Chapter 4

System Administration

This chapter is an overview to Linux system administration, including a number of advanced features which aren't necessarily for system administrators only. Just as every dog has its day, every system has its administrator, and running the system is a very important and sometimes time-consuming job, even if you're the only user on your system.

We have tried to cover here the most important things about system administration you need to know when you use Linux, in sufficient detail to get you comfortably started. In order to keep it short and sweet, we have only covered the very basics, and have skipped many an important detail. You should read the *Linux System Administrator's Guide* if you are serious about running Linux. It will help you understand better how things work, and how they hang together. At least skim through it so that you know what it contains and know what kind of help you can expect from it.

4.1 About Root, Hats, and the Feeling of Power

As you know, UNIX differentiates between different users, so that what they do to each other and to the system can be regulated (one wouldn't want anybody to be able to read one's love letters, for instance). Each user is given an **account**, which includes a username, home directory, and so on. In addition to accounts given to real people, there are special system-defined accounts which have special privileges. The most important of these is the **root account**, for the username `root`.

4.1.1 The `root` account

Ordinary users are generally restricted so that they can't do harm to anybody else on the system, just to themselves. File permissions on the system are arranged such that normal users aren't allowed to delete or modify files in directories shared by all users (such as `/bin` and `/usr/bin`. Most users also protect their own files with the appropriate file permissions so that other users can't access or modify those files.

There are no such restrictions on `root`. The user `root` can read, modify, or delete any file on the system, change permissions and ownerships on any file, and run special programs, such as those which partition the drive or create filesystems. The basic idea is that the person or persons who run and take care of the system logs in as `root` whenever it is necessary to perform tasks that cannot be executed as a normal user. Because `root` can do anything, it is easy to make mistakes that have catastrophic consequences when logged in using this account.

For example, as a normal user, if you inadvertently attempt to delete all of the files in `/etc`, the system will not permit you to do so. However, when logged in as `root`, the system won't complain at all. It is very easy to trash your system when using `root`. The best way to prevent accidents is to:

- Sit on your hands before you press `return` on a command which may cause damage. For example, if you're about to clean out a directory, before hitting `return`, re-read the entire command and make sure that it is correct.
- Don't get accustomed to using `root`. The more comfortable you are in the role of the `root` user, the more likely you are to confuse your privileges with those of a normal user. For example, you might *think* that you're logged in as `larry`, when you're really logged in as `root`.
- Use a different prompt for the `root` account. You should change `root`'s `.bashrc` or `.login` file to set the shell prompt to something other than your regular user prompt. For example, many people use the character "`$`" in prompts for regular users, and reserve the character "`#`" for the `root` user prompt.
- Only login as `root` when absolutely necessary. And, as soon as you're finished with your work as `root`, log out. The less you use the `root` account, the less likely you'll be to do damage on your system.

Of course, there is a breed of UNIX hackers out there who use `root` for virtually everything. But every one of them has, at some point, made a silly mistake as `root` and trashed the system. The general rule is, until you're familiar with the lack of restrictions on `root`, and are comfortable using the system without such restrictions, login as `root` sparingly.

Of course, everyone makes mistakes. Linus Torvalds himself once accidentally deleted the entire kernel directory tree on his system. Hours of work were lost forever. Fortunately, however, because of his knowledge of the filesystem code, he was able to reboot the system and reconstruct the directory tree by hand on disk.

Put another way, if you picture using the `root` account as wearing a special magic hat that gives you lots of power, so that you can, by waving your hand, destroy entire cities, it is a good idea to be a bit careful about what you do with your hands. Since it is easy to move your hand in a destructive way by accident, it is not a good idea to wear the magic hat when it is not needed, despite the wonderful feeling.

4.1.2 Abusing the system

Along with the feeling of power comes the tendency to do harm. This is one of the grey areas of UNIX system administration, but everyone goes through it at some point in time. Most users of

UNIX systems never have the ability to wield this power—on university and business UNIX systems, only the highly-paid and highly-qualified system administrators ever login as `root`. In fact, at many such institutions, the `root` password is a highly guarded secret: it is treated as the Holy Grail of the institution. A large amount of hubbub is made about logging in as `root`; it is portrayed as a wise and fearsome power, given only to an exclusive cabal.

This kind of attitude towards the `root` account is, quite simply, the kind of thing which breeds malice and contempt. Because `root` is so fluffed-up, when some users have their first opportunity to login as `root` (either on a Linux system or elsewhere), the tendency is to use `root`'s privileges in a harmful manner. I have known so-called "system administrators" who read other user's mail, delete user's files without warning, and generally behave like children when given such a powerful "toy".

Because `root` has such privilege on the system, it takes a certain amount of maturity and self-control to use the account as it was intended—to run the system. There is an unspoken code of honor which exists between the system administrator and the users on the system. How would you feel if your system administrator was reading your e-mail or looking over your files? There is still no strong legal precedent for electronic privacy on time-sharing computer systems. On UNIX systems, the `root` user has the ability to forego all security and privacy mechanisms on the system. It is important that the system administrator develop a trusting relationship with the users on the system. I can't stress that enough.

4.1.3 Dealing with users

UNIX security is rather lax by design. Security on the system was an afterthought—the system was originally developed in an environment where users intruding upon other users was simply unheard of. Because of this, even with security measures, there is still the ability for normal users to do harm.

System administrators can take two stances when dealing with abusive users: they can be either paranoid or trusting. The paranoid system administrator usually causes more harm than he or she prevents. One of my favorite sayings is, "Never attribute to malice anything which can be attributed to stupidity." Put another way, most users don't have the ability or knowledge to do real harm on the system. 90% of the time, when a user is causing trouble on the system (by, for instance, filling up the user partition with large files, or running multiple instances of a large program), the user is simply unaware that what he or she is doing is a problem. I have come down on users who were causing a great deal of trouble, but they were simply acting out of ignorance—not malice.

When you deal with users who are causing potential trouble, don't be accusative. The old rule of "innocent until proven guilty" still holds. It is best to simply talk to the user, and question about the trouble, instead of causing a confrontation. The last thing you want to do is be on the user's bad side. This will raise a lot of suspicion about you—the system administrator—running the system correctly. If a user believes that you distrust or dislike them, they might accuse you of deleting files or breaching privacy on the system. This is certainly not the kind of position that you want to be in.

If you do find that a user has been attempting to "crack" the system, or was intentionally doing harm to the system, don't return the malicious behavior with malice of your own. Instead, simply provide a warning—but be flexible. In many cases, you may catch a user "in the act" of doing harm to the system—give them a warning. Tell them not to let it happen again. However, if you *do* catch them causing harm again, be absolutely sure that it is intentional. I can't even begin to describe the number of cases where it appeared as though a user was causing trouble, when in fact it was either an accident or a fault of my own.

4.1.4 Setting the rules

The best way to run a system is not with an iron fist. That may be how you run the military, but UNIX was not designed for such discipline. It makes sense to lay down a simple and flexible set of guidelines for users—but remember, the fewer rules you have, the less chance there is of breaking them. Even if your rules for using the system are perfectly reasonable and clear, users will always at times break these rules without intending to. This is especially true in the case of new UNIX users, who are just learning the ropes of the system. It's not patently obvious, for example, that you shouldn't download a gigabyte of files and mail them to everyone on the system. Users need help understanding the rules, and why they are there.

If you do specify usage guidelines for your system, make sure that the reason behind a particular guideline is made clear. If you don't, then users will find all sorts of creative ways to get around the rule, and not know that they are in fact breaking it.

4.1.5 What it all means

We can't tell you how to run your system to the last detail. Most of the philosophy depends on how you're using the system. If you have many users, things are much different than if you only have a few users, or if you're the only user on the system. However, it's always a good idea—in any situation—to understand what being the system administrator really means.

Being the system administrator doesn't make you a UNIX wizard. There are many system admins out there who know very little about UNIX. Likewise, there are many "normal" users out there who know more about UNIX than any system administrator could. Also, being the system administrator does not allow you to use malice against your users. Just because the system gives you the privilege to mess with user files does not mean that you have any right to do so.

Lastly, being the system administrator is really not a big deal. It doesn't matter if your system is a little 386 or a Cray supercomputer. Running the system is the same, regardless. Knowing the `root` password isn't going to earn you money or fame. It will allow you to maintain the system, and keep it running. That's it.

4.2 Booting the System

There are several ways to boot the system, either from floppy or from the hard drive.

4.2.1 Using a boot floppy

Many people boot Linux using a "boot floppy" which contains a copy of the Linux kernel. This kernel has the Linux root partition coded into it, so it will know where to look on the hard drive for the root filesystem. (The `rdev` command can be used to set the root partition in the kernel image; see below.) This is the type of floppy created by SLS during installation, for example.

To create your own boot floppy, first locate the kernel image on your hard disk. It should be in the file `/Image` or `/etc/Image`. Some installations use the file `/vmlinux` for the kernel.

You may instead have a compressed kernel. A compressed kernel uncompresses itself into memory at boot time, and takes up much less space on the hard drive. If you have a compressed kernel, it may be found in the file `/zImage` or `/etc/zImage`.

Once you know where the kernel is, set the root device in the kernel image to the name of your Linux root partition with the `rdev` command. The format of the command is

`rdev` ⟨*kernel-name*⟩ ⟨*root-device*⟩

where ⟨*kernel-name*⟩ is the name of the kernel image, and ⟨*root-device*⟩ is the name of the Linux root partition. For example, to set the root device in the kernel `/etc/Image` to `/dev/hda2`, use the command

`#` *rdev /etc/Image /dev/hda2*

`rdev` can set other options in the kernel as well, such as the default SVGA mode to use at boot time. Just use "`rdev -?`" to get a help message.

After setting the root device, you can simply copy the kernel image to the floppy. Whenever copying data a floppy, it's a good idea to MS-DOS format the floppy first. This lays down the sector and track information on the floppy, so it can be detected as either high or low density.

For example, to copy the kernel in the file `/etc/Image` to the floppy in `/etc/fd0`, use the command

`#` *cp /etc/Image /dev/fd0*

This floppy should now boot Linux.

4.2.2 Using LILO

Another method of booting is to use LILO, a program which resides in the boot sector of your hard disk. This program is executed when the system is booted from the hard disk, and can automatically boot up Linux from a kernel image stored on the hard drive itself.

LILO can also be used as a first-stage boot loader for several operating systems, allowing you to select at boot time which operating system (such as Linux or MS-DOS) to boot. When you boot using LILO, the default operating system is booted unless you press `ctrl`, `alt`, or `shift` during the bootup sequence. If you press any of these keys, you will be provided with a boot prompt, at which you type the name of the operating system to boot (such as "`linux`" or "`msdos`"). If you press `tab` at the boot prompt, a listing of available operating systems will be provided.

LILO is located in the directory `/etc/lilo` (if you have it). The easy way to install LILO is to edit the configuration file, `/etc/lilo/config`, and then run the command

```
# /etc/lilo/lilo
```

The LILO configuration file contains a "stanza" for each operating system that you want to boot. The best way to demonstrate this is with an example LILO config file. The below setup is for a system which has a Linux root partition on `/dev/hda1`, and an MS-DOS partition on `/dev/hda2`.

```
# Tell LILO to modify the boot record on /dev/hda (the first
# non-SCSI hard drive). If you boot from a drive other than /dev/hda,
# change the following line.
boot = /dev/hda

# Name of the boot loader. No reason to modify this unless you're doing
# some serious hacking on LILO.
install = /etc/lilo/boot.b

# Have LILO perform some optimization.
compact

# Stanza for Linux root partition on /dev/hda1.
image = /etc/Image    # Location of kernel
   label = linux      # Name of OS (for the LILO boot menu)
   root = /dev/hda1   # Location of root partition
   vga = ask          # Tell kernel to ask for SVGA modes at boot time

# Stanza for MSDOS partition on /dev/hda2.
other = /dev/hda2     # Location of partition
   table = /dev/hda   # Location of partition table for /dev/hda2
   label = msdos      # Name of OS (for boot menu)
```

The first operating system stanza in the config file will be the default OS for LILO to boot. You can select another OS to boot at the LILO boot prompt, as discussed above.

The program `/etc/lilo/QuickInst` will ask you questions about your setup and generate a LILO config file for you.

Remember that every time you update the kernel image on disk, you should rerun `/etc/lilo/lilo` in order for the changes to be reflected on the boot sector of your drive.

Also note that if you use the "`root =`" line, above, there's no reason to use `rdev` to set the root partition in the kernel image. LILO sets it for you at boot time.

The Linux FAQ (see Appendix A) provides more information on LILO, including how to use LILO to boot with OS/2's Boot Manager.

4.3 Shutting Down

Shutting down a Linux system is a bit tricky. Remember that you should never just turn off the power or hit the reset switch while the system is running. The kernel keeps track of disk I/O in memory buffers. If you reboot the system without giving the kernel the chance to write its buffers to disk, you can corrupt your filesystems.

Other precautions are taken at shutdown time as well. All processes are sent a signal, which allows them to die gracefully (writing and closing all files, and so on). Filesystems are unmounted for safety. If you wish, the system can also alert users that the system is going down and give them a change to log off.

The easiest way to shutdown is with the `shutdown` command. The format of the command is

`shutdown` ⟨*time*⟩ ⟨*warning-message*⟩

The ⟨*time*⟩ argument is the time to shutdown the system (in the format *hh:mm:ss*), and ⟨*warning-message*⟩ is a message displayed on all user's terminals before shutdown. Alternately, you can specify the ⟨*time*⟩ as "`now`", to shutdown immediately. The `-r` option may be given to `shutdown` to reboot the system after shutting down.

For example, to shutdown the system at 8:00pm, use the command

```
# shutdown -r 20:00
```

The command `halt` may be used to force an immediate shutdown, without any warning messages or grace period. `halt` is useful if you're the only one using the system, and want to shut down the system and turn it off.

◇ Don't turn off the power or reboot the system until you see the message:

```
The system is halted
```

It is very important that you shutdown the system "cleanly" using the `shutdown` or `halt` commands. On some systems, pressing `ctrl-alt-del` will be trapped and cause a `shutdown`; on other systems, however, using the "Vulcan nerve pinch" will reboot the system immediately and may cause disaster.

4.4 Managing Users

Whether or not you have many users on your system, it's important to understand the aspects of user management under Linux. Even if you're the only user, you should presumably have a separate account for yourself (an account other than `root` to do most of your work).

Each person using the system should have his or her own account. It is seldom a good idea to have several people share the same account. Not only is security an issue, but accounts are used to uniquely identify users to the system. You need to be able to keep track of who is doing what.

4.4.1 User management concepts

The system keeps track of a number of pieces of information about each user. They are summarized below.

username The username is the unique identifier given to every user on the system. Examples of usernames are `larry`, `karl`, and `mdw`. Letters and digits may be used, as well as the characters "_" (underscore) and "." (period). Usernames are usually limited to 8 characters in length.

user ID The user ID, or UID, is a unique number given to every user on the system. The system usually keeps track of information by UID, not username.

group ID The group ID, or GID, is the ID of the user's default group. In Section 3.9 we discussed group permissions; each user belongs to one or more groups defined by the system administrator. More about this below.

password The system also stores the user's encrypted password. The `passwd` command is used to set and change user passwords.

full name The user's "real name" or "full name" is stored along with the username. For example, the user `schmoj` may have the name "Joe Schmo" in real life.

home directory
The home directory is the directory in which the user is initially placed at login time. Every user should have his or her own home directory, usually found under `/home`.

login shell The user's login shell is the shell which is started for the user at login time. Examples are `/bin/bash` and `/bin/tcsh`.

The file `/etc/passwd` contains this information about users. Each line in the file contains information about a single user; the format of each line is

```
username:encrypted password:UID:GID:full name:home directory:login shell
```

An example might be:

```
kiwi:Xv8Q981g71oKK:102:100:Laura Poole:/home/kiwi:/bin/bash
```

As we can see, the first field, "`kiwi`", is the username.

The next field, "`Xv8Q981g71oKK`", is the encrypted password. Passwords are not stored on the system in any human-readable format. The password is encrypted using itself as the secret key. In other words, you need to know the password to decrypt it. This form of encryption is fairly secure.

Some systems use "shadow password" in which password information is relegated to the file `/etc/shadow`. Because `/etc/passwd` is world-readable, `/etc/shadow` provides some degree of extra security because it is not. Shadow password provides some other features such as password expiration and so on; we will not go into these features here.

The third field, "102", is the UID. This must be unique for each user. The fourth field, "100", is the GID. This user belongs to the group numbered 100. Group information, like user information, is stored in the file /etc/group. See Section 4.4.5 for more information.

The fifth field is the user's full name, "Laura Poole". The last two fields are the user's home directory (/home/kiwi) and login shell (/bin/bash), respectively. It is not required that the user's home directory be given the same name as the username. It does help identify the directory, however.

4.4.2 Adding users

When adding a user, there are several steps to be taken. First, the user must be given an entry in /etc/passwd, with a unique username and UID. The GID, fullname, and other information must be specified. The user's home directory must be created, and the permissions on the directory set so that the user owns the directory. Shell initialization files must be provided in the new home directory and other system-wide configuration must be done (for example, setting up a spool for incoming e-mail for the new user).

While it is not difficult to add users by hand (I do), when you are running a system with many users it is easy to forget something. The easiest way to add users is to use an interactive program which asks you for the required information and updates all of the system files automatically. The name of this program is useradd or adduser, depending on what software was installed. The man pages for these commands should be fairly self-explanatory.

4.4.3 Deleting users

Similarly, deleting users can be accomplished with the commands userdel or deluser depending on what software was installed on the system.

If you'd like to temporarily "disable" a user from logging into the system (without deleting the user's account), you can simply prepend an asterisk ("*") to the password field in /etc/passwd. For example, changing kiwi's /etc/passwd entry to

```
kiwi:*Xv8Q981g71oKK:102:100:Laura Poole:/home/kiwi:/bin/bash
```

will restrict kiwi from logging in.

4.4.4 Setting user attributes

After you have created a user, you may need to change attributes for that user, such as home directory or password. The easiest way to do this is to change the values directly in /etc/passwd. To set a user's password, use the passwd command. For example,

```
# passwd larry
```

will change larry's password. Only root may change other user's password in this manner. Users can change their own passwords with passwd as well.

On some systems, the commands chfn and chsh will be available to allow users to set their own fullname and login shell attributes. If not, they will have to ask the system administrator to change these attributes for them.

4.4.5 Groups

As we have mentioned, each user belongs to one or more groups. The only real importance of group relationships pertains to file permissions, as you'll recall from Section 3.9, each file has a "group ownership" and a set of group permissions which defines how users in that group may access the file.

There are several system-defined groups such as bin, mail, and sys. Users should not belong to any of these groups; they are used for system file permissions. Instead, users should belong to an individual group such as users. If you want to be cute, you can maintain several groups of users such as student, staff, and faculty.

The file /etc/group contains information about groups. The format of each line is

```
group name:password:GID:other members
```

Some example groups might be:

```
root:*:0:
users:*:100:mdw,larry
guest:*:200:
other:*:250:kiwi
```

The first group, root, is a special system group reserved for the root account. The next group, users, is for regular users. It has a GID of 100. The users mdw and larry are given access to this group. Remember that in /etc/passwd each user was given a default GID. However, users may belong to more than one group, by adding their usernames to other group lines in /etc/group. The groups command lists what groups you are given access to.

The third group, guest, is for guest users, and other is for "other" users. The user kiwi is given access to this group as well.

As you can see, the "password" field of /etc/group is rarely used. It is sometimes used to set a password on group access. This is seldom necessary. To protect users from changing into priveleged groups (with the newgroup command), set the password field to "*".

The commands addgroup or groupadd may be used to add groups to your system. Usually, it's easier just to add entries in /etc/group yourself, as no other configuration needs to be done to add a group. To delete a group, simply delete its entry in /etc/group.

4.5 Archiving and Compressing Files

Before we can talk about backups, we need to introduce the tools used to archive software on UNIX systems.

4.5.1 Using `tar`

The `tar` command is most often used to archive software.

The format of the `tar` command is

`tar` ⟨*options*⟩ ⟨*file1*⟩ ⟨*file2*⟩ ...⟨*fileN*⟩

where ⟨*options*⟩ is the list of commands and options for `tar`, and ⟨*file1*⟩ through ⟨*fileN*⟩ is the list of files to add or extract from the archive.

For example, the command

```
# tar cvf backup.tar /etc
```

would pack all of the files in `/etc` into the tar archive `backup.tar`. The first argument to `tar`—"`cvf`"—is the `tar` "command". "`c`" tells `tar` to create a new archive file. The "`v`" option forces `tar` into verbose mode—printing each filename as it is archived. The "`f`" option tells `tar` that the next argument—`backup.tar`—is the name of the archive to create. The rest of the arguments to `tar` are the file and directory names to add to the archive.

The command

```
# tar xvf backup.tar
```

will extract the tar file `backup.tar` in the current directory. This can sometimes be dangerous—when extracting files from a tar file, old files are overwritten.

Furthermore, before extracting tar files it is important to know where the files should be unpacked. For example, let's say you archived the following files: `/etc/hosts`, `/etc/group`, and `/etc/passwd`. If you use the command

```
# tar cvf backup.tar /etc/hosts /etc/group /etc/passwd
```

the directory name `/etc/` is added to the beginning of each filename. In order to extract the files to the correct location, you would need to use the following commands:

```
# cd /
# tar xvf backup.tar
```

because files are extracted with the pathname saved in the archive file.

If, however, you archived the files with the command

```
# cd /etc
# tar cvf hosts group passwd
```

the directory name is not saved in the archive file. Therefore, you would need to "`cd /etc`" before extracting the files. As you can see, how the tar file is created makes a large difference in where you extract it. The command

```
# tar tvf backup.tar
```

may be used to display an "index" of the tar file before unpacking it. In this way you can see what directory the filenames in the archive are stored relative to, and can extract the archive from the correct location.

4.5.2 gzip and compress

Unlike archiving programs for MS-DOS, `tar` does not automatically compress files as it archives them. Therefore, if you are archiving two 1-megabyte files, the resulting tar file will be two megabytes in size. The `gzip` command may be used to compress a file (the file to compress need not be a tar file). The command

```
# gzip -9 backup.tar
```

will compress `backup.tar` and leave you with `backup.tar.gz`, the compressed version of the file. The `-9` switch tells `gzip` to use the highest compression factor.

The `gunzip` command may be used to uncompress a gzipped file. Equivalently, you may use "`gzip -d`".

`gzip` is a relatively new tool in the UNIX community. For many years, the `compress` command was used instead. However, because of several factors[1], `compress` is being phased out.

`compress`ed files end in the extension `.Z`. For example, `backup.tar.Z` is the compressed version of `backup.tar`, while `backup.tar.gz` is the gzipped version[2]. The `uncompress` command is used to expand a `compress`ed file; `gunzip` knows how to handle `compress`ed files as well.

4.5.3 Putting them together

Therefore, to archive a group of files and compress the result, you can use the commands:

```
# tar cvf backup.tar /etc
# gzip -9 backup.tar
```

The result will be `backup.tar.gz`. To unpack this file, use the reverse set of commands:

[1] These factors include a software patent dispute against the `compress` algorithm and the fact that `gzip` is much more efficient than `compress`.

[2] To add further confusion, for some time the extension `.z` (lowercase "z") was used for gzipped files. The official `gzip` extension is now `.gz`.

gunzip backup.tar.gz
tar xvf backup.tar

Of course always make sure that you are in the correct directory before unpacking a tar file.

You can use some UNIX cleverness to do all of this on one command line, as in the following:

tar cvf - /etc | gzip -9c > backup.tar.gz

Here, we are sending the tar file to "-", which stands for `tar`'s standard output. This is piped to `gzip`, which compresses the incoming tar file, and the result is saved in `backup.tar.gz`. The `-c` option to `gzip` tells `gzip` to send its output to stdout, which is redirected to `backup.tar.gz`.

A single command used to unpack this archive would be:

gunzip -c backup.tar.gz | tar xvf -

Again, `gunzip` uncompresses the contents of `backup.tar.gz` and sends the resulting tar file to stdout. This is piped to `tar`, which reads "-", this time referring to `tar`'s standard input.

Happily, the `tar` command also includes the `-z` option to automatically compress/uncompress files on the fly. However, it does this as per the `compress` algorithm—it does not use `gzip`. (Please note that the newest versions of GNU `tar` do in fact use `gzip` when using the `-z` option. However, if you use a `tar` binary from the Stone Age, like me, then don't expect `-z` to use `gzip` compression.)

For example, the command

tar cvfz backup.tar.Z /etc

is equivalent to

tar cvf backup.tar /etc
compress backup.tar

Just as the command

tar xvfz backup.tar.Z

may be used instead of

uncompress backup.tar.Z
tar xvf backup.tar

Refer to the man pages for `tar` and `gzip` for more information.

4.6 Using Floppies and Making Backups

Floppies are usually used as backup media. If you don't have a tape drive connected to your system, floppy disks can be used (although they are slower and somewhat less reliable).

You may also use floppies to hold individual filesystems—in this way, you can **mount** the floppy to access the data on it.

4.6.1 Using floppies for backups

The easiest way to make a backup using floppies is with `tar`. The command

```
# tar cvfzM /dev/fd0 /
```

will make a complete backup of your system using the floppy drive `/dev/fd0`. The "`M`" option to `tar` allows the backup to be a multivolume backup; that is, when one floppy is full, `tar` will prompt for the next. The command

```
# tar xvfzM /dev/fd0
```

can be used to restore the complete backup. This method can also be used if you have a tape drive (`/dev/rmt0`) connected to your system.

Note that using the method, you must settle for the `compress` algorithm; `tar` doesn't use `gzip` with the "`z`" option. Several other programs exist for making multiple-volume backups; the `backflops` program on `tsx-11.mit.edu` may come in handy.

Making a complete backup of the system can be time- and resource-consuming. Most system administrators use a incremental backup policy, in which every month a complete backup is taken, and every week only those files which have been modified in the last week are backed up. In this case, if you trash your system in the middle of the month, you can simply restore the last full monthly backup, and then restore the last weekly backups as needed.

The `find` command can be useful in locating files which have changed since a certain date. Several scripts for managing incremental backups can be found on `sunsite.unc.edu`.

4.6.2 Using floppies as filesystems

You can create a filesystem on a floppy just as you would on a hard drive partition. For example,

```
# mke2fs /dev/fd0 1440
```

creates a filesystem on the floppy in `/dev/fd0`. The size of the filesystem must correspond to the size of the floppy. High-density 3.5" disks are 1.44 megabytes, or 1440 blocks, in size. High-density 5.25" disks are 1200 blocks.

In order to access the floppy, you must **mount** the filesystem contained on it. The command

```
# mount -t ext2 /dev/fd0 /mnt
```

will mount the floppy in `/dev/fd0` on the directory `/mnt`. Now, all of the files on the floppy will appear under `/mnt` on your drive. The "`-t ext2`" specifies an ext2fs filesystem type. If you created another type of filesystem on the floppy, you'll need to specify its type to the `mount` command.

The "mount point" (the directory where you're mounting the filesystem) needs to exist when you use the `mount` command. If it doesn't exist, simply create it with `mkdir`.

See Section 4.8 for more information on filesystems, mounting, and mount points.

◇ Note that any I/O to the floppy is buffered just as hard disk I/O is. If you change data on the floppy, you may not see the drive light come on until the kernel flushes its I/O buffers. It's important that you not remove a floppy before you unmount it; this can be done with the command

```
# umount /dev/fd0
```

Do not simply switch floppies as you would on an MS-DOS system; whenever you change floppies, `umount` the first one and `mount` the next.

4.7 Upgrading and Installing New Software

Another duty of the system administrator is upgrading and installing new software.

The Linux community is very dynamic. New kernel releases come out every few weeks, and other software is updated almost as often. Because of this, new Linux users often feel the need to upgrade their systems constantly to keep up the the rapidly changing pace. Not only is this unnecessary, it's a waste of time: to keep up with all of the changes in the Linux world, you would be spending all of your time upgrading and none of your time using the system.

So, when should you upgrade? Some people feel that you should upgrade when a new distribution release is made—for example, when SLS comes out with a new version. Many Linux users completely reinstall their system with the newest SLS release every time. This, also, is a waste of time. In general, changes to SLS releases are small. Downloading and reinstalling 30 disks when only 10% of the software has been actually modified is, of course, pointless.

The best way to upgrade your system is to do it by hand: only upgrade those software packages which you know that you should upgrade. This scares a lot of people: they want to know what to upgrade, and how, and what will break if they don't upgrade. In order to be successful with Linux, it's important to overcome your fears of "doing it yourself"— which is what Linux is all about. In fact, once you have your system working and all software correctly configured, reinstalling with the newest SLS release will no doubt wipe all of your configuration and things will be broken again, just as they were when you first installed your system. Setting yourself back in this manner is unnecessary—all that is needed is some know-how about upgrading your system, and how to do it right.

You'll find that when you upgrade one component of your system, other things should not break. For example, most of the software on my system is left over from an ancient 0.96 MCC Interim installation. Yet, I run the newest version of the kernel and libraries with this software with no problem. For the most part, senselessly upgrading to "keep up with the trend" is not important at all. This isn't MS-DOS or Microsoft Windows. There is no important reason to run the newest version of all of the software. If you find that you would like or need features in a new version, then upgrade. If not, then don't. In other words, only upgrade what you have to, and when you have to. Don't just upgrade for the sake of upgrading. That will waste a lot of time and effort trying to keep up.

The most important software to upgrade on your system is the kernel, the libraries, and the `gcc` compiler. These are the three essential parts of your system, and in some cases they all depend on each other for everything to work successfully. Most of the other software on your system does not need to be upgraded periodically.

4.7.1 Upgrading the kernel

Upgrading the kernel is simply a matter of getting the sources and compiling them yourself. You must compile the kernel yourself in order to enable or disable certain features, as well as to ensure that the kernel will be optimized to run on your machine. The process is quite painless.

The kernel sources may be retrieved from any of the Linux FTP sites (see Section C for a list). On `sunsite.unc.edu`, for instance, the kernel sources are found in `/pub/Linux/kernel`. Kernel versions are numbered using a version number and a patchlevel. For example, kernel version 0.99 patchlevel 11 is usually written as `0.99.pl11`, or just `0.99.11`.

The kernel sources are released as a gzipped tar file[3]. For example, the file containing the 0.99.pl11 kernel sources is `linux-0.99.11.tar.gz`.

Unpack this tar file from the directory `/usr/src`; it creates the directory `/usr/src/linux` which contains the kernel sources. You should delete or rename your existing `/usr/src/linux` before unpacking the new version.

Once the sources are unpacked, you need to make sure that two symbolic links in `/usr/include` are correct. To create these links, use the commands

```
# ln -sf /usr/src/linux/include/linux /usr/include/linux
# ln -sf /usr/src/linux/include/asm /usr/include/asm
```

Once you have created these links once, there is no reason to create them again when you install the next version of the kernel sources. (See Section 3.10 for more about symbolic links.)

Note that in order to compile the kernel, you must have the `gcc` and `g++` C and C++ compilers installed on your system. You may need to have the most recent versions of these compilers: see Section 4.7.3, below, for more information.

To compile the kernel, first `cd` to `/usr/src/linux`. Run the command `make config`. This

[3] Often, a patch file is also released for the current kernel version which allows you to patch your current kernel sources from the last patchlevel to the current one (using the program `patch`). In most cases, however, it's usually easier to install the entire new version of the kernel sources

command will prompt you for a number of configuration options, such as what filesystem types you wish to include in the new kernel.

Next, edit `/usr/src/linux/Makefile`. Be sure that the definition for `ROOT_DEV` is correct—it defines the device used as the root filesystem at boot time. The usual definition is

```
ROOT_DEV = CURRENT
```

Unless you are changing your root filesystem device, there is no reason to change this.

Next, run the command `make dep` to fix all of the source dependencies. This is a very important step.

Finally, you're ready to compile the kernel. The command `make Image` will compile the kernel and leave the new kernel image in the file `/usr/src/linux/Image`. Alternately, the command `make zImage` will compile a compressed kernel image, which uncompresses itself at boot time and uses less drive space.

Once you have the kernel compiled, you need to either copy it to a boot floppy (with a command such as "`cp Image /dev/fd0`") or install it using LILO to boot from your hard drive. See Section 4.2.2 for more information.

4.7.2 Upgrading the libraries

As mentioned before, most of the software on the system is compiled to use shared libraries, which contain common subroutines shared among different programs.

If you see the message

```
Incompatible library version
```

when attempting to run a program, then you need to upgrade to the version of the libraries which the program requires. Libraries are back-compatible; that is, a program compiled to use an older version of the libraries should work with the new version of the libraries installed. However, the reverse is not true.

The newest version of the libraries can be found on the Linux FTP sites. On `sunsite.unc.edu`, they are located in `/pub/Linux/GCC`. The "release" files there should explain what files you need to download and how to install them. Briefly, you should get the files `image-`*version*`.tar.gz` and `inc-`*version*`.tar.gz` where *version* is the version of the libraries to install, such as `4.4.1`. These are gzipped tar files; the `image` file contains the library images to install in `/lib` and `/usr/lib`. The `inc` file contains include files to install in `/usr/include`

The `release-`*version*`.tar.gz` should explain the installation procedure in detail (the exact instructions vary for each release). In general you need to install the library `.a` and `.sa` files in `/usr/lib`. These are the libraries used at compilation time.

In addition, the shared library image files, `libc.so.`*version* are installed in `/lib`. These are the shared library images loaded at runtime by programs using the libraries. Each library has a symbolic link using the major version number of the library in `/lib`.

For example, the `libc` library version 4.4.1 has a major version number of `4`. The file containing the library is `libc.so.4.4.1`. A symbolic link of the name `libc.so.4` is also in `/lib` pointing to this file. You need to change this symbolic link when upgrading the libraries. For example, when upgrading from `libc.so.4.4` to `libc.so.4.4.1`, you need to change the symbolic link to point to the new version.

◇ It is very important that you change the symbolic link in one step, as given below. If you somehow delete the symbolic link `libc.so.4`, then programs which depend on the link (including basic utilities like `ls` and `cat`) will stop working. Use the following command to update the symbolic link `libc.so.4` to point to the file `libc.so.4.4.1`:

```
# ln -sf /lib/libc.so.4.4.1 /lib/libc.so.4
```

You also need to change the symbolic link `libm.so.`*version* in the same manner. If you are upgrading to a different version of the libraries substitute to appropriate filenames above. The library release notice should explain the details. (See Section 3.10 for more information about symbolic links.)

4.7.3 Upgrading gcc

The `gcc` C and C++ compiler is used to compile software on your system, most importantly the kernel. The newest version of `gcc` is found on the Linux FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/Linux/GCC` (along with the libraries). There should be a `release` file for the `gcc` distribution detailing what files you need to download and how to install them.

4.7.4 Upgrading other software

Upgrading other software is usually just a matter of downloading the appropriate files and installing them. Most software for Linux is distributed at gzipped tar files, including either sources or binaries or both. If binaries are not included in the release, you may need to compile them yourself; usually, this means typing `make` in the directory where the sources are held.

Reading the USENET newsgroup `comp.os.linux.announce` for announcements of new software releases is the easiest way to find out about new software. Whenever you are looking for software on an FTP site, downloading the `ls-lR` index file from the FTP site and using `grep` to find the files in question is the easiest way to locate software. If you have `archie` available to you, it can be of assistance as well[4]. See Appendix A for more details.

One handy source of Linux software is the SLS distribution disk images. Each disk contains a number of `.tgz` files which are simply gzipped tar files. Instead of downloading the disks, you can download the desired `.tgz` files from the SLS directories on the FTP site and install them directly. If you run the SLS distribution, the `sysinstall` command can be used to automatically load and install a complete series of disks. For example, the command

```
# sysinstall -series t
```

[4] If you don't have `archie`, you can telnet to an `archie` server such as `archie.rutgers.edu`, login as "`archie`" and use the command "`help`"

will install the entire SLS `t` series of disks. Of course, most of the time you may not download or wish to install an entire series, in which case you'll need to unpack the `.tgz` files by hand.

Again, it's usually not a good idea to upgrade by reinstalling with the newest version of SLS, or another distribution. SLS in particular was not designed to be upgradeable. If you reinstall in this way, you will no doubt wreck your current installation, including user directories and all of your customized configuration. The best way to upgrade software is piecewise; that is, if there is a program that you use often that has a new version, upgrade it. Otherwise, don't bother. Rule of thumb: If it ain't broke, don't fix it. If your current software works, there's no reason to upgrade.

4.8 Managing Filesystems

Another task of the system administrator is taking care of filesystems. Most of this job entails periodically checking the filesystems for damage or corrupted files; many systems automatically check the filesystems at boot time.

4.8.1 Mounting filesystems

First, a few concepts about filesystems. Before a filesystem is accessible to the system, it must be **mounted** on some directory. For example, if you have a filesystem on a floppy, you must mount it under some directory, say `/mnt`, in order to access the files on it (see Section 4.6.2). After mounting the filesystem on that directory, all of the files in the filesystem appear in that directory. In the case of the floppy, the files on the floppy will appear in the directory `/mnt`. After unmounting the floppy, the directory `/mnt` will be empty.

The same is true of filesystems on the hard drive. The system automatically mounts filesystems on your hard drive for you at bootup time. The so-called "root filesystem" is mounted on the directory `/`. If you have a separate filesystem for `/usr`, for example, it is mounted on `/usr`. If you only have a root filesystem, all files (including those in `/usr`) exist on that filesystem.

The command `mount` is used to mount a filesystem. The command

```
mount -av
```

is executed from the file `/etc/rc` (which is the system initialization file executed at boot time; see Section 4.10.1). The `mount -av` command obtains information on filesystems and mount points from the file `/etc/fstab`. An example `fstab` file appears below.

```
# device        directory       type      options
/dev/hda2        /              ext2      defaults
/dev/hda3        /usr           ext2      defaults
/dev/hda4        none           swap      sw
/proc            /proc          proc      none
```

The first field is the device—the name of the partition to mount. The second field is the mount point. The third field is the filesystem type—such as `ext2` (for ext2fs) or `minix` (for Minix filesystems). The last field contains `mount` options—usually, this is set to "`defaults`".

As you can see, swap partitions are included in `/etc/fstab` as well. They have a mount directory of `none`, and type `swap`. The `swapon -a` command, executed from `/etc/rc` as well, is used to enable swapping on all swap devices listed in `/etc/fstab`.

The `fstab` file contains one special entry—for the `/proc` filesystem. As mentioned in Section 3.11.1, the `/proc` filesystem is used to store information about system processes, available memory, and so on. If `/proc` is not mounted, commands such as `ps` will not work.

◇ The `mount` command may only be used by root. This is to ensure security on the system; you wouldn't want regular users mounting and unmounting filesystems on a whim. There are several software packages available which allow regular users to mount and unmount filesystems (floppies in particular) without compromising system security.

The `mount -av` command actually mounts all filesystems other than the root filesystem (in the table above, `/dev/hda2`). The root filesystem is automatically mounted at boot time by the kernel.

Instead of using `mount -av`, you can mount a filesystem by hand. The command

```
# mount -t ext2 /dev/hda3 /usr
```

is equivalent to mounting the filesystem with the entry `/dev/hda3` in the `fstab` example file above.

In general, you should never have to mount or unmount filesystems by hand. The `mount -av` command in `/etc/rc` takes care of mounting the filesystems at boot time. Filesystems are automatically unmounted by the `shutdown` or `halt` commands before bringing the system down.

4.8.2 Checking filesystems

It is usually a good idea to check your filesystems for damage or corrupt files every now and then. Some systems automatically check their filesystems at boot time (with the appropriate commands in `/etc/rc`).

The command used to check a filesystem depends on the type of the filesystem in question. For ext2fs filesystems (the most commonly used type), this command is `e2fsck`. For example, the command

```
# e2fsck -av /dev/hda2
```

will check the ext2fs filesystem on `/dev/hda2` and automatically correct any errors.

It is usually a good idea to unmount a filesystem before checking it. For example, the command

```
# umount /dev/hda2
```

will unmount the filesystem on `/dev/hda2`, after which you can check it. The one exception is that you cannot unmount the root filesystem. In order to check the root filesystem when it's unmounted, you should use a maintenance boot/root diskette (see Section 4.11.1). You also cannot unmount a filesystem if any of the files in it are "busy"—that is, being used by a running process. For example, you cannot unmount a filesystem if any user's current working directory is on that filesystem. You will receive a "`Device busy`" error if you attempt to unmount a filesystem which is in use.

Other filesystem types use different forms of the `e2fsck` command, such as `efsck` and `xfsck`. On some systems, you can simply use the command `fsck`, which will determine the filesystem type and execute the appropriate command.

◇ It is important that you reboot your system immediately after checking a filesystem if any corrections were made to that filesystem. For example, if `e2fsck` reports that it corrected any errors with the filesystem, you should immediately `shutdown -r` in order to reboot the system. This is to allow the system to re-sync its information about the filesystem when `e2fsck` modifies it.

The `/proc` filesystem never needs to be checked in this manner. `/proc` is a memory filesystem, managed directly by the kernel.

4.9 Using a swap file

Instead of reserving an individual partition for swap space, you can use a file. However, to do so you'll need install the Linux software and get everything going *before* you create the swap file.

If you have a Linux system installed, you can use the following commands to create a swap file. Below, we're going to create a swap file of size 8208 blocks (about 8 megs).

```
# dd if=/dev/zero of=/swap bs=1024 count=8208
```

This command creates the swap file itself. Replace the "`count=`" with the size of the swap file in blocks.

```
# mkswap /swap 8208
```

This command will initialize the swapfile; again, replace the name and size of the swapfile with the appropriate values.

```
# /etc/sync
# swapon /swap
```

Now we are swapping on the file `/swap` which we have created, after syncing, which ensures that the file has been written to disk.

The one major drawback to using a swapfile in this manner is that all access to the swap file is done through the filesystem. This means that the blocks which make up the swap file may not be contiguous. Therefore, performance may not be as great as using a swap partition, for which blocks are always contiguous and I/O requests are done directly to the device.

Another drawback in using a swapfile is the chance to corrupt your filesystem data—when using large swap files, there is the chance that you can corrupt your filesystem if something goes wrong. Keeping your filesystems and swap partitions separate will prevent this from happening.

Using a swap file can be very useful if you have a temporary need for more swap space. For example, if you're compiling a large program and would like to speed things up somewhat, you can temporarily create a swap file and use it in addition to your regular swap space.

To get rid of a swap file, first use `swapoff`, as in

```
# swapoff /swap
```

And you can safely delete the file.

```
# rm /swap
```

Remember that each swap file (or partition) may be as large as 16 megabytes, but you may use up to 8 swap files or partitions on your system.

4.10 Miscellaneous Tasks

Believe it or not, there are a number of housekeeping tasks for the system administrator which don't fall into any major category.

4.10.1 System startup files

When the system boots, a number of scripts are executed automatically by the system before any user logs in. Here is a description of what happens.

At bootup time, the kernel spawns the process `/etc/init`. `init` is a program which reads its configuration file, `/etc/inittab`, and spawns other processes based on the contents of this file. One of the important processes started from `inittab` is the `/etc/getty` process started on each virtual console. The `getty` process grabs the VC for use, and starts a `login` process on the VC. This allows you to login on each VC; if `/etc/inittab` does not contain a `getty` process for a certain VC, you will not be able to login on that VC.

Another process executed from `/etc/inittab` is `/etc/rc`, the main system initialization file. This file is a simple shell script which executes any initialization commands needed at boot time, such as mounting the filesystems (see Section 4.8) and initializing swap space.

Your system may execute other initialization scripts as well, such as `/etc/rc.local`. `/etc/rc.local` usually contains initialization commands specific to your own system, such as setting the hostname (see the next section). `rc.local` may be started from `/etc/rc` or from `/etc/inittab` directly.

4.10.2 Setting the hostname

In a networked environment, the hostname is used to uniquely identify a particular machine, while in a standalone environment the hostname just gives the system personality and charm. It's like naming a pet: you can always address to your dog as "The dog," but it's much more interesting to assign the dog a name such as Spot or Woofie.

Setting the system's hostname is a simple matter of using the `hostname` command. If you are on a network, your hostname should be the full hostname of your machine, such as `goober.norelco.com`. If you are not on a network of any kind, you can choose an arbitrary host and domainname, such as `loomer.vpizza.com`, `shoop.nowhere.edu`, or `floof.org`.

When setting the hostname, the hostname must appear in the file `/etc/hosts`, which assigns an IP address to each host. Even if your machine is not on a network, you should include your own hostname in `/etc/hosts`.

For example, if you are not on a TCP/IP network, and your hostname is `floof.org`, simply include the following line in `/etc/hosts`:

```
127.0.0.1        floof.org localhost
```

This assigns your hostname, `floof.org`, to the loopback address 127.0.0.1 (used if you're not on a network). The `localhost` alias is also assigned to this address.

If you are on a TCP/IP network, however, your real IP address and hostname should appear in `/etc/hosts`. For example, if your hostname is `goober.norelco.com`, and your IP address is 128.253.154.32, add the following line to `/etc/hosts`:

```
128.253.154.32       goober.norelco.com
```

If your hostname does not appear in `/etc/hosts`, you will not be able to set it.

To set your hostname, simply use the `hostname` command. For example, the command

```
# hostname -S goober.norelco.com
```

sets the hostname to `goober.norelco.com`. In most cases, the `hostname` command is executed from one of the system startup files, such as `/etc/rc` or `/etc/rc.local`. Edit these two files and change the `hostname` command found there to set your own hostname; upon rebooting the system the hostname will be set to the new value.

4.11 What To Do In An Emergency

On some occasions, the system administrator will be faced with the problem of recovering from a complete disaster, such as forgetting the root password or trashing filesystems. The best advice is, *don't panic.* Everyone makes stupid mistakes—that's the best way to learn about system administration: the hard way.

Linux is not an unstable version of UNIX. In fact, I have had fewer problems with system hangs than with commercial versions of UNIX on many platforms. Linux also benefits from a strong complement of wizards who can help you get out of a bind.

The first step in investigating any problem is to attempt to fix it yourself. Poke around, see how things work. Too much of the time, a system administrator will post a desperate plea for help before looking into the problem at all. Most of the time, you'll find that fixing problems yourself is actually very easy. It is the path to guruhood.

There are very few cases where reinstalling the system from scratch is necessary. Many new users accidentally delete some essential system file, and immediately reach for the installation disks. This is not a good idea. Before taking such drastic measures, investigate the problem and ask others to help fix things up. In almost all cases, you can recover your system from a maintenance diskette.

4.11.1 Recovering using a maintenance diskette

One indispensable tool for the system administrator is the so called "boot/root disk"—a floppy which can be booted for a complete Linux system, independent of your hard drive. Boot/root disks are actually very simple—you create a root filesystem on the floppy, place all of the necessary utilities on it, and install LILO and a bootable kernel on the floppy. Another technique is to use one floppy for the kernel and another for the root filesystem. In any case, the result is the same: you are running a Linux system completely from floppy.

The canonical example of a boot/root disk is the Slackware boot disks[5]. These diskettes contain a bootable kernel and a root filesystem, all on floppy. They are intended to be used to install the Slackware distribution, but come in very handy when doing system maintenance.

The H.J Lu boot/root disk, available from `/pub/Linux/GCC` on `sunsite.unc.edu`, is another example of such a maintenance disk. Or, if you're ambitious, you can create your own. In most cases, however, using a pre-made boot/root disk is much easier and will probably be more complete.

Using a boot/root disk is very simple. Just boot the disk on your system, and login as `root` (usually no password). In order to access the files on your hard drive, you will need to mount your filesystems by hand. For example, the command

```
# mount -t ext2 /dev/hda2 /mnt
```

will mount an ext2fs filesystem on `/dev/hda2` under `/mnt`. Remember that `/` is now on the boot/root disk itself; you need to mount your hard drive filesystems under some directory in order to access the files. Therefore, `/etc/passwd` on your hard drive is now `/mnt/etc/passwd` if you mount your root filesystem on `/mnt`.

[5] See Section 2.1.1 for information on downloading these from the Internet. For this procedure, you don't need to download the entire Slackware release—only the boot and root diskettes.

4.11.2 Fixing the root password

If you forget your root password, no problem. Just boot the boot/root disk, mount your root filesystem on `/mnt`, and blank out the password field for `root` in `/mnt/etc/passwd`, as so:

```
root::0:0:root:/:/bin/sh
```

Now `root` has no password; when you reboot from the hard drive you should be able to login as `root` and reset the password using `passwd`.

Aren't you glad you learned how to use `vi`? On your boot/root disk, other editors such as Emacs probably aren't available, but `vi` should be.

4.11.3 Fixing trashed filesystems

If you somehow trash your filesystems, you can run `e2fsck` (if you use the ext2fs filesystem type, that is) to correct any damaged data on the filesystems from floppy. Other filesystem types use different forms of the `fsck` command; see Section 4.8 for details.

When checking your filesystems from floppy, it's best for the filesystems to not be mounted.

One common cause of filesystem damage is superblock corruption. The *superblock* is the "header" of the filesystem that contains information on the filesystem status, size, free blocks, and so forth. If you corrupt your superblock (for example, by accidentally writing data directly to the filesystem's partition), the system may not recognize the filesystem at all. Any attempt to mount the filesystem could fail, and `e2fsck` won't be able to fix the problem.

Happily, the *ext2fs* filesystem type saves copies of the superblock at "block group" boundaries on the drive—usually, every 8K blocks. In order to tell `e2fsck` to use a copy of the superblock, you can use a command such as

*e2fsck -b 8193* ⟨*partition*⟩

where ⟨*partition*⟩ is the partition on which the filesystem resides. The `-b 8193` option tells `e2fsck` to use the copy of the superblock stored at block 8193 in the filesystem.

4.11.4 Recovering lost files

If you accidentally deleted important files on your system, there's no way to "undelete" them. However, you can copy the relevant files from the floppy to your hard drive. For example, if you deleted `/bin/login` on your system (which allows you to login), simply boot the boot/root floppy, mount the root filesystem on `/mnt`, and use the command

*cp -a /bin/login /mnt/bin/login*

The `-a` option tells `cp` to preserve the permissions on the file(s) being copied.

Of course, if the files you deleted weren't essential system files which have counterparts on the boot/root floppy, you're out of luck. If you made backups, you can always restore from them.

4.11.5 Fixing trashed libraries

If you accidentally trashed your libraries or symbolic links in `/lib`, more than likely commands which depended on those libraries will no longer run (see Section 4.7.2). The easiest solution is to boot your boot/root floppy, mount your root filesystem, and fix the libraries in `/mnt/lib`.