last field in its log file):

```
awk '{ bytes += $NF } END {print bytes}' access.log
```

find Allows you to find files matching a certain description. For example to find all files bigger than 1MB which have changed in the last week.

```
find / -size +2048 -mtime -7 -print
```

To add up the size of all files owned by a user on /home.

```
find /home -user joe -type f -ls |
        awk '{ bytes += $7 } END {print bytes}'
```

sort & uniq Sort sorts what it is given, uniq removes duplicate lines which are adjacent to one another. This is often useful to the administrator in the `| sort | uniq` pipe. For instance to make a list of all users which have been su'ed to.

```
grep su:  /var/log/messages |
        awk '{ print $NF }' | sort | uniq
```

Other commands which are worth learning to use include: `head, tail, sed, xargs` and `diff`. A tool called `perl` provides a powerful scripting language — including many of the features of `grep, awk, sed, sh, csh` and `find`. However, Unix vendors are only beginning to provide this tool as a standard.

4 Startup, Shutdown and Rebooting

The starting of Unix on any system consists of two stages. First the Unix kernel must be loaded and then the Kernel runs the `init` program which calls the programs necessary to place the system in a usable state.

The first stage is both vendor and hardware dependent. The behavior of `init` is related to the family to which your Unix belongs (BSD or SysV). System V booting, as in described in the DRS/NX manual will be covered below.

4.1 Finding the Kernel

The DRS6000 uses a 3 stage boot to find the kernel. A multi-stage boot is useful because it allows the same version of Unix to be booted from several different media, by using a middle stage suitable for the medium in question.

- The system boots from the CMS PROM, which loads defaults from NVRAM.
- The CMS PROM examines the default boot device to find a boot program, `boot2` stored in a "boot partition". This partition is what would be thought of as a boot block on a floppy disk.
- The `boot2` program locates the `/stand` partition which contains a simple file system of type `bfs`. From here `boot2` loads the `boot3` program which performs some initialization and then loads Unix.

The CMS PROM has some simple commands. The `boot` command boots the system from the default device — or if a channel and device number are provided from that device. The `setdefault` command sets the default channel and device if they are given, otherwise it shows the current defaults.

The `boot2` program does not really take any options, as it exists mainly to find the `boot3` program. The `boot3` program takes quite a few options which can be listed in `/stand/preconf`. See pages 3a-28 – 3a-31 and 4-11 – 4-12 for details of these options.

4.2 After the Kernel

Once the kernel has been loaded it will perform various initializations and then start the `init` program. This program will kick start the rest of the boot process. At this stage only the root file system will be mounted (probably read only), and the only process running is `init`.

The behavior of `init` is determined by the `/etc/inittab` file. For an exact description of this file it is best to refer to the manual pages of the system in question. The most important notion of this file is that of a run level. For a description of the various run levels in DRS/NX see page 3a-16.

`/etc/inittab` lists the default run level and jobs to be done when entering a new run level. These jobs typically include:

- Activating swap devices.
- Preening file systems with `fsck`.
- Setting permissions on files in `/dev`.
- Starting the most basic dæmons.

On System V machines there is a standard procedure for adding new jobs to the sequence for entering a new run level. Take, for example, the startup of run level 2. The script `rc2` runs all files in `/etc/rc2.d` with names like `S23cron` with the argument `start`, sorted by the number in their name. These files will usually be links to files in `/etc/init.d` for easy management.

This leads to the following standard procedure for adding new jobs to run level 2.

1. Write a script that when passed the argument `start` performs the required job.
2. Copy this to `/etc/init.d`.
3. Make a link to this file from `/etc/init.d` with a suitable name.

Sometimes it may be more appropriate to add a job directly to `/etc/inittab` - this will usually be when the job does a one off task and does not provide a service.

4.3 Stopping and Restarting

Stopping or restarting the machine really corresponds to changing the run level. Changing the run level will usually end all the sessions of people working on the machine, and so should only be done in emergencies or at a scheduled time. Warning of impending shutdown can be provided to users with the `wall` command.

There are two ways of changing the run level. The most direct way is to just to tell `init` to go into a new run level. For example to go to run level 6 (reboot):

```
init 6
```

This, however, provides little time to change your mind, and doesn't warn users or try any cleaning up. For a more graceful change of run level use the `shutdown` command. On SysV it commonly takes the following arguments:

`-g`*n* Wait *n* seconds before changing run level.
`-i`*l* Change to run level *l*.
`-y` Answer "yes" to all "are you sure?" questions.

For instance, to shut the machine down completely in about 15 minutes:

```
shutdown -g900 -i0 -y
```

If you change your mind during this time, you can often prevent the change of run level by killing the shutdown process (find the process ID with `ps`).

4.4 Special Situations

There are certain times when booting fully may not be desirable, and having the system booted to the stage where it is just usable is useful. This state is called single user mode, and is usually run level `s`. You can enter single user mode by using `init` or `shutdown` to go to run level `s`. In this state you may find that file systems have been unmounted or may be mounted read only. If you wish to change these file systems you will have to change their status manually.

Single user mode can often be entered from the ROM stage of booting a system, by providing options to the Unix kernel. The DRS/NX manual does not specify how to do this, but it may be possible with the `unixopts` option to `boot3` in the `/etc/preconf` file. Check the manual pages to be certain.

Occasionally booting as far as as the Unix kernel may be impossible because of a damaged disk. In this case you may have to boot from an emergency file system. Pages 3a-41 – 31a-42 detail this procedure for the DRS/6000. Test that you can perform this procedure periodically — this (in combination with regular backups) may be your only means of recovering the machine in case of serious failure.

5 Shell Scripting

Having a basic grasp of shell scripting is essential for effective administration. It provides a way of automating administration tasks and is also necessary for understanding parts of the system such as the boot time scripts.

Shell scripting involves knowing how to string Unix commands together, and how to use a shell to connect these commands sensibly. It is probably best learned by example — Firsch provides several commented examples in Chapter 8 and Appendix A. However examining boot scripts will provide examples and familiarity with the boot process.

5.1 Hupping a Process

This script sends a `HUP` signal to the HTTP dæmon, whose process ID should be in the file `/var/run/httpd.pid`.

```
#!/bin/sh
pidfile=/var/run/httpd.pid
if [ -f $pidfile ] ; then
  kill -HUP ‘cat $pidfile‘
fi
```

5.2 Zapping Certain Processes

This script searches the process table for processes which match a certain description. It then asks if you would like to kill those which match.

```
#!/bin/sh

tmpfile=/tmp/proclist.$$

echo -n "String to search for:  "
read grepstring

ps -aux | fgrep $grepstring > $tmpfile

for pid in ‘awk ’{ print $2 }’ $tmpfile‘
do
  awk ’$2 == ’$pid’ { print $0 }’ $tmpfile
  echo -n "Kill this process(y/n)?  "
  read ans
  if [ $ans = "y" ] ; then
    kill $pid
  fi
done
```

5.3 Rotating Log Files

This script moves on log files listed on the command line. It keeps all but the most recent one compressed, and removes the last one once there are more than `CYCLES` of them. For example, `CYCLES=3 ; rotate messages` Would have the following effects.

```
messages      ⟶  messages.1
messages.1    ⟶  messages.2.gz
messages.2.gz ⟶  messages.3.gz
messages.3.gz ⟶  removed
```

```
#!/bin/sh
# Rotate a log file and keep N copies
# Mostly stolen from inn

CYCLES=${CYCLES-5}
COMPRESS=/usr/local/bin/gzip
Z=.gz

for F in $* ; do
  ## Compress yesterday's .1
  test -f ${F}.1 \
    && ${COMPRESS} <${F}.1 >${F}.1${Z} \
    && rm -f ${F}.1 \
    && chmod 0440 ${F}.1${Z}

  ## Do rotation.
  EXT=${CYCLES}
  rm -f ${F}.${CYCLES}${Z}
  while [ ${EXT} -gt 0 ] ; do
    NEXT=${EXT}
    EXT=`expr ${EXT} - 1`
    test -f ${F}.${EXT}${Z} \
      && rm -f ${F}.${NEXT}${Z} \
      && mv ${F}.${EXT}${Z} ${F}.${NEXT}${Z}
  done
  mv ${F} ${F}.1
done
```

5.4 The Cron Facility

The cron facility in Unix makes shell scripts very useful for performing tasks such as log file rotation, backups and regular system monitoring completely automated.

Cron allows a command to be executed every hour, day, week, month or year at a given time. Often this command will be a shell script designed to perform some job and mail the results to the user who set it up.

Cron has taken several forms in different versions of Unix, and to find out which version you are using you will need to look at the manual pages for `cron`, `crontab` (the program) and `crontab` (the file). Files belonging to cron can often be found in `/etc/crontab`, `/var/cron` and `/var/spool/cron`.

When removing user accounts on the system it is a good idea to check if they have cron jobs (or `at` jobs) — if they had the ability to set these up. It is often possible to restrict who may set up these jobs.

5.5 A Final Note

Some tasks are difficult or impossible to achieve directly with a shell script. For these you may have to resort to more powerful tools. More powerful scripting languages include `perl` and `expect`, though you may have resort to C on occasions.

Tools for synchronizing file systems over several machines include `rdist` and `rsync`. You can actually even write "scripts" for the `find` program — as it is quite complicated.

```
/usr/local/bin/find / -fstype nfs -prune -o \
  -path '/news/spool' -prune -o \
  -path '/www/EMIS' -prune -o \
  -path '/empty/cache*' -prune -o \
  \( -name '*.core' \
  \) -a -atime +1 -print0 \
  -o \
  \( -name '[#,]*' \
    -o -name '.#*' \
    -o -name '*.#old' \
    -o -name '.*.#old' \
    -o -name a.out \
    -o -name '*.CKP' \
    -o -name '.emacs_[0-9]*' \
    -o -name '*~' \
    -o -name '.*~' \
  \) -a -atime +3 -print0 \
  -o \
  \( -name .nfs \
  \) -a -atime +7 -print0 \
  -o \
  \( -name .mosaic-global-history \
    -o -name .newsrc.bak \
    -o -name .netscape-history \
    -o -name .MCOM-global-history \
  \) -a -atime +60 -print0 \
  | /usr/local/bin/perl -n0e 'unlink;'
```

6 Monitoring the System

The system should be monitored from a day to day basis. This is not necessarily to justify continuous small or cosmetic "tweaks" to the system, but more to make it possible to determine when a major change is needed. That is not to say that tweaking is unhelpful, but there are often more important tasks at hand.

Some aspects of the system which may require monitoring are listed below.

- Running processes.
- Available disk space.
- Disk and network performance.
- Log files and process accounting.

Some of these require very little effort to monitor, others require some degree of scripting to produce useful reports. The key to all is to become familiar with the normal state of the system.

6.1 Monitoring Processes

There are two main resources which can be used by processes — memory and CPU. It is usually fairly easy to tell if the system is short of these, as response to commands will feel unusually sluggish. The primary source of information about the state of processes is the `ps` command. It can display the size of a process, who owns it and how much CPU time it has used. It can produce rather too much information in some cases — use `grep`, `sort` and `diff` to find what is needed.

One problem that frequently occurs a runaway process which is using all the CPU time it can get. Use `ps` to find these processes and then `kill` to get rid of them.

A summary of the use of the CPU is the load averages. These are displayed by the commands `uptime` and `who`. You should be familiar with the usual values of the load.

Serious memory shortages are usually characterized by jerky responses from the system. The amount of memory available can be examined using `vmstat`. There may be a process hogging memory, but this is more unusual that a process hogging the CPU.

Chronic memory shortages — running out of swap space — will usually be signaled by random processes dying, awful performance and messages to the console or log files. should never happen in the normal operation of the system.

Real shortages of CPU, memory and swap space can be aided by tuning kernel parameters, or adding more of the scarce resource.

6.2 Monitoring Available Disk Space

The main commands for monitoring disk space are `df` and `du`. An overview of used and available disk space on each file system is provided by `df`, while `du` gives a breakdown by directory.

The main complication with monitoring disk space is that no one notices file systems filling up until they are full. For this reason it is useful to keep the output of `df` going back several days or weeks. From this it can be determined if the filling disk is due to normal growth, or is due to some accident.

If an accident is identified, then it is usually possible to find the files in question using the `find` command - from the previous outputs you should be able to determine when the file was created, and its approximate size.

If usual use of the system creates temporary files it is useful to have a periodic job which searches for old temporary files and removes them. You should warn your users about this!

The best solution to a real disk space problem is to get more disk space. Some people advocate convincing users to clean up — this is rarely effective.

6.3 Disk, Network Performance

If a machine is heavily loaded and performing badly, but seems to have CPU time and memory available for use then the problem is likely to be disk or network related. You can use `iostat` to monitor how busy various disks are.

Again, this is difficult to monitor as no one notices until a problem arises. Again, the best solution is to provide a faster disk or network, but planning of disk usage can provide solutions also.

1. Use the network from scripts during off peak times to transfer large amounts of data.
2. Where data is being processed by a machine, try to ensure it is on a local disk first.
3. Split disk intensive services across different physical disks.
4. Remember — if the system is short of memory then adding swap space on a already busy disk may slow things down.

The command `netstat` can provide you with various network statistics. Bad network performance can be caused by bad cabling, which may show up in these statistics. However, bad cabling usually shows up as intermittent network outages.

6.4 Log files

Log files provide lots of information, often too much. Again `grep` and friends will allow you to make sense of what is provided. Log files often contain early warnings of pending hardware failures. You should check the log files regularly so you know what to expect.

Log files are usually created by `syslogd`. This program can be configured to log different amounts to different places. It is configured through the file `/etc/syslog.conf`, and must be sent a `HUP` signal when a change is made to the file.

Various versions of `syslogd` allow log messages to be sent to files, users' terminals, other Unix machines or other programs. Usually `syslogd` will not create a new file for logging in to. The creation of the file should be performed whenever log files are rotated.

Finally, it is sometimes useful to send all log messages to your terminal if you are trying to solve a problem.

7 System Security

Computer security involves protecting your computer system (including software, hardware, data and services) from theft, modification and destruction. This includes both accidental and malicious actions.

The most basic level of defence is a recent full backup of the system. “Recent” means recent enough that only an acceptable amount of work will be lost. “Full” means sufficient to rebuild the system and place it in the same state as it was in when the backup was taken. For security a copy of such a backup should be kept in a location remote from the actual machine.

7.1 Basic Unix Security

Since protection of files in Unix is based on file ownership and attributes it is important to ensure files have these set correctly.

Likewise, as privilege is based on the idea of a single privileged user, access to process running as this user should be carefully controlled.

This is not always possible, as services which users or outsiders interact with must sometimes run as root. A poorly designed or coded service may be a security hole. Usually the only solution to these sort of problems is to disable the service or get an upgrade from the vendor.

However, even if these are correctly protected it is difficult to protect Unix from denial of service (DOS) attacks by malicious users of the system. Some protection from this sort of attack can be provided by process limits, but Unix is not designed with protection from this form of attack in mind.

Section 5 of the DRS/NX manual provides some advice on security. Some bits of it are more sensible than other bits.

7.2 Security and People

Security starts and ends with people. People should be told the degree of responsibility that goes with the degree of access they have to a system. Encourage them to form habits of good security practices.

At the end of the day remember, if three guys with baseball bats turn up on your door step asking for the root password, you’ll probably tell them.

7.3 Security and Passwords

The most direct assault on security the obtaining of a password. A password could either be guessed or stolen.

Stealing passwords might be achieved by watching someone typing, stealing a piece of paper containing the password, tricking the person into typing the password or observing the password by watching network traffic.

Passwords can be guessed in several ways. The first it just to try to log in as the person, and try likely passwords. This scheme is rather inefficient as often a system will only allow one login attempt every few seconds.

The other scheme for password guessing is to obtain the users encrypted password. This allows you to guess passwords at the full speed allowed by available CPU power. Using a modest network of PC's it would be possible to try 300,000,000 passwords per hour. This makes large dictionaries (with "obvious" modifications) viable guess lists.

To attempt to avoid this problem, more recent versions of Unix have moved the encrypted passwords from `/etc/password` to a file only readable by root, often called `/etc/shadow`.

The other option is to choose difficult to guess passwords. These passwords can be difficult to remember. Most users type their password once or twice a day, which is often frequently enough for their fingers to "learn" the password within a week.

Note, that services such as remote login will not always ask for a password. It is important that if use services which use network based authentication without passwords that you trust the networks and hosts involved.

Many vendors advocate regular password changes. There is debate concerning the wisdom of this measure. It does not improve security in all situations.

7.4 Security and Files

Checking the permissions on files is a job for `find`. There are two obvious security problems which can easily be checked for with `find` — world writable files and new suid files. For example, to find world writable files:

```
find / -perm -002 -print
```

On systems with the `ncheck` command, `ncheck -s` may provide a faster way of finding all suid files. Lists of suid files are often kept, and compared from night to night for changes. It is wise to also monitor devices.

7.5 Some Things to Watch for

- Users in the password file with user ID 0.
- Having "`.`" in your path somewhere before the end.
- People having physical access to the machine.
- Users should not tell anyone their password. Remind them that the system administrator does not need their password, and no one else should know it.
- If a security breach is suspected/detected then gather as much information as you can quickly. Check log files, the "dot" files of the user involved, the `last` command, and the `lastcomm` command. The information may be gone tomorrow.

8 Extra Items from Day 1

Here are some of the extra items covered on day 1.

sar The Syetem Activity Reported, which provides the information that `vmstat` or `iostat` would. It has many options some of which are descirbed in the manual page and on pages 15-19 to 15-36 of the DRS/NS manual.

quot A breakdown of which users are using how much disk space can be obtained in an efficent manner with the `quot` command.

umask Umask controls what permissions are not given when creating files and directories. For instance `umask 022` would not write permission to group and others, while `umask 007` would give user and group all permission while not giving others any. It is a good command for login scripts.

ulimit & limit Process limits can be controled with the command `ulimit`, or `csh`'s built in alternative `limit`. It is often useful to limit `coredumpsize` to zero in peoples login scripts.

/etc/defaults This is where the default config files of many programs are kept under SysV.

PATH This is a ":" seperated list of directories to search for commands (from left to right). It can be set in `csh` by saying:

`setenv PATH /bin:/usr/bin:/usr/sbin` Likewise it may be set in `sh` by saying:

`PATH=/bin:/usr/bin:/usr/sbin export PATH`

Files with Holes In Unix a file may have a "hole" in it. If you opened a file, wrote a byte, seeked forward 1000000 bytes and wrote another byte you would have a file with a hole. The file would actually take two blocks. You can use `du` to show the amount of space a file actually takes. Utilities such as `cp` and `tar` often expand holes.

9 Disks, File Systems and Swap Space

The addition of disks to a system is another area of disagreement between vendors. The stages are as follows:

1. Plan the use of the disk. Decide how you will partition the disk and what each partition will be used for.
2. Figure out how you will connect the disk to the system. For example, on a SCSI system this will require you to find an unused SCSI ID on one of your SCSI busses.
3. Shut down the machine, and actually connect the disk.

4. If the systems lists connected disks at boot time check the disks shows up.
5. Make the device[1] files for the disk if necessary.
6. Now format, partition and create file systems on the disk.
7. Mount the file systems on the disk, and activate the swap space.
8. Populate file systems from tape or disk.

9.1 Formatting, Partitions and File Systems

The use of "formatting" has become somewhat confused by the advent of floppy disks. There are three stages to preparing a disk for use: formatting, partitioning and file system creation.

On a floppy disk all these stages are rolled into one. To add to the confusion, hard disks do not need to be formatted any more (unless there is a serious failure), but do need to be partitioned and have file systems created.

A disk is usually split into regions called partitions. A partition is just a chunk of the disk starting at sector a and going to sector b. Under Unix partitions are usually allowed to overlap, and as long as two overlapping partitions are not used at the same time all should be well.

Partitions are also usually assigned some sort of "type". Under DRS/NX a partition may be used for a file system, swap, dump, boot or nothing. This type is assigned at the time the disk is partitioned, but may be changed later if the partition is not in use.

The command for partitioning disks under Unix is both vendor and hardware dependent. Sometimes partitioning is called labeling. On the DRS/6000 the command for partitioning is `fmthard` — you can understand why people are confused.

Creating a file system on a partition involves creating the basic structure on the hard disk needed to represent an empty file system. There is a greater degree of standardization for this task, and it can usually be performed with the `mkfs` program. On some versions of Unix this has been replaced/complimented by the less obtuse `newfs` program.

DRS/NX offers you a choice of file systems. The old SysV file system is supported as `s5`. The new "fast file system" is supported as `ufs` and a simple file system, used for booting is provided as `bfs`. The file system you will want to use for all normal file systems is `ufs`. There are various parameters for each file system which can be changed, however for most purposes the defaults are fine.

9.2 Mounting and Unmounting File Systems

Once a partition has a file system on it, it must be "mounted" somewhere within the currently mounted file systems. This makes the files in that file system available below the point chosen.

[1]The creation of device files on DRS/NX is covered on pages 12-11 to 12-12. Adding devices to DRS/NX's database is discussed on pages 12-14 to 12-28

Most file systems are mounted automatically at boot time. These files systems are listed in the file `/etc/vfstab`. Unmounted or new file systems may be mounted while the system is running with the `mount` command.

```
mount -F bfs /dev/rdsk/c0d1s3 /mnt
```

A file system which is mounted may be unmounted using `unmount`, however no file in the file system may be in use when you try to unmount it. "In use" includes directories who are the current directory of some running process — this means you may have to kill some processes to be able to unmount a file system.

Check the manual pages for `mount` and `unmount` — they often have useful options which are system specific.

9.3 Fsck

During the Unix boot process, the sanity of the file systems is checked. This is with the `fsck` program. This program can correct many minor file system problems, without interaction from the user. However sometimes it encounters a problem which it can not fix with out potential loss of data.

In this case the boot process will usually halt, and you will be asked to check the file systems by hand. This involves running `fsck`, which will prompt you and ask if you wish to make changes. Unless you know a lot about the file system, you probably have no choice but to answer "yes". In fact there is usually an option `-y` to fsck which answers yes to all questions.

There are some alternatives to answering yes.

1. The system may allow you to mount the file system "read only". If the file system is not too badly damaged you can now back it up to tape. You can then safely answer yes to all `fsck`'s questions and restore the data from tape. However, if the file system is very badly damaged, mounting it read only may cause the system to panic.

2. If all the file systems on a disk seem to be damaged it may be the case that the cable to the disk has come loose or some such. In this case check all connections to the disk before answering yes to these questions.

3. You can go through the list of questions answering no, to see what `fsck` intends to do. You may find that it will try to remove various files, some of which you may not consider important. Say yes to the removal of these first, and then run `fsck` again to see if it can now clear up the mess.

Note, that you may need to rerun `fsck` several times to get the file system in a sane state. When `fsck` runs with no complaints, then the damage has been cleared up.

In very rare cases `fsck` may get stuck on a very messed up inode. From the messages `fsck` produces you should know the number of this inode. In this case you can use `clri` and `fsdb` to clear this inode. This procedure is not for the weak of heart.

Remember, if the file system is badly enough damaged it may be easier to clear the file system with `mkfs` and restore from tape. Also, if `fsck` finds any files and cannot work out where they should be it will put them in the `lost+found` directory at the top of that file system.

10 Backup and Restore

Backup and restore essentially consists of the copying of large numbers of files from one place to another. DRS/NX provides a suite of tools for backup and restore which are described in some detail in the manual. Unix in general provides three tools for the large scale storage of files: `tar`, `cpio` and `dump/restore`. At least two of these systems should be available on any system.

Forming a schedule for backups can be simple if all your data than can be backed up in one session. You just do full backups every night. If you end up with more data you may have to work with incremental backups. In an incremental backup system each backup has a level, which is a number between 0 and 9 (say). During a level 0 backup you backup everything. During a level 1 backup you backup everything that has changed since the last level 0. During a level 2 you back up everything that has changed since the last level 0 or 1 dump, and so on.

Using a tape drive on Unix is also an interesting direct encounter with a file in `/dev`. The tape device behaves like a series of files. When you read it you read the current file on the tape, and when you write it you write to the current file on the tape. Usually there are at least two different tape devices. One rewinds the tape when you finish reading or writing, and the other leaves the tape ready to read the next file. Using the `mt` command you can rewind, eject, fast forward the tape drive.

10.1 Tar

The `tar` command is for Tape Archiving. Today is is often used for the distribution of Unix software. What `tar` does is take a list of files and make one file from them — a "tar file". This is much like the `pkzip` program for the PC, except no compression is done[2].

The tar program has two weaknesses. First, many vendors implementations of it cannot deal with special files. Second it does not understand the tape system well, and has trouble with multi-volume backups. These reasons combined make `tar` unsuitable for full backups, but quite suitable for distribution of software.

One other use of `tar` is the copying of directories from one place to another. The `cp` command is often not suitable for this as it may change the ownership of files. A way to copy directory `/dir1` to `/dir2` follows.

```
cd /dir1 ; tar cf - .  | ( cd /dir2 ; tar xpf - )
```

10.2 Cpio

All System V machines come with `cpio`, but some do not come with `tar`. Both programs do much the same job. The main difference is that `tar` takes lists of files for archive/extraction on the command line and `cpio` takes the list on stdin.

Many versions of `cpio` also suffer from the same problems as `tar` has with multi-volume backups, however it doesn't seem to have as much trouble with device files. For this reason `cpio` can be used to back up small file systems quite successfully.

[2]The file may be compressed later with any compression program.

There are several different formats of archive which `cpio` can write and read. Not all versions of `cpio` support all the archive formats, so `cpio` is less often used as a way of distributing software in the Unix world.

Is is also possible to copy directories with `cpio`.

```
cd /dir1 ; find .  -depth -print | cpio -pdm /dir2
```

10.3 Dump and Restore

`Dump` and `restore` are the best widely available Unix tools for backup and resore. They provide incremental backups, interactive restore and can deal with all types of special file. They examine the file system at very low level, and so usually one version of dump is required for each type of file system.

Since `dump` and `restore` deal with the file system at a low level they are rarely used for software distribution. For the same reason it is not possible to copy a directory with dump and restore unless you want to copy the whole file system.

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

10.4 Bits and Bobs

In times of emergency your backups may be essential. For this reason you should have a backup you can work from even if the machine has had a disk failure. This will mean that you may need:

- A spare hard disk.
- The install media (and instructions) for Unix.
- A full, recent backup.
- A print out of what it on the backup, and how to restore it.

You may not need to back up all file systems either for example if `/tmp` is on a disk by itself. Check your tapes work from time to time by checking you can restore random files. Finally remember that your tapes contain all the data on your system, so if someone steal a tape they may as well have carted the whole system off.

11 References

- “Essential System Administration”, second edition, Æleen Frisch, O’Reilly & Associates.
- “Unix System Administration Handbook”, second edition, Evi Nemeth et al., Prentice Hall.

The first book is a readable and contains a suggested reading list. The second book is a useful reference. The O’Reilly & Associates “nutshell handbooks” are highly praised as no nonsense guides to computing.