Debian Tutorial

Havoc Pennington <hp@debian.org>

16 March 1999

Copyright Notice

Contents

Chapter 1

About this manual

This is the Debian Tutorial. It is aimed at readers who are new to Debian GNU/Linux. It assumes no prior knowledge of GNU/Linux or other Unix-like systems, but it does assume very basic general knowledge about computers and hardware (you should know what the basic parts of a computer are, and what one might use a computer to do).

This manual is meant to be read in order; each chapter assumes some knowledge of prior chapters, though you may find it useful to skip around.

There is also a Debian Reference Guide planned, which will be more comprehensive but less introductory.

This tutorial assumes that you have already installed and configured Debian GNU/Linux according to the installation manual (which is incomplete as of this writing). However, you may want to look over the tutorial before you install, in order to learn more about Debian.

In general this tutorial tries to explain the reasons for things, and help you understand what's going on inside the system. The idea is to empower you to solve new problems and get the most out of your computer. Thus there's plenty of theory, history, and fun facts thrown in with the "How To" aspects of the manual.

Please send comments about this manual to the Debian Documentation Project mailing list `<debian-doc@lists.debian.org>`. We're especially interested in whether it was helpful to you, and how we could make it better. If you get confused while reading, or notice that we use a term without explaining it first, please email us.

Please, DO NOT send the authors technical questions about Debian, as there are other forums for that. See 'Getting help from a person' on page 28. Only send mail regarding the manual itself to the above address.

To find the latest version of this manual, go to `http://www.debian.org/doc/` and follow the links.

1.1 Acknowledgements

Many people have helped with this manual.

The biggest thanks go to Larry Greenfield and his Linux User's Guide, which formed the basis for the manual. The Linux User's Guide is a part of the Linux Documentation Project.

Many thanks to those who have helped me edit the manual; they have made it far, far better. If you thought this manual was pleasant to read, send your thanks to Thalia Hooker and Day Irmiter.

Thanks to Richard Stallman of the Free Software Foundation for advice, editing, and offering to publish the text.

Thanks to James Treacy for letting me borrow some of his writings from the Debian web site.

Thanks to everyone who has written parts of the manual: Craig Sawyer wrote about shells, Ole Tetlie is writing about programming, Oliver Elphick contributed discussion of some basic utilities, Ivan E. Moore II contributed the discussion of PPP.

Many people have submitted patches and comments, including Eric Fischer and Mike Touloumtzis.

Many thanks to Ardo van Rangelrooij for getting things started and maintaining the DebianDoc DTD used to write the manual.

Of course, it's impossible to thank the hundreds of Debian developers and thousands of free software authors that gave us something to write about and use.

Thanks also to anyone I left out, since I'm sure I screwed this up. I hope no one will take offense — please email me and let me know if your name should be here.

Chapter 2

Introduction

2.1 What is Debian?

Debian is a free operating system (OS) for your computer. An operating system is the set of basic programs and utilities that make your computer run. At the core of an operating system is the *kernel*. The kernel is the most fundamental program on the computer: it does all the basic housekeeping and lets you start other programs. Debian uses the *Linux* kernel, a completely free piece of software started by Linus Torvalds and supported by (probably over 1000) programmers worldwide. A large part of the basic tools that fill out the operating system come from the (GNU project (`http://www.gnu.org`)), and these tools are also free. Of course, what people want is application software: programs to help them get what they want to do done, from editing documents to running a business to playing games to writing more software. Debian comes with over 1000 *packages* (precompiled software bundled up in a nice format for easy installation on your machine) — all of it free.

It's a bit like a tower. At the base is Linux. On top of that are all the basic tools, mostly from GNU. Next is all the application software that you run on the computer: much of this is also from GNU. The Debian developers act as architects and coordinators — carefully organizing the system and fitting everything together into an integrated, stable operating system: Debian GNU/Linux.

2.1.1 What's an operating system, and what sort of operating system is Debian?

An operating system is the collection of software that makes a computer usable. It manages hardware devices and provides utilities and applications.

Debian GNU/Linux is based on the Unix operating system, which has a long history (see 'Unix History' on page 103). Debian is basically compatible with Unix, but adds a significant number of additional features.

The design philosophy of GNU/Linux (and Unix) is to distribute its functionality into small, multipurpose parts. That way, you can easily achieve new functionality and new features by combining the small parts (programs) in new ways. Debian is like an erector set; you can build all sorts of things with it.

When you're using an operating system, you want to minimize the amount of work you put into getting your job done. Debian supplies many tools that can help you, but only if you know what these tools do. Spending an hour trying to get something to work and then finally giving up isn't very productive. This manual will teach you about the core tools that make up Debian: what tools to use in what situations, and how to tie these various tools together.

2.1.2 Who creates Debian?

Debian is an all-volunteer internet development project. There are hundreds of volunteers working on it. Most are in charge of a small number of software packages and are intimately familiar with the software they package.

These volunteers work together by following a strict set of guidelines governing how packages are assembled. These guidelines are developed cooperatively in discussions on internet mailing lists and *internet relay chat* (IRC) forums.

2.2 What's free software?

When Debian developers and users speak of "free software", they refer to *freedom* rather than price. Debian is free in this sense: you are free to modify and redistribute it, and will always have access to the source code for this purpose. The Debian Free Software Guidelines (`http://www.debian.org/social_contract#guidelines`) describe in more details exactly what is meant by "free". The Free Software Foundation (`http://www.fsf.org`), originator of the GNU project, is another source of information. You can find a more detailed discussion of free software on the Debian web site (`http://www.debian.org/intro/free`).

Free software is sometimes called Open Source (R) software — Open Source is a certification mark. Since Open Source (R) is trademarked, only truly free software can call itself Open Source (R). You may encounter vendors who try to mislead you by claiming their software is "free", while in reality it has significant strings attached. The Open Source (R) trademark gives you some assurance that the software really is free software. 'Open Source software' is occasionally abbreviated 'OSS'.

You may be wondering: why would people spend hours of their own time to write software, carefully package it, and then give it all away? The answers are as varied as the people who contribute.

Many believe in sharing information and having the freedom to cooperate with one another, and feel that free software encourages this. There's a long tradition starting in the 1950s upholding these values, sometimes called the Hacker Ethic. (You can read more about it in Steven Levy's enjoyable book, *Hackers: Heroes of the Computer Revolution*.)

Others want to learn more about computers. More and more people are looking for ways to avoid the inflated price of commercial software. A growing crowd contribute as a thank you for all the great free software they've received from others.

Many in academia create free software to help get the results of their research into wider use. Businesses help maintain free software so they can have a say in how it develops — there's no quicker way to get a new feature than to implement it yourself or hire a consultant to do so! Business is also interested in greater reliability and the ability to choose between support vendors.

Still others see free software as a social good, democratizing access to information and preventing excessive centralization of the world's information infrastructure. Of course, a lot of us just find it great fun.

Debian is so committed to free software that we thought it would be useful if it was formalized in a document of some sort. Our Social Contract (`http://www.debian.org/social_contract`) promises that Debian will always be 100% free software. When you download a package from the Debian main distribution, you can be sure it meets our Free Software Guidelines.

Although Debian believes in free software, there are cases where people want or need to put proprietary software on their machine. Whenever possible Debian will support this; though proprietary software is not included in the main distribution, it is sometimes available on the ftp site in the `non-free` directory, and there are a growing number of packages whose sole job is to install proprietary software we are not allowed to distribute ourselves.

It is important to distinguish *commercial* software from *proprietary* software. Proprietary software is non-free software, while commercial software is software sold for money. Debian permits commercial software to be a part of the main distribution, but not proprietary software. Remember that the phrase "free software" does not refer to price; it is quite possible to sell free software. For more clarification of the terminology, see `http://www.opensource.org` or `http://www.fsf.org/philosophy/categories.html`.

2.3 How to Read This Book

The best way to learn about almost any computer program is at your computer. Most people find that reading a book without using the program isn't beneficial. The best way to learn Unix and GNU/Linux is by using them. Use GNU/Linux for everything you can. Experiment. Don't be afraid — it's *possible* to mess things up, but you can always reinstall. Keep backups and have fun!

Debian isn't as intuitively obvious as some other operating systems. Thus, you will probably end up reading at least the first few chapters. GNU/Linux is like a race car, a master chef's kitchen, or a classic novel; its power and complexity make it difficult to approach at first, but far more rewarding in the long run.

The suggested way to learn is to read a little, then play a little. Keep playing until you're comfortable with the concepts, and then start skipping around in the book. You'll find a variety of topics are covered, some of which you might find interesting and some of which you'll find boring. After a while, you should feel confident enough to start using commands without knowing exactly what they do. This is a good thing.

A helpful thing to know: if you ever mistakenly type a command, or don't know how to exit a program, C-c (the Ctrl key and the lowercase letter c held simultaneously) will often stop the program.

2.4 The Linux Documentation Project

This manual borrows heavily from the Linux Documentation Project's *Linux User's Guide*, by Larry Greenfield. Thanks Larry! That project has a number of other excellent manuals, many of them targetted at more experienced users and system administrators. The LDP also maintains the Linux HOWTOs, an invaluable resource you should become familiar with. You can find the LDP at their homepage (`http://sunsite.unc.edu/LDP/`).

Chapter 3

Getting started

So you've just finished installing Debian! Congratulations. Dive right in and start learning to use it.

As a part of the install process, you should have figured out how to boot the Debian system (with a special floppy disk, by simply turning your computer on, or by holding down the `Alt` key at the LILO prompt and selecting Linux).

3.1 A multiuser, multitasking operating system

As we mentioned earlier ('What is Debian?' on page 3), the design of Debian GNU/Linux comes from the Unix operating system. Unlike common desktop OS's such as DOS, Windows, and MacOS, Unix is usually found on large servers and *multiuser* systems.

This means that Debian has features those other OS's lack. It allows a large number of people to use the same computer at once, as long as each user has their own *terminal*

[1].

To permit many users to work at once, Debian must permit many programs and applications to run simultaneously. This feature is called *multitasking*.

Much of the complexity (and power) of Unix-like systems stems from these two features. For example, the system must have a way to keep users from accidentally deleting each other's files, and it has to coordinate the many programs running at once, e.g. to ensure that they don't all use the hard drive at the same time.

If you keep in mind what Debian was originally designed to do, many aspects of it will make a lot more sense. You'll learn to take advantage of the power of these features.

[1](A terminal is just a keyboard and a screen, connected to the computer through the network, over a modem, or directly. Your keyboard and monitor form a terminal which is directly attached to the computer: this special terminal is often called the *console*.)

3.2 Logging in

To use Debian you must identify yourself to the system. This is so it knows who you are, what you have permission to do, and what your preferences are.

To this end, you have a *user name* or *login* — if you installed Debian yourself, you should have been asked to give such a name during installation. If you are logging on to a system administered by someone else, you'll have to ask them for an account on the system, and a corresponding username.

You also have a password, so no one else can pretend to be you. If you don't have a password, anyone can log on to your computer from the Internet, and do bad things (see 'A few words on security' on page 96). If you're worried about security, you should have a password.

Many people prefer to trust others not to do anything malicious with their account; hopefully your work environment doesn't encourage paranoia. This is a perfectly reasonable attitude; it depends on your personal priorities, and your environment. Obviously a home system does not need to be as secure as a military installation. Debian allows you to be as secure or as insecure as you like.

When you start Debian, you'll see a *prompt*; a request from the computer for some information. In this case, the prompt is `login:`.

You should enter your username, and when requested, your password. The password does not appear on the screen as you type it — that's so no one can look over your shoulder and see what it is. Press `Enter` after both the username and the password. If you type your username or password incorrectly, you'll have to start over.

If you do it correctly, you'll see a brief message and then a `$` prompt. The `$` is printed by a special program called the *shell*, and is thus called a *shell prompt*: this is where you give commands to the system.

Try entering the command `whoami` now. There is a *cursor* to the right of the shell prompt. Your cursor is a small underscore or rectangle which indicates where you're typing; it should move as you type. Always press `RET` (the Enter or Return key) when you're done typing a shell command.

`whoami` tells your username. You'll then get a new shell prompt.

For the rest of the manual, when we say to enter a command, you should type it at the shell prompt and press the `RET` key. On some keyboards, this key is labeled `Enter` and on others it's `Return`. Same key, different name.

When you're done working, you may want to log out of the system. To exit the shell, enter the `exit` command. Keep in mind that if you remain logged in, someone could come along and use your account. Hopefully you can trust those in your office or home not to do this; but if you do not trust your environment, you should be certain to log out when you leave.

3.3 Keys

Before going on, it's important to be familiar with the conventions in this manual for describing key commands.

When you should simultaneously hold down multiple keys, a notation like `C-a` will be used. This means "hold the control key, and type lowercase letter a." Other abbreviations include the Alt key, `A`, and the Meta key `M`. Some keyboards have both Alt and Meta; most home computers have only Alt, but the Alt key behaves like a Meta key. So if you have no Meta key, try the Alt key instead.

Keys like Alt and Meta are called *modifier* keys because they change the meaning of standard keys like the letter A. Sometimes you need to hold down more than one modifier; for example, `M-C-a` means to simultaneously press Meta, Ctrl, and lowercase a.

Some keys have a special notation; for example, `RET` (Return/Enter), `DEL` (Delete or sometimes Backspace), `ESC` (Escape). These should be fairly self-explanatory.

Spaces instead of hyphens mean to type the keys in sequence. So, for example, `C-a x RET` means to simultaneously type Control and lowercase a, followed by the letter x, followed by pressing Return.

3.4 Command history and editing the command line

Whatever you type after the shell prompt before pressing `RET` is called a *command line* — it's a line of text that commands the computer to do something. The Debian default shell offers several features to make entering command lines easy.

You can scroll up to previous commands to run them again, or modify them slightly and *then* run them again. Try this: enter any command, such as `whoami`; then press the up arrow key. The `whoami` command will reappear at the prompt. You can then press `RET` to run `whoami` a second time.

If you've entered several commands, you can keep pressing the up arrow key to go back through them. This feature is handy if you're doing the same thing several times, or if you type a command incorrectly and want to go back to fix it. You can press the down arrow key to move in the other direction, toward your more recent commands. If there are no more commands to move to, the computer will beep.

You can also move around on the command line to make changes. The easiest way is with the left and right arrow keys — try typing `whoasmi` instead of `whoami`, then use the left arrow key to move back to the `s`. You can erase the `s` with the Backspace or Delete keys.

There are more advanced features as well (no need to memorize them all now, though). Try typing `C-a`. This moves you to the beginning of the line. `C-k` (the `k` stands for "kill") deletes until the end of the line — try it from the middle of the command line. Using `C-a` followed by `C-k`, you can delete the entire command line. `C-y` pastes the last thing you killed, inserting it at the current cursor position (`y` stands for "yank," as in "yank it back"). `C-e` will move the cursor at the end of the command line.

Go ahead and play around with command line editing to get a feel for it. Experiment.

3.5 Logging in as root

Since Debian is a multiuser system, it's designed to keep any one user or program from breaking the entire system. The kernel will not allow normal users to change important system files. This means that things stay the way they're supposed to, safe from accidents, viruses, and even malicious pranks. Unlike other operating systems, Debian is safe from these threats. You won't need an antivirus program.

However, sometimes you need to change important system files — for example, you might want to install new software, or configure your network connection. To do so, you have to have greater powers than a normal user; you must become the *root user* (also called the *superuser*).

To become root, just log on with the username `root` and the root password, if you have it. Hopefully you remember the password from when you installed the system — if not, you have a problem. [2]

At many sites, only the system administrator has the root password, and only the system administrator can do the things that one must be root to do. If you're using your own personal computer, *you* are the system administrator, of course. If you don't have root privileges, you will have to rely on your system administrator to perform any tasks that require root privileges.

Sometimes you'll have the root password even on a shared corporate or educational server, because the sysadmin trusts you to use it properly. In that case, you'll be able to help administer the system and customize it for your needs. But you should be sure to use the password responsibly, respecting other users at all times.

If you have the password, try logging on as root now. Enter the `whoami` command to verify your identity. Then *log out immediately*. When you're root, the kernel will not protect you from yourself, because root has permission to do anything at all to the system. For example, you can type `rm -rf /` and erase your *entire system* in a few keystrokes. (Needless to say, you should *NOT* type this). Don't experiment while you're root. In fact, don't do anything as root, unless absolutely necessary. This isn't a matter of security, but rather of stability. Your system will run much better if it can keep you from making silly mistakes.

You may find the `su` command more convenient than logging in as root. `su` allows you to assume the identity of another user, usually root unless you specify someone else. (You can remember that `su` stands for Super User, though some say it stands for Set UserID.)

Try this:

1. Log on as yourself, i.e. not as root.
2. `whoami`

   Confirm your username.

[2]The solution to this problem is fairly technical. You have to boot with a rescue disk, mount your normal root partition, and edit `/etc/passwd` to remove the old root password. Ask for help if this doesn't make sense to you (see 'Getting help from a person' on page 28).

3. `su`

   Enter the `su` command. It will prompt for a password; enter the root password. If you give the correct password, you should see a new shell prompt. By default, root's shell prompt is # rather than `$`.

4. `whoami`

   This should give "root" as your new username.

5. `exit`

   Exit the root shell. Your prompt will return to `$`.

6. `exit`

   Exit your own shell.

When you're doing system administration tasks, you should do as much as possible as yourself. Then `su`, do the part that requires root privileges, and `exit` to turn off privileges so you can no longer harm anything.

You can use `su` to assume the identity of any user on the system, not just root. To do this, type `su` *`user`* where *user* is the user you want to become. You'll have to know their password, of course, unless you're root at the time or they have no password.

3.6 Virtual consoles

The Linux kernel supports *virtual consoles*. These are a way of making your single screen and keyboard seem like multiple terminals, all connected to the same system. Thankfully, using virtual consoles is one of the simplest things about Debian: there are "hot keys" for switching among the consoles quickly. To try it, log in to your system, and type `A-F2` (simultaneously press the `Alt` key, and `F2`, that is, function key number 2).

You should find yourself at another login prompt. Don't panic: you are now on virtual console (VC) number 2! Log in here and do some things — more `whoami`'s or whatever — to confirm that this is a real login shell. Now you can return to virtual console number 1, with `A-F1`. Or you can move on to a *third* virtual console, in the obvious way (`A-F3`).

Debian comes with six virtual consoles enabled by default, accessed with the Alt key and function keys `F1-F6` (technically, there are more virtual consoles enabled, but only 6 of them allow you to log in. The others are used for the X Window System or other special purposes).

If you're using the X Window System, it will generally start up on the first unused virtual console — probably VC 7. Also, to switch from the X virtual console to one of the first six, you'll have to add `Ctrl` to the key sequence. So that's `C-A-F1` to get to VC 1. But you can go from a text VC to the X virtual console using only `Alt`. If you never leave X, you won't have to worry about this; X automatically switches you to its virtual console when it starts up.

Once you get used to them, virtual consoles will probably become an indispensable tool for getting many things done at once. (The X Window System serves much the same purpose, providing multiple windows rather than multiple consoles). You can run a different program on each VC or log on as root on one VC and as yourself on another. Or everyone in the family can use their own VC — this is especially handy if you use X, in which case you can run several X sessions at once, on different virtual consoles.

3.7 Shutting down

Do not just turn off the computer! You risk losing valuable data!

If you are the only user of your computer, you might want to turn the computer off when you're done with it.

3

Unlike most versions of DOS, it's a bad thing to just hit the power switch when you're done using the computer. It is also bad to reboot the machine (with the reset button) without first taking proper precautions. The Linux kernel, in order to improve performance, has a *disk cache*. This means it temporarily stores information meant for permanent storage in RAM: since memory is thousands of times faster than a disk, this makes many file operations move more quickly. Periodically, the information Linux has in memory is actually written to the disk. This is called *syncing*. In order to turn off or reboot the computer, you'll have to tell the computer to clear everything out of memory and put it in permanent storage.

To reboot, just type `reboot`, or press `C-A-DEL` (that's Control, Alt, and Delete).

To shut down, you'll have to be `root`. As root, just type the command `shutdown -h now`. This will go through the entire shutdown procedure, including the `sync` command which clears the disk cache as described above. When you see `System halted`, it's safe to turn off the computer. If you have Advanced Power Management (APM) support in your kernel and BIOS, the computer might shut itself off and save you the trouble. APM is common in laptops and is also found in certain desktop mainboards.

Some people find it simplest to shut down by typing `C-A-DEL` to reboot, then powering off the computer before the Linux kernel begins to reload. However, once the kernel begins to load, you have to wait for it to finish and then properly reboot or shutdown again.

[3] To avoid possibly weakening some hardware components, only turn off the computer when you're done for the day. Power up and power down are the two greatest contributors to wear and tear on computer components. Turning the computer on and off once a day is probably the best compromise between your electric bill and your computer's lifespan.

Chapter 4

The Basics

4.1 The command line and `man` pages

We've already discussed the *command line*, that is commands you type after the shell prompt. This section describes the structure of more complicated command lines.

A minimal command line contains just a command name, such as `whoami`. But other things are possible. For example, you might type:

```
man whoami
```

This command requests the online manual for the `whoami` program (you may have to press the space bar to scroll through the documentation, or press `q` to quit). A more complicated example:

```
 man -k Postscript
```

This command line has three parts. It begins with the command name, `man`. Then it has an *option* or *switch*, `-k`, followed by an *argument*, `Postscript`. Some people refer to everything except the command name as the *parameters* of the command. So, options and arguments are both parameters.

Options change the behavior of a command, switching on particular features or functionality. They usually have a `-` before them. The GNU utilities also have "long forms" for the options; the long form of `-k` is `--apropos`. Enter `man -h` or `man --help` to get a full list of options for the `man` command. Every command will have its own set of options, though most have `--help` and `--version` options. Some commands are bizarre; `tar`, for example, does not require the `-` before its options, for historical reasons.

Anything which isn't an option and isn't the command name is an *argument*. In this case, `Postscript`. Arguments can serve many purposes; most commonly, they are filenames that the command should operate on. In this case, `Postscript` is the word you want `man` to search for. In the case of `man whoami`, the argument was the command you wanted information about.

Breaking down the `man -k Postscript` command line:

- `man`, the command name, tells the computer to look at the manual pages. These provide documentation for commands. For example, `man whoami` will give you documentation on the `whoami` command.
- `-k`, the option, changes the behavior of `man`. Normally `man` expects a command name for an argument, such as `whoami`, and looks for documentation of that command. But with the `-k` or `--apropos` option, it expects the argument to be a keyword. It then gives a list of all manual pages with that keyword in their description.
- `Postscript` is the argument; since we used the `-k` option, it's the keyword to search for.
- `-k` and `Postscript` are both parameters.

Go ahead and type `man -k Postscript`, and you will see a list of all the manual pages on your system that have something to do with Postscript. If you haven't installed much software, you might see `Postscript:  nothing appropriate` instead.

4.1.1 Describing the command line

Note: This is a skippable section, if you want to move on.

There's a traditional concise way of describing command *syntax* [1] that you should know. For example, if you type `man man` to get the manual page about `man`, you'll see several syntax descriptions beginning with the command name `man`. One of them will look like this:

```
man -k [-M path] keyword ...
```

Anything in brackets (`[]`) is an optional unit. So you don't have to use the `-M` option, but if you do, you must use a `path` argument. You must use the `-k` option and the `keyword` argument. The `...` means that you could have more of whatever came before it, so you could look up several keywords.

Let's look at one of the more complex descriptions from the `man` manual page:

```
 man  [-c|-w|-tZT  device]  [-adhu7V] [-m system[,...]] [-L
      locale] [-p string] [-M path] [-P pager] [-r  prompt]  [-S
      list] [-e extension] [[section] page ...] ...
```

[1] *Syntax* means the correct ways to combine various options and arguments.

There's no need to go through all of this (and don't worry about what it all means), but do pay attention to the organization of the description.

First, clusters of options usually mean you can use one or more of them in different combinations, so `-adhu7V` means you can also use `-h`. However, you can't always use all combinations; this description doesn't make that clear. For example, `-h` is incompatible with other options, but you could do `man -du`. Unfortunately the description's format does not make this clear.

Second, the `|` symbol means "or". So you can use *either* the `-c`, the `-w`, *or* the `-tZT` options, followed by a `device` argument.

Third, notice that you can nest the brackets, since they indicate an optional *unit*. So if you have a `section`, you must also have a `page`, since `page` is not optional within the `[[section] page]` unit.

There's no need to memorize any of this, just refer to this section as you read documentation.

4.2 Files and Directories

4.2.1 Introduction to files

Files are a facility for storing and organizing information, analagous to paper documents. They're organized into *directories*, which are called *folders* on some other systems. Let's look at the organization of files on a Debian system:

`/` A simple `/` represents the root directory. All other files and directories are contained in the root directory. If you are coming from the DOS/Windows world, `/` is very similar to what `C:` is for DOS, that is the root of the filesystem. A notable difference between DOS and Linux however, is that DOS keeps several filesystems: `C:` (first hard disk), `A:` (first floppy disk), `D:` (either CD-ROM or second hard disk) while Linux has all its files organized above the same `/` root. See '`mount` and `/etc/fstab`' on page 73 for more details.

`/home/janeq` This is the home directory of user "janeq". Reading left to right, to get to this directory you start in the root directory, enter directory `home`, then enter directory `janeq`.

`/etc/X11/XF86Config` This is the configuration file for the X Window System. It resides in the `X11` subdirectory of the `/etc` directory. `/etc` is in turn a subdirectory of the root directory, `/`.

Things to note:

- Filenames are case sensitive. That is, `MYFILE` and `MyFile` are *different* files.
- The root directory is referred to as simply `/`. Don't confuse this "root" with the root user, the user on your system with "super powers."

- Every directory has a name which can contain any letters or symbols *except* `/`. The root directory is an exception; its name is `/` (pronounced "slash" or "the root directory") and it cannot be renamed. [2]
- Each file or directory is designated by a *fully-qualified filename*, *absolute filename*, or *path*, giving the sequence of directories which must be passed through to reach it. The three terms are synonymous. All absolute filenames begin with the `/` directory, and there's a `/` between each directory or file in the filename. The first `/` is the name of a directory, but the others are simply separators to distinguish the parts of the filename.

  The words used here can be confusing. Take the following example:

  ```
  /usr/share/keytables/us.map.gz
  ```

  This is a fully-qualified filename; some people call it a *path*. However, people will also refer to `us.map.gz` alone as a filename. [3]
- Directories are arranged in a tree structure. All absolute filenames start with the root directory. The root directory has a number of branches, such as `/etc` and `/usr`. These subdirectories in turn branch, into still more subdirectories, such as `/etc/init.d` and `/usr/local`. The whole thing together is called the "directory tree."

  You can think of an absolute filename as a route from the base of the tree (`/`) to the end of some branch (a file). You'll also hear people talk about the directory tree as if it were a *family* tree: thus subdirectories have "parents," and a path shows the complete ancestry of a file.

  There are also relative paths that begin somewhere other than the root directory. More on this later.
- There's no directory that corresponds to a physical device, such as your hard disk. This differs from DOS and Windows, where all paths begin with a device name such as `C:\`. The directory tree is meant to be an abstraction of the physical hardware, so you can use the system without knowing what the hardware is. All your files could be on one disk — or you could have 20 disks, some of them connected to a different computer elsewhere on the network. You can't tell just by looking at the directory tree, and nearly all commands work just the same way no matter what physical device(s) your files are really on.

Don't worry if all this isn't completely clear yet. There are many examples to come.

[2]While you *can* use almost any letters or symbols in a fi le name, in practice it's a bad idea. It is better to avoid any characters that often have special meanings on the command line, including: `{ } ( ) [ ] ' ‘ " \ / > < | ; ! # & ^ * % @`

Also avoid putting spaces in fi lenames. If you want to separate words in a name, good choices are the period, hyphen, and underscore. You could also capitalize each word, `LikeThis`.

[3]There is also another use for the word "path" . The intended meaning is usually clear from the context.

4.2.2 Using files: a tutorial

To use your system you'll have to know how to create, move, rename, and delete files and directories. This section describes how to do so with the standard Debian commands.

The best way to learn is to try things. As long as you aren't root (and haven't yet created any important personal files), there's nothing you can mess up too seriously. Jump in — type each of these commands at the prompt and press enter:

1. `pwd`

   One directory is always considered the *current working directory* for the shell you're using. You can view this directory with the `pwd` command, which stands for Print Working Directory. `pwd` prints the name of the directory you're working in — probably `/home/yourname`.

2. `ls`

   `ls` stands for "list", as in "list files". When you type `ls`, the system displays a list of all the files in your current working directory. If you've just installed Debian, your home directory may well be empty. If your working directory is empty, `ls` produces no output, since there are no files to list.

3. `cd /`

   `cd` means Change Directory. In this case, you've asked to change to the root directory.

4. `pwd`

   Verify that you're working in the root directory.

5. `ls`

   See what's in `/`.

6. `cd`

   Typing `cd` with no arguments selects your home directory as the current working directory — `/home/yourname`. Try `pwd` to verify this.

Before continuing, you should know that there are actually two different kinds of filename. Some of them begin with `/`, the root directory, such as `/etc/profile`. These are called *absolute* filenames because they refer to the same file no matter what your current directory is. The other kind of filename is *relative*.

Two directory names are used *only* in relative filenames: `.` and `..`. The directory `.` refers to the current directory and `..` is the parent directory. These are "shortcut" directories. They exist in *every* directory. Even the root directory has a parent directory — it's its own parent!

So filenames which include `.` or `..` are *relative*, because their meaning depends on the current directory. If I'm in `/usr/bin` and type `../etc`, then I'm referring to `/usr/etc`. If I'm in `/var` and type `../etc`, then I'm referring to `/etc`. Note that a filename without the root directory at the front implicitly has `./` at the front. So you can type `local/bin` or `./local/bin` and it means the same thing.

A final handy tip: the tilde `~` is equivalent to your home directory. So typing `cd ~` is the same as typing `cd` with no arguments. Also, you can type things like `cd ~/practice/mysubdirectory` to change to the directory `/home/yourname/practice/mysubdirectory`. In a similar way, `~vincent` is equivalent to the home directory of the user "vincent", which is probably something like `/home/vincent`; so `~vincent/docs/debian.ps` is equivalent to `/home/vincent/doc/debian.ps`.

Here are some more file commands to try out, now that you know about relative filenames. `cd` to your home directory before you begin.

1. `mkdir practice`

   In your home directory, make a directory called `practice`. You'll use this directory to try out some other commands. You might type `ls` to verify that your new directory exists.

2. `cd practice`

   Change directory to `practice`.

3. `mkdir mysubdirectory`

   Create a subdirectory of `practice`.

4. `cp /etc/profile .`

   `cp` is short for "copy." `/etc/profile` is just a random file on your system, don't worry about what it is for now. We've copied it to `.` — recall that `.` just means "the directory I'm in now", or the current working directory. So we've created a copy of `/etc/profile`, and put it in our `practice` directory. Try typing `ls` to verify that there's indeed a file called `profile` in your working directory, alongside the new `mysubdirectory`.

5. `more profile`

   View the contents of the file `profile`. `more` is used to view the contents of text files. It's called `more` because it shows a screenfull of the file at a time, and you press the space bar to see more. `more` will exit when you get to the end of the file, or when you type `q` (quit).

6. `more /etc/profile` Verify that the original looks just like the copy you made.

7. `mv profile mysubdirectory`

   `mv` stands for "move". We've moved the file `profile` from the current directory into the subdirectory we created earlier.

8. `ls`

   Verify that `profile` is no longer in the current directory.

9. `ls mysubdirectory`

   Verify that `profile` has moved to `mysubdirectory`.

10. `cd mysubdirectory`

    Change to the subdirectory.

11. `mv profile myprofile`

    Note that unlike some operating systems, there is no difference between moving a file and renaming it. Thus there's no separate `rename` command. Note that the second argument to `mv` can be a directory to move the file or directory into, or a new filename. `cp` works the same way.

    As usual, you can type `ls` to see the result of `mv`.

12. `mv myprofile ..`

    Just as `.` means "the directory I'm in now", `..` means "parent of the current directory", in this case the `practice` directory we created earlier. Use `ls` to verify that that's where `myprofile` is now.

13. `cd ..`

    Change directories to the parent directory — in this case `practice`, where you just put `myprofile`.

14. `rm myprofile`

    `rm` means "remove" — this deletes `myprofile`. Be careful! Deleting a file on a GNU/Linux system is *permanent* — there is no undelete. If you `rm` it, it's *gone, forever*. Be carefull! Deleting a file on a GNU/Linux system is *permanent* — there is no undelete. If you `rm` it, it's *gone, forever*.

15. `rmdir mysubdirectory`

    `rmdir` is just like `rm`, only it's for directories. Notice that `rmdir` only works on empty directories — if the directory contains files, you must delete those files first, or alternatively use `rm -r` in place of `rmdir`.

16. `cd ..`

    Move out of the current directory, and into its parent directory. Now you can type:

17. `rmdir practice`

    This will delete the last remnants of your practice session.

So now you know how to create, copy, move, rename, and delete files and directories. You also learned some shortcuts, like typing simply `cd` to jump to your home directory, and `.` and `..` to refer to the current directory and its parent, respectively. You should also remember the concept of the *root directory*, or `/`, and the alias ˜ for your home directory.

4.3 Processes

We mentioned before that GNU/Linux is a *multitasking* system. It can do many tasks at once. Each of these tasks is called a *process*. The best way to get a sense of this is to type `top` at the shell prompt. You'll get a list of processes, sorted according to how much of the computer's processing time they're using. The order will continuously change before your eyes. At the top of the display, there's some information about the system: how many users are logged in, how many total processes there are, how much memory you have and how much you're using.

In the far left column, you'll see the user owning each process. The far right column shows which command invoked the process. You'll probably notice that `top` itself, invoked by you, is near the top of the list (since anytime `top` checks on CPU usage, it will be active and using CPU to do the check).

Note all the commands ending in `d` — such as `kflushd` and `inetd` — the `d` stands for *daemon*[4]. A daemon is a non-interactive process, that is, it's run by the system and users never have to worry about it. Daemons provide services like internet connectivity, printing, or email.

Now press `u` and give `top` your user name when it asks. The `u` command asks to see only those processes belonging to you; it allows you to ignore all the daemons and whatever other people are doing. You might notice `bash`, the name of your shell. You'll pretty much always be running `bash`.

Note that column two of the `top` display shows you the *PID*, or Process IDentification number. Each process is assigned a unique PID. You can use the PID to control individual processes — more on that later. Another useful trick: type "?" to get a list of `top` commands.

You may wonder about the difference between a "process" and a "program" — in practice people use the terms interchangeably. Technically, the *program* is the set of instructions written by a programmer, and kept on disk. The *process* is the working instantiation of the program kept in memory by Linux. But it's not that important to keep the terms straight.

Much of your interaction with a computer involves controlling processes. You'll want to start them, stop them, and see what they're up to. Your primary tool for this is the *shell*.

4.4 The shell

The *shell* is a program that allows you to interact with your computer. It's called a shell because it provides an environment for you to work in — sort of a little electronic home for you as you compute. (Think hermit crab.)

The simplest function of the shell is to launch other programs. You type the name of the program you want to run, followed by the arguments you want, and the shell asks the system to run the program for you.

[4]daemon originally means Disks And Extensions MONitor

Of course, graphical windowing systems also fill this need. Technically, Windows 95 provides a graphical shell, and the X Window System is another kind of graphical shell — but "shell" is commonly used to mean "command line shell."

Needless to say, the hackers who work on shells aren't satisfied with simply launching commands. Your shell has a bewildering number of convenient features if you want to take advantage of them.

There are countless different shells available; most are based on either the *Bourne shell* or the *C shell*, two of the oldest shells. The original Bourne shell's program name is `sh` while `csh` is the C shell. Bourne shell variants include the Bourne Again Shell from the GNU project (`bash`, the Debian default), the Korn shell (`ksh`), and the Z shell (`zsh`). There is also `ash`, a traditionalist implementation of the Bourne shell. The most common C shell variant is `tcsh` (the `t` pays tribute to the TENEX and TOPS-20 operating systems, which inspired some of `tcsh`'s improvements over `csh`).

Bash is probably the best choice for new users. It is the default, and has all the features you're likely to need. But all the shells have loyal followings; if you want to experiment, install some different shell packages and change your shell with the `chsh` command. Just type `chsh`, supply a password when asked, and chose a shell. When you next log in, you'll be using the new shell.

4.5 Managing processes with Bash

Debian is a multitasking system, so you need a way to do more than one thing at once. Graphical environments like X provide a natural way to do this; they allow multiple windows on the screen at any one time. Naturally, Bash (or any other shell) provides similar facilities.

Earlier you used `top` to look at the different processes on the system. Your shell provides some convenient ways to keep track of only those processes you've started from the command line. Each command line starts a *job* (also called a *process group*) to be carried out by the shell. A job can consist of a single process or a set of processes in a *pipeline* — more on pipelines later.

Entering a command line will start a job. Try typing `man cp` and the `cp` manual page will appear on the screen. The shell will go into the background, and return when you finish reading the manual page (or type `q` to quit rather than scrolling through the whole thing).

But say you're reading the manual page, and you want to do something else for a minute. No problem. Type `C-z` while you're reading to *suspend* the currently foregrounded job, and put the shell in the foreground. When you suspend a job, Bash will first give you some information on it, and then a shell prompt. You will see something like this on the screen:

```
NAME
       cp - copy files

SYNOPSIS
```

```
       cp [options] source dest
       cp [options] source... directory
       Options:
       [-abdfilprsuvxPR]  [-S backup-suffix] [-V {numbered,exist
       ing,simple}]   [--backup]   [--no-dereference]   [--force]
       [--interactive] [--one-file-system] [--preserve] [--recur
       sive]  [--update]   [--verbose]   [--suffix=backup-suffix]
       [--version-control={numbered,existing,simple}] [--archive]
       [--parents] [--link]  [--symbolic-link]  [--help]  [--ver
       sion]

DESCRIPTION
--More--
[1]+  Stopped                 man cp
$
```

Note the last two lines. The next-to-last is the job information, and then you have a shell prompt.

Bash assigns a *job number* to each command line you run from the shell. This allows you to refer to the process easily. In this case, `man cp` is job number 1, displayed as `[1]`. The `+` means that this is the last job you had in the foreground. Bash also tells you the current state of the job — `Stopped` — and the job's command line.

There are many things you can do with jobs. With `man cp` still suspended, try this:

1. `man ls`

   Start a new job.

2. `C-z`

   Suspend the `man ls` job by pressing Control and lowercase `z`; you should see its job information.

3. `man mv`

   Start yet another job.

4. `C-z`

   Suspend it.

5. `jobs`

   Ask Bash for a display of current jobs:

   ```
   $ jobs
   [1]   Stopped                 man cp
   ```

```
[2]-  Stopped                 man ls
[3]+  Stopped                 man mv
$
```

Notice the `-` and `+`, denoting respectively the next-to-last and last foregrounded jobs.

6. `fg`

   Place the last foregrounded job (`man mv`, the one with the `+`) in the foreground again. If you press the spacebar, the man page will continue scrolling.

7. `C-z`

   Re-suspend `man mv`.

8. `fg %1`

   You can refer to any job by placing a `%` in front of its number. If you use `fg` without specifying a job, the last active one is assumed.

9. `C-z`

   Re-suspend `man cp`.

10. `kill %1`

    Kill off job 1. Bash will report the job information:

    ```
    $ kill %1
    [1]-  Terminated              man cp
    $
    ```

    Bash is only asking the job to quit, and sometimes a job will not want to do so. If the job doesn't terminate, you can add the `-9` option to kill to stop asking and start demanding. For example:

    ```
    $ kill -9 %1
    [1]-  Killed                  man mv
    $
    ```

    The `-9` option forcibly and unconditionally kills off the job. [5]

[5] In technical terms, `kill` simply sends a signal. By default it sends a signal which requests termination (`TERM`, or signal 15); but you can also specify a signal, and signal 9 (`KILL`) is the signal which forces termination. The command name `kill` is not necessarily appropriate to the signal sent; for example, sending the `TSTP` (terminal stop) signal suspends the process but allows it to be continued later.

11. `top`

    Bring the `top` display back up. Give the `u` command in `top` to see only your processes. Look in the right-hand column for the `man ls` and `man mv` commands. `man cp` won't be there since you killed it. `top` is showing you the system processes corresponding to your jobs; notice that the PID on the left of the screen does not correspond to the job number.

    You may not be able to find your processes because they're off the bottom of the screen; if you're using X, you can resize the `xterm` to solve this problem.

    Even these simple jobs actually consist of multiple processes, including the `man` process and the pager `more` which handles scrolling a page at a time. You may notice the `more` processes are also visible in `top`.

You can probably figure out how to clean up the remaining two jobs. You can either kill them (with the `kill` command) or foreground each one (with `fg`) and exit it. Remember that the `jobs` command will tell you the list existing jobs and their status.

One final note: the documentation for Bash is quite good, but it is found in the Info help system rather than the man pages. To read it, type `info bash`. See 'Using info' on page 27 for instructions on using the `info` program. Bash also contains a very good summary of its commands accessible by the `help` command. `help` displays a list of available topics; more informations about each of them being accessible with the command `help topicname`; Try to type

```
help cd
```

for example. This will give you details on the `-P` and `-L` arguments recognized by `cd`.

4.6 A few Bash features

This section mentions just a few of the most commonly used Bash features; for a more complete discussion see 'Using the shell' on page 30.

4.6.1 Tab Completion

The Bash shell can guess what filename or command you are trying to type, and automatically finish typing it for you. Just type the beginning of a command or filename, and press `TAB`. If Bash finds a single unique completion, it will finish the word and put a space after it. If it finds multiple possible completions, it will fill out the part all completions have in common and beep. You can then enter enough of the word to make it unique, and press `TAB` again. If it finds no completions, it will simply beep.

4.7 Managing your identity

Unix-like systems are multiuser, and so you have your own electronic identity as a user on the system. Type `finger` *`yourusername`* to have a look at some of the information about you that's publically available. To change the name and shell listed there, you can use the commands `chfn` and `chsh`. Only the superuser can change your login (username) and directory. You'll notice that it says "No plan" — a "plan" is just some information you can make available to others. To create a plan, put whatever information you want people to see in a file called `.plan` — to do this you'll use a text editor (see 'Creating and editing text files' on page 48). Then finger yourself to see your plan. Others can finger you to see your plan, and to check whether you've received new mail or read your mail.

Note that this finger information is available to the entire Internet by default. If you don't want this, read about configuring `inetd` and the file `/etc/services` — eventually the installation manual will describe this configuration, for now you might try the man pages, or just put nonsense in for your finger information.

Chapter 5

Reading documentation and getting help

5.1 Kinds of documentation

Unfortunately documentation on Unix-like systems is a little disorganized. On Debian, you can find documentation in at least the following places:

- `man` pages, read with the `man` command.
- `info` pages, read with the `info` command.
- The `/usr/doc/`*`package`* directories, where *package* is the name of the Debian package.
- `/usr/doc/HOWTO/` contains the Linux Documentation Project's HOWTO documents, if you've installed the Debian packages containing them.
- Many commands have a `-h` or `--help` option. Type the command name followed by one of these options to try it.
- The Debian Documentation Project has written some manuals, including this one. Check out their home page (`http://www.debian.org/~elphick/ddp/`).
- The Debian support page (`http://www.debian.org/support/`) has a FAQ and other resources. You can also try the Linux web site (`http://www.linux.org`).
- You can buy many proprietary books with helpful information. Most people praise the O'Reilly brand very highly. However, do consider supporting freely modifiable and redistributable manuals such as this one when possible. If you want hard copy, purchasing free manuals from the Free Software Foundation (available at many bookstores, such as Borders, and direct from the FSF) supports the creation of more free software.

The confusing variety of documentation sources exists for many reasons. For example, `info` is supposed to replace `man`, but `man` hasn't disappeared yet. However, it's nice to know that so much documentation exists!

So where to look for help? Here are some suggestions:

- Use the `man` pages and the `--help` or `-h` option to get a quick summary of a command's syntax and options. Also use `man` if a program doesn't yet have an `info` page.
- Use `info` if a program has `info` documentation.
- If neither of those work, look in `/usr/doc/`*`packagename`*.
- `/usr/doc/`*`packagename`* often has Debian-specific information, even if there's a man page or info page.
- Use the `HOWTOs` for instructions on how to set up a particular thing, or information on your particular hardware. For example, the Ethernet HOWTO has a wealth of information on ethernet cards, and the PPP HOWTO explains in detail how to set up PPP.
- Use the Debian Documentation Project manuals for conceptual explanations and Debian-specific information.
- If all else fails, ask someone. See 'Getting help from a person' on the next page.

Using `man` pages is discussed above in 'The command line and `man` pages' on page 13. (It's very simple: press the spacebar to go to the next page, and press `q` to quit reading.) `info`, viewing files in `/usr/doc`, and asking for help from a person are all discussed in this chapter.

5.2 Using info

A brief keystroke summary/tutorial, mention TkInfo, apologize for ridiculous keystrokes.

5.3 Viewing text files with more and less

Use these to view some docs. Mention zless and when to use it.

5.4 HOWTOs

In addition to their books, the Linux Documentation Project has made a series of short documents describing how to set up a particular aspect of GNU/Linux. For instance, the SCSI-HOWTO describes some of the complications of using SCSI — a standard way of talking to devices — with GNU/Linux. In general, the HOWTOs have more specific information about particular hardware configurations, and will be more up to date than this manual.

There are Debian packages for the HOWTOs. *doc-linux-text* contains the various HOWTOs in text form; while the *doc-linux-html* package contains the HOWTOs in (surprise!) browsable HTML format. Note also that Debian has packaged translations of the HOWTOs in various languages that you may prefer if English is not your native language. [1] If you've installed one of these, you should have them in `/usr/doc/HOWTO`. However, you may be able to find more recent versions on the net on the LDP Project home page (`http://sunsite.unc.edu/LDP/`).

5.5 Getting help from a person

The correct place to ask for help with Debian is the debian-user mailing list `<debian-user@lists.debian.org>`. If you know how to use IRC (Internet Relay Chat), there is a `#debian` channel on `irc.debian.org`. You can find general GNU/Linux help on the `comp.os.linux.*` Usenet hierarchy. You can search past Usenet questions and answers with the DejaNews service (`http://www.dejanews.com`). It is also possible to hire paid consultants to provide guaranteed support services. The Debian web site (`http://www.debian.org`) has more information on many of these resources.

Again, please *do not* ask the authors of this tutorial for help. We probably don't know the answer to your specific problem anyway; if you mail `debian-user`, you will get higher-quality responses, and more quickly.

Always be polite and make an effort to help yourself by reading the documentation. Remember, Debian is a volunteer effort and people are doing you a favor by giving their time to help you. Many of them charge hundreds of dollars for the same services during the day.

5.5.1 Dos and Don'ts of asking a question

- DO read the obvious documentation first. Things like command options and what a command does will be there.
- DO check the HOWTO documents if your question is about setting up something, such as PPP or Ethernet.
- DO try to be sure the answer isn't in this tutorial (though we realize an index would be helpful—we're working on it!).
- DON'T be afraid to ask, after you've made a basic effort to look it up.

[1]Debian have packages for the German, French, Spanish, Italian, Japanese, Korean, Polish, Swedish and Chinese versions of the HOWTOs; usually available in the package *doc-linux-languagecode*, with *languagecode* being `fr` for French, `es` for Spanish, etc. ..

- DON'T be afraid to ask for conceptual explanations, advices, and other things not often found in the documentation.
- DO include any information that seems relevant. You'll almost always want to mention the version of Debian you're using. You may also want to mention the version of any pertinent packages: the command `dpkg --status` *`packagename`* will tell you this. It's also useful to say what you've tried so far and what happened. Please include the exact error messages, if any.
- DON'T apologize for your ignorance, or make excuses for being a newbie. There's no reason everyone should be a GNU/Linux expert to use it, any more than everyone should be a mechanic to use a car.
- DON'T post or mail in HTML. Some versions of Netscape and Internet Explorer will post in HTML rather than plain text. Most people will not even read these posts, because they are difficult to read in most mail programs. There should be a setting somewhere in the preferences to disable HTML.
- DO be polite. Remember that Debian is an all-volunteer effort, and anyone who helps you is doing it just because they're a nice person.
- DO re-mail your question to the list if you've gotten no responses after several days. Perhaps there were lots of messages and it was overlooked. Or perhaps no one knows the answer — if no one answers the second time, this is a good bet. You might want to try including more information the second time.
- DO answer questions yourself, when you know the answer. Debian depends on everyone doing their part — if you ask a question, and later on someone else asks the same question, you'll know how to answer it. Do so!

5.6 Getting information from the system

When diagnosing problems or asking for help, you'll need to get information about your system. Here are some ways to do so.

/var/log/*, dmesg, uname -a

Chapter 6

Using the shell

6.1 Environment variables

Every process has an *environment* associated with it. An environment is a collection of *environment variables*. A variable is a changeable value with a fixed name. For example, the name `EMAIL` could refer to the value `joe@nowhere.com`. The value can vary — `EMAIL` could also refer to `jane@somewhere.com`.

Since your shell is a process like any other, it has an environment. You can view your shell's environment by entering the `printenv` command. Here's some example output:

```
PAGER=less
HOSTNAME=icon
MAILCHECK=60
MOZILLA_HOME=/usr/local/lib/netscape
PS1=$
USER=hp
MACHTYPE=i486-pc-linux-gnu
EDITOR=jed
DISPLAY=:0.0
LOGNAME=hp
EMAIL=hp@pobox.com
SHELL=/bin/bash
HOSTTYPE=i486
OSTYPE=linux-gnu
HISTSIZE=150
HOME=/home/hp
TERM=xterm-debian
TEXEDIT=jed
```

```
PATH=/home/hp/local/bin:/usr/sbin:/home/hp/.bin:/home/hp/local/bin:/usr/sbin:/us
_=/usr/bin/printenv
```

On your system, the output will be different, but similar.

Environment variables are one way to configure the system. For example, the `EDITOR` variable lets you select your preferred editor for posting news, writing email, and so on. The `HISTSIZE` variable tells Bash how many command lines to keep in its history; you can return to that many command lines with the up arrow key.

Setting environment variables is simple. Once you learn how, you'll probably want to set them automatically whenever you log on; see 'Customizing the shell' on page 57 for instructions.

For practice, try customizing your shell's prompt and your text file viewer with environment variables:

1. `man less`

   View the online manual for the `less` command. In order to show you the text one screenful at a time, `man` invokes a *pager* which shows you a new page of text each time you press the space bar. By default, it uses the pager called `more`.

   Go ahead and glance over the man page for `less`, which is an enhanced pager. Scroll to a new page by pressing space; press `q` to quit. `more` will also quit automatically when you reach the end of the man page.

2. `export PAGER=less`

   After reading about the advantages of `less`, you might want to use it to read man pages. To do this, you set the environment variable `PAGER`.

   The command to set an environment variable within bash always has this format: `export` *`NAME`*`=`*`value`*. If you happen to run `tcsh` or another C Shell derivative, the equivalent command is `setenv` *`NAME value`*.

   `export` means to move the variable from the shell into the environment. This means that programs other than the shell will be able to access it.

3. `echo` *`$PAGER`*

   This is the easiest way to see the value of a variable. `$PAGER` tells the shell to insert the value of the `PAGER` variable *before* invoking the command. `echo` echoes back its argument: in this case, the it echoes the current `PAGER` value, `less`.

4. `man more`

   Read the `more` manual. This time, `man` should have invoked the `less` pager.

   `less` has lots of features `more` lacks. For example, you can scroll backward with the `b` key. You can also move up and down (even sideways) with the arrow keys. `less` won't exit when it reaches the end of the man page; it will wait for you to press `q`.

5. `PAGER=more man more`

   If you want a different setting temporarily, you can put a new value in effect for the current command line only. Put the *`NAME=value`* at the start of the command line, followed by the command to execute. Be sure to omit `export`.

   You can try out some `less`-specific commands, like `b`, to verify that they don't work with `more` and you are indeed using `more`.

6. `echo $PAGER`

   The value of `PAGER` should still be `less`; the above setting was only temporary.

7. `unset PAGER`

   If you don't want to specify a pager anymore, you can `unset` the variable. `man` will then use `more` by default, just as it did before you set the variable.

8. `echo $PAGER`

   Since `PAGER` has been unset, `echo` won't print anything.

9. `PS1=hello:`

   Just for fun, change your shell prompt. `$` should become `hello:`.

   `export` is not necessary, because we're changing the shell's own behavior. There's no reason to export the variable into the environment for other programs to see. Technically, `PS1` is a *shell variable* rather than an environment variable.

   If you wanted to, you could `export` the shell variable, transforming it into an environment variable. Then other programs could see it: specifically, the *children* of the current shell process. The next section explains this.

6.1.1 Parent and child processes

All processes come from an earlier process, called their *parent process*.

1

The `ps` command is a useful tool for exploring processes, and it can be used to examine parent-child relationships.

1. `ps f`

   This command asks to see a list of processes belonging to you, in a format that shows how processes are related.

[1] You may see a chicken and egg problem here. There is an original process that starts all the others; it's process number 1, `init`. You can see it running by typing `ps u 1`.

`ps f` might produce output like this:

```
$ ps f
  PID  TT STAT   TIME
 7270  p5 S      0:00 bash
15980  p5 R      0:00  \_ ps f
19682  p4 S      0:00 bash
15973  p4 S      0:00  \_ man ps
15976  p4 S      0:00      \_ sh -c /bin/gzip -dc '/var/catman/cat1/ps.1.gz' | {
port MAN_PN LESS; MAN_PN='ps(1)'; LESS="$LESS\$-Pm\:\$i
15977  p4 S      0:00          \_ /bin/gzip -dc /var/catman/cat1/ps.1.gz
15978  p4 S      0:00          \_ sh -c /bin/gzip -dc '/var/catman/cat1/ps.1.gz'
port MAN_PN LESS; MAN_PN='ps(1)'; LESS="$LESS\$-Pm\
15979  p4 S      0:00              \_ less
$
```

Here you can see that I have a number of processes running, including two shells. The shells have child processes: shell process 7270 has child process 15980 (`ps f`) and shell 19682 has child process 15973 (`man ps`). `man ps` has in turn invoked a complex set of subprocesses in order to display a man page. Don't worry about what these subprocesses do for now.

Parents and children have a complex relationship. Most of the time, when a parent dies the child will die as well. So you can kill a whole set of processes — for example, all the `man ps` children in the above example — by killing the parent process, 15973.

Children inherit the environment variables of their parents, and some other attributes such as the current working directory.

When a shell runs a command, it spawns the command as a child process. So the `man` command inherits the shell's environment; if you've set the `PAGER` variable, `man` will be able to see it.

If you fail to `export` a variable, only the shell itself will see it, and it will not be passed on to children such as `man`.

6.2 Where commands live: the `PATH` variable

When you type a command into the shell, it has to find the program on your hard disk before executing it. If the shell had to look all over the disk, it would be very slow; instead, it looks in a list of directories contained in the `PATH` environment variable. This list of directories makes up the shells' *search path*; when you enter a command, it goes through each one in turn looking for the program you asked to run.

You may need to change the `PATH` variable if you install programs yourself in a nonstandard location.

The value of `PATH` is a colon-separated list of directories. The default value on Debian systems is:

```
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

This value is defined in the file `/etc/profile` and applies to all users. You can easily change the value, just as you can change any environment variable.

If you type the command `ls`, the shell will first look in `/usr/local/bin`; `ls` isn't there, so it will try `/usr/bin`; when that fails, it will check `/bin`. There it will discover `/bin/ls`, stop its search, and execute the program `/bin/ls`. If `/usr/bin/X11/ls` existed (it doesn't, but pretend), it would be ignored.

You can see which `ls` the shell is going to use with the `type` command. `type ls` will give you the answer `/bin/ls` — try it yourself.

Try asking where `type` itself resides:

```
$ type type
type is a shell builtin
```

`type` isn't actually a program; it's a function provided by the shell. However, you use it just like an external program. [2]

There are a number of commands like this; type `man builtins` to read the man page describing them. In general, you don't need to know whether a command is a builtin or a real program; however, builtins will not show up in the output of `ps` or `top` since they aren't separate processes. They're just part of the shell.

6.3 Aliases and shell functions

If you use the same command often, you might get tired of typing it. `bash` lets you write shorter *aliases* for your commands. You can also write *shell functions*, which are custom commands made up of several other commands.

Say you always use the `--almost-all` and `--color=auto` options to `ls`. You quickly get tired of typing `ls --almost-all --color=auto`. So you make an alias:

```
alias myls='ls --almost-all --color=auto'
```

Now you can type `myls` instead of the full command. To see what `myls` really is, run the command `type myls`. To see a list of aliases you've defined, type simply `alias` on a line by itself.

Shell functions are a little more flexible than aliases. An alias simply substitutes a longer command when you type a shorter one. Functions let you use a series of commands to perform some action.

First let's see how a shell function could be used in place of the above alias:

[2] If you're running a C Shell derivative, the equivalent builtin to `type` is `which`.

```
myls() {
    ls --almost-all --color=auto $*
}
```

The above is called a *function definition*, because it gives a function name (`myls`), then defines the meaning of the name (some commands to execute). To define a function, write its name, followed by `()`. Then include the commands to execute inside braces (`{}`). The portion inside braces is known as the `body` of the function.

The arguments to the function can be referred to as `$*`. So if you type:

```
myls /usr  /etc
```

`$*` will be `/usr /etc`, the two arguments. If there are no arguments, `$*` will be empty.

You can also refer to the arguments by number. So `$1` in the function body would be replaced by `/usr`, and `$2` would be replaced by `/etc`. Type in this function (you can type it at the shell prompt; hit return after each line):

```
print_arguments() {
    echo "First argument:   $1"
    echo "Second argument:  $2"
    echo "All arguments:    $*"
}
```

You can verify you entered the function definition correctly with the `type` command; `type print_arguments` should say:

```
print_arguments is a function
print_arguments ()
{
    echo "First argument:   $1";
    echo "Second argument:  $2";
    echo "All arguments:    $*"
}
```

Try the function out. If you enter `print_arguments one two` it will display:

```
First argument:   one
Second argument:  two
All arguments:    one two
```

There are many more complex things you can do in a shell function; you're limited only by your imagination. For more, see 'Introduction to shell scripting' on page 88.

6.4 Controlling input and output

Stdin, stdout, pipelines, and redirection

Every process has at least three connections to the outside world. The *standard input* is one source of the process's data; the *standard output* is one place the process sends data; and the *standard error* is a place the process can send error messages. (These are often abbreviated `stdin`, `stdout`, and `stderr`.)

The words 'source' and 'place' are intentionally vague. These standard input and ouput locations can be changed by the user; they could be the screen, the keyboard, a file, even a network connection. The user can specify which locations to use.

When you run a program from the shell, usually standard input comes from your keyboard and standard output and error both go to your screen. However, you can ask the shell to change these defaults.

For example, the `echo` command sends it output to standard output, normally the screen. But you can send it to a file instead, with the *output redirection operator*, '`>`'. For example, to put the word "Hello" in the file `myfile`:

```
echo Hello > myfile
```

Use `cat` or your text file pager (`more` or `less`) to view `myfile`'s contents.

You can change the standard input of a command with the *input redirection operator*, '`<`'. For example, `more < myfile` will display the contents of `myfile`. This is not useful in practice; for convenience, the `more` command accepts a filename argument. So you can simply say `more myfile` and the effect will be the same.

Under the hood, `more < myfile` means that the shell opens `myfile`, then feeds its contents to the standard input of `more`. `more myfile`, without the redirection operator, means that the `more` command receives one argument, `myfile`, opens the file itself, and then displays the file.

There's a reason for the double functionality, however. For example, you can connect the standard output of one command to the standard input of another. This is called a *pipeline*, and it uses the *pipe operator*, '`|`'.

Perhaps you want to see the GNU General Public License in reverse. To do this, you use the `tac` command (it's `cat`, only backward). Try it out:

```
tac /usr/doc/copyright/GPL
```

Unfortunately, it goes by too quickly to read. So you only get to see a couple of paragraphs. The solution is a pipeline:

```
tac /usr/doc/copyright/GPL | more
```

This takes the standard output of `tac`, which is the GPL in reverse, and sends it to the standard input of `more`.

You can chain as many commands together as you like. Say you have an inexplicable desire to replace every `G` with `Q`; for this you use the command `tr G Q`, like this:

```
tac /usr/doc/copyright/GPL | tr G Q | more
```

You could get the same effect using temporary files and redirection. For example:

```
tac /usr/doc/copyright/GPL > tmpfile
tr G Q < tmpfile > tmpfile2
more < tmpfile2
rm tmpfile tmpfile2
```

Clearly a pipeline is more convenient.

6.5 Specifying how and when to run commands

"modifiers" like batch, at, nohup, nice

6.6 Filename expansion ("Wildcards")

Often you want a command to work with a group of files. "Wildcards" are used to create a *filename expansion pattern*: a series of characters and wildcards that expands to a list of filenames. For example, the pattern `/etc/*` expands to a list of all the files in `/etc` [3]. `*` is a wildcard which can stand for any series of characters, so the pattern `/etc/*` will expand to a list of all the filenames beginning with `/etc/`.

This filename list is most useful as a set of arguments for a command. For example, the `/etc` directory contains a series of subdirectories called `rc0.d`, `rc1.d`, etc. Normally to view the contents of these, you would type:

```
ls /etc/rc0.d /etc/rc1.d /etc/rc2.d /etc/rc3.d /etc/rc4.d /etc/rc5.d /etc/rc6.d /
```

This is tedious. Instead, you can use the `?` wildcard:

```
ls /etc/rc?.d
```

[3]Actually, fi les beginning with `.` are not included in the expansion of `*`

`/etc/rc?.d` expands to a list of filenames which begin with `rc`, followed by any single character, followed by `.d`.

Available wildcards are:

`*` Matches any group of 0 or more characters

`?` Matches exactly one character

`[...]` If you enclose some characters in brackets, the result is a wildcard which matches those characters. For example, `[abc]` matches either a, or b, or c. If you add a ˆ after the first bracket, the sense is reversed; so `[ˆabc]` matches any character that is not a, b, or c. You can include a range, such as `[a-j]`, which matches anything between a and j. The match is case sensitive, so to allow any letter, you must use `[a-zA-Z]`.

Expansion patterns are simple, once you see some concrete examples:

`*.txt` This will give you a list of any filename which ends in `.txt`, since the `*` matches anything at all.

`*.[hc]` This gives a list of filenames which end in either `.h` or `.c`.

`a??` This gives you all three-letter filenames that begin with `a`.

`[ˆa]??` This gives you all three-letter filenames that do *not* begin with `a`.

`a*` This gives you every filename that starts with `a`, regardless of how many letters there are.

6.7 Interactive/non-interactive

Bash has two different modes: *interactive* and *non-interactive*. Interactive means you can type into it, and have it do things for you. Non-interactive shells interpret shell scripts, similar to DOS batch files. You give it a list of commands to carry out, and it goes and does them, but without your intervention. You don't see all the commands being typed in, Of course any output will be recorded somewhere (the standard output, or stdout, normally the screeen or a log file). We will get more into non-interactive shells a little later on.

6.7.1 Interactive shells

Interactive shells will take a very long time for one to master, just because they're so powerful — you'll probably never learn everything! There is just so much out there that a shell can do, and of course it's always changing. We will talk about `bash` here, and some basic commands that will make your life with a shell easier. In bash, one can have several different things going on all at once, and this can get confusing.

A shell is a Line Oriented or command line environment. The shell will always prompt you with a prompt, whenever it is waiting on you to do things. The default debian prompt is a $. At the $ prompt is where you can type in commands to tell linux to do things, it can be a program name, or it can be a "builtin" command that the shell provides for your convenience.

Chapter 7

More on files

In ‘Files and Directories’ on page 15 we covered moving/renaming files with `mv`, copying them with `cp`, removing them with `rm`, removing directories with `rmdir`, and creating directories with `mkdir`. This chapter will cover some more aspects of files.

7.1 Permissions

GNU and Unix systems are set up to allow many people to use the same computer, while keeping certain files private or keeping certain people from modifying certain files. You can verify this for yourself:

1. Log in as yourself, i.e. *NOT* as root.
2. `whoami`

   Verifies that you are not root.
3. `rm /etc/resolv.conf`

   You should be told "Permission denied." `/etc/resolv.conf` is an essential system configuration file — you aren’t allowed to change or remove it unless you’re root. This keeps you from accidentally messing up the system, and if the computer is a public one such as at an office or school, it keeps users from messing up the system on purpose.

Now type `ls -l /etc/resolv.conf`

This will give you output that looks something like this:

```
-rw-r--r-- 1 root root 119 Feb 23 1997 /etc/resolv.conf
```

The `-l` option to `ls` requests all that additional information. The info on the right is easy - the size of the file is `119` bytes, the date the file was last changed is `Feb 23 1997`, the file's name is `/etc/resolv.conf`. On the left side of the screen, things get a little more complicated.

First, the brief, technical explanation: the `-rw-r--r--` is the *mode* of the file, the `1` is the number of hard links to this file (or the number of files in a directory), and the two `root` are the user and group owning the file.

So that was cryptic. Let's go through it slowly (except the hard links part — for that see 'The real nature of files: hard links and inodes' on page 88).

7.1.1 File Ownership

Every file has two owners — a user, and a group. The above case is a little confusing, since there's a group called `root` in addition to the `root` user. Groups are just collections of users who are collectively permitted access to some part of the system. A good example is a `games` group. Just to be mean, you might set up your system so that only people in a `games` group are allowed to play games.

A more practical example: say you're setting up a computer for a school. You might want certain files to be accessible only to teachers, not students, so you put all the teachers in a single group. Then you can tell the system that certain files belong to members of the group `teachers`, and that no one else can access those files.

Here are some things you can do to explore groups on your system:

1. `groups`

   Typing this at the shell prompt will tell you what groups you're a member of. It's likely that you're a member of only one group, which is identical to your username.

2. `more /etc/group`

   This file lists the groups that exist on your system. Notice the `root` group (the only member of this group is the root user), and the group which corresponds to your username. There are also groups like `dialout` (users who are allowed to dial out on the modem), and `floppy` (users who can use the floppy drive). However, your system is probably not configured to make use of these groups — it's likely that only root can use the floppy or the modem right now. For details about this file, try reading `man group`.

3. `ls -l /home`

   Observe how every user's directory is owned by that user and that user's personal group. (If you just installed Debian, you may be the only user.)

7.1.2 Mode

In addition to being owned by one user and one group, every file and directory also has a mode, which determines who's allowed to read, write, and execute the file. There are a few other things also determined by the mode, but they're advanced topics so we'll skip them for now.

The mode looks like this in the `ls` output: `-rw-r--r--`. There are ten "elements" here, and the mode actually consists of twelve bits (think of bits as switches which can be on or off). But for now, we'll consider only nine of these bits: those that control *read*, *write*, and *execute* permissions for the *user* owning the file, the *group* owning the file, and *others* (everyone on the system, sometimes called *world*).

Notice that three kinds of permission (read, write, execute) times three sets of people who can have permission (user, group, others) makes a total of nine elements.

In the mode line, the first "element" gives the type of the file. The `-` in this case means it's a regular file. If it was `d`, we'd be looking at a directory. There are other possibilities too complex to go into now (see 'Advanced aspects of file permissions' on page 94).

The remaining nine "elements" are used to display the 12 bits that make up the file's mode. The basic 9 bits (read, write, and execute for user, group, and other) are displayed as three blocks of `rwx`.

So if all permissions are turned on and this is a regular file, the mode will look like this: `-rwxrwxrwx`. If it was a directory with all permissions turned off for others and full permissions for user and group, it would be `drwxrwx---`.

(The remaining three bits are displayed by changing the `x` to `s`, `t`, `S`, or `T`, but this is a complex topic we're saving for 'Advanced aspects of file permissions' on page 94.)

For regular files, "read", "write", and "execute" have the following meanings:

- Read permission, indicated by `r`, gives permission to examine the contents of a file. For directories, it gives permission to list the contents of the directory.
- Write permission, indicated by `w`, gives permission to make changes to a file. For directories, it gives permission to create and remove files in the directory.
- Execute permission, indicated by `x`, gives permission to run the file as a command. Clearly it only makes sense to set execute permission if the file actually is a command.

  Since directories can never be executed, the execute bit has a different meaning. For directories, execute permission means permission to access files in the directory. Note that this interacts with write permissions: execute permissions must be set to be able to access files in a directory *at all*, so without execute permission on a directory, write permission is useless. Execute permission for directories is often called "search" permission, since it really has nothing to do with execution. "File access" permission would probably be a still better name.

Directory modes are a little confusing, so here are some examples of the effects of various combinations:

- `r--`

  The user, group, or other with these permissions may list the contents of the directory, but nothing else. The files in the directory can't be read, changed, deleted, or manipulated in any way. The only permitted action is reading the directory itself, that is, seeing what files it contains.

- `rw-`

  Write permission has no effect in the absence of execute permission, so this mode behaves just like the above mode.

- `r-x`

  This mode permits the files in a directory to be listed, and permits access to those files. However, files can't be created or deleted. *Access* means that you can view, change, or execute the files as permitted by the files' own permissions.

- `--x`

  Files in this directory can be accessed, but the contents of the directory can't be listed, so you have to know what filename you're looking for in advance (unless you're a good guesser). Files can't be created or deleted.

- `rwx`

  You can do anything you want with the files in this directory, as long as it's permitted by the permissions on the files themselves.

Directory write permission determines whether you can delete files in a directory — a read-only file can be deleted, if you have permission to write to the directory containing it. You can't delete a file from a read-only directory, even if you're allowed to make changes to the file. File permissions have nothing to do with deleting files.

This also means that if you own a directory you can always delete files from it, even if those files belong to root.

Directory execute permission determines whether you have access to files — and thus whether file permissions come into play. *If* you have execute permissions to a directory, file permissions for that directory become relevant. Otherwise file permissions just don't matter; you can't access the files anyway.

If you have execute permission for the directory, file permissions determine whether you can read the contents of the file, change the file, and/or execute the file as a command.

Finally, permission to change permissions on a file or directory is not affected by the permissions of that file or directory. Rather, you can always change the permissions on files or directories that you own, but not on files owned by someone else, as long as you are permitted access to the file. So if you can access a file you own at all (that is, you have execute permission for the directory containing it) then you can change its permissions.

This means that you can't permanently remove permissions from yourself because you can always give them back. Say you remove user write permission from a file you own, then try to change the file. It won't be permitted, but you can always give yourself write permission again and *then* change the file. The only way to lose the ability to change permissions back is to lose access to the file entirely.

7.1.3 Permissions in practice

This section goes through a short example session to demonstrate how permissions are used.

To change permissions, we'll use the `chmod` command.

1. `cd; touch myfile`

   There are a couple of new tricks here. First, you can use `;` to put two commands on one line. You can type the above as:

   ```
   $ cd
   $ touch myfile
   ```

   or as:

   ```
   $ cd; touch myfile
   ```

   and the same thing will end up happening.

   Recall that `cd` by itself returns you to your home directory. `touch` is normally used to change the modification time of the file to the current time, but it has another interesting feature: if the file doesn't exist, `touch` creates the file. So we're using it to create a file to practice with. Use `ls -l` to confirm that the file has been created, and notice the permissions mode:

   ```
   $ ls -l
   -rw-r--r-- 1 havoc havoc 0 Nov 18 22:04 myfile
   ```

   Obviously the time and user/group names will be different when you try it. The size of the file is 0, since `touch` creates an empty file. `-rw-r--r--` is the default permissions mode on Debian .

2. `chmod u+x myfile`

   This command means to add (`+`) execute (`x`) permissions for the user (`u`) who owns the file. Use `ls -l` to see the effects.

3. `chmod go-r myfile`

   Here we've subtracted (`-`) read permission (`r`) from the group (`g`) owning the file, and from everyone else (others, `o`). Again, use `ls -l` to verify the effects.

4. `chmod ugo=rx myfile`

   Here we've set (`=`) user, group, and other permissions to read and execute. This sets permissions to *exactly* what you've specified, and unsets any other permissions. So all `rx` should be set, and all `w` should be unset. Now, no one can write to the file.

5. `chmod a-x myfile`

   `a` is a shortcut for `ugo`, or "all". So all the `x` permissions should now be unset.

6. `rm myfile`

   We're removing the file, but without write permissions. `rm` will ask if you're sure:

   ```
   rm: remove 'myfile', overriding mode 0444?
   ```

   You should respond by typing `y` and pressing enter. This is a feature of `rm`, not a fact of permissions - permission to delete a file comes from the directory permissions, and you have write permission in the directory. However, `rm` tries to be helpful, figuring that if you didn't want to change the file (and thus removed write permission), you don't want to delete it either, so it asks you.

What was that `0444` business in the question from `rm`? The permissions mode is a twelve-digit binary number, like this: `000100100100`. `0444` is this binary number represented as an octal (base 8) number, which is the conventional way to write a mode. So you can type `chmod 444 myfile` instead of `chmod ugo=r myfile`. This is fully explained in 'Advanced aspects of file permissions' on page 94.

7.2 What files are on my system? Where can I put my own files?

Now that you can navigate the directory tree, let's take a guided tour of the files and directories you created when you installed Debian. If you're curious, `cd` to each directory and type `ls` to see its contents. If the listing doesn't fit on the screen, try `ls | more`, where `|` is the "pipe" character, generally found on the same key with backslash.

/ As already mentioned, this is the root directory, which contains every other directory.

/root But don't get `/` confused with `/root`! `/root` is the home directory of the root user, or superuser. It's a directory called `/root`, but it isn't *the* root directory `/`.

/home This is where all normal users — that is, all users except root — have their home directories. Home directories are named after the user who owns them, for example, `/home/jane`. If you're using a large system at a school or business, your system administrator may create additional directories to contain home directories: `/home1` and `/home2` for example. On some other systems, you'll see an additional level of subdirectories: `/home/students/`*`username`*, `/home/staff/`*`username`*, etc. ..

Your home directory is where you put all your personal work, email and other documents, and personal configuration preferences. It's your home on the system.

`/bin` This directory contains "binaries," executable files which are essential to the operation of the system. Examples are the shell (`bash`), and file commands such as `cp`.

`/sbin` This directory contains "system binaries", utilities that the root user or system administrator might want to use, but probably you won't want to use in your day-to-day activities.

`/usr` `/usr` contains most of the files you'll be interested in. It has many subdirectories: `/usr/bin` and `/usr/sbin` are pretty much like `/bin` and `/sbin`, except that the directories in `/usr` are not considered "essential to the operation of the system".

While not essential to get the computer working,`/usr` does contain the applications you'll use to get real work done. Also in `/usr` you'll find the `/usr/man`, `/usr/info`, and `/usr/doc` directories — these contain manual pages, info pages, and other documentation, respectively. And don't forget `/usr/games`!

`/usr/local` The Debian system doesn't install anything in this directory. You should use it if you want to install software that you compile yourself, or any software not contained in a Debian package. You can also install software in your home directory, if you'll be the only one using it.

`/etc` `/etc` contains all the system-wide configuration files. Whenever you want to change something that affects all users of your computer — such as how you connect to the internet, or what kind of video card you have — you'll probably have to log on as root and change a file in `/etc`.

`/tmp` Here you'll find temporary files, most of them created by the system. This directory is generally erased on a regular basis, or every time you reboot the system. You can create files here if you want, just be aware they might get deleted automatically.

`/var` `/var` contains "variable" files, that the system changes automatically. For example, incoming mail is stored here. The system keeps a log of its actions here. There are a number of other automatically generated files here as well. You'll mostly be interested in the contents of `/var/log`, where you can find error messages and try to figure out what you're system's up to if something goes wrong.

Clearly there are many more directories on the system, too many to describe every one.

For changing things, you'll usually want to confine yourself to your home directory and `/etc`. On a Debian system, there's rarely an occasion to change anything else, because everything else is automatically installed for you.

`/etc` is used to configure the *system* as a whole. You'll use your own home directory, a subdirectory of `/home`, for configuring your own preferences, and storing your personal data. The idea is that on a day-to-day basis you confine yourself to `/home/yourname`, so there's no way you can break anything. Occasionally you log in as root to change something in a system-wide directory, but only when absolutely necessary. Of course, if you're using Debian at a school or business and someone else is the system administrator, you won't have root access and will only be able to change your home directory. This limits what you can do with the system.

7.3 Using a filemanager

Instead of moving files around by hand, you can use a *file manager*. If you move a lot of files around a file manager can make your work more efficient. There are text-based file managers, such as GNU Midnight Commander (type `mc`), and a number of file managers for the X Window System (for example `gmc` for the X Window version of GNU Midnight Commander).

Describing each of these is outside the scope of this manual; but you may want to try them out if the command line doesn't meet your needs.

Chapter 8

Creating and editing text files

8.1 What's a text file?

A *text file* is simply a normal file that happens to contain human-readable text. There's nothing special about it otherwise. The other kind of file, a binary file, is meant to be interpreted by the computer.

You can view either kind of file with the `less` file pager, if you have it installed (install it if you haven't, it's quite useful). Type `less /etc/profile` to view a sample text file — notice that you can read the characters, even if their meaning is obscure. Type `less /bin/ls` to view a binary file; as you can see, the `ls` program is not meant to be read by humans.

The difference between the two kinds of files is purely a matter of what they contain, unlike some other systems (such as DOS or MacOS) which actually treat the files differently.

Text files can contain shell scripts, documentation, copyright notices, or any other human-readable text.

Incidentally, this illustrates the difference between *source code* and *binary executables*. `/bin/ls` is a binary executable you can download from Debian, but you can also download a text file which tells the computer how to create `/bin/ls`. This text file is the source code. Comparing `/bin/ls` to `/etc/profile` illustrates how important source code is if someone wants to understand and modify a piece of software. Free software provides you or your consultants with this all-important source code.

8.2 Text editors

A *text editor* is a program used to create and change the contents of text files. Most operating systems have a text editor; DOS has `edit`, Windows has `Notepad`, MacOS has `SimpleText`.

Debian provides a bewildering variety of text editors. `vi` and `emacs` are the classic two, probably both the most powerful and the most widely used. Both `vi` and `emacs` are quite complex and require some practice, but they can make editing text extremely efficient. `emacs` runs both in a terminal and under the X Window System; `vi` normally runs in a terminal but the `vim` variant has a `-g` option which allows it to work with X.

Simpler editors include `nedit`, `ae`, `jed`, and `xcoral`. `nedit` and `xcoral` provide easy-to-use X Window System graphical interfaces. There are also several `vi` variants, and an Emacs variant called `XEmacs`.

This tutorial will not cover the use of any particular editor in detail, though we will briefly introduce `vi` since it is small, fast, nearly always available, and you may need to use it sometime regardless of your preferred editor. Emacs provides an excellent interactive tutorial of its own; to read it, load Emacs with the `emacs` command and type `F1 t`. Emacs is an excellent choice for new users interested in a general-purpose or programming editor.

8.3 Creating and editing a text file with `vi`

`vi` (pronounced "vee eye") is really the only editor that comes with almost every Unix-like operating system, and Debian is no exception. `vi` was originally written at the University of California at Berkeley. The editor's name is short for "visual", referring to the fact that `vi` provides a visual display of the text file; this was once considered a unique feature, giving you an idea how old the program is.

`vi` is somewhat hard to get used to, but has many powerful features. In general, we suggest that a new user use Emacs for daily tasks such as programming. However, `vi` is sometimes more convenient or the only available editor; it is also a much smaller file to download.

The following discussion of `vi` should also apply to `vi` variants such as `elvis` and `vim`.

8.3.1 Creating a file

1. `vi testfile`

   In your home directory, invoke vi by typing `vi` followed by the name of the file you wish to create. You will see a screen with a column of tildes (`~`) along the left side. `vi` is now in command mode. Anything you type will be understood as a command, not as content to add to the file. In order to input text, you must type a command.

2. `i`

   The two basic input commands are `i`, which means "insert the text I'm about to type to the left of the cursor", and `a`, which means "append the text I'm about to type to the right of the cursor". Since you are at the beginning of an empty file, either of these would work. We picked `i` arbitrarily.

3. Type in some text; here's a profound statement from philosopher Charles Sanders Peirce, if you can't think of your own:

```
And what, then, is belief? It is the demi-cadence
which closes a musical phrase in the symphony of our
intellectual life.  We have seen that it has just
three properties: First, it is something that we are
aware of; second, it appeases the irritation of doubt;
and, third, it involves the establishment in our
nature of a rule of action, or, say for short, a
habit.
```

Press `RET` after each line, since `vi` will not move to the next line automatically; when you finish typing, press the `ESC` key to leave insert or append mode and return to command mode.

4. `:wq`

   If you've done everything correctly, when you type this command it should appear at the bottom of your screen, below all the ˜ characters. The `:` tells `vi` you're about to give a series of commands; the `w` means to write the file you've just typed in — in most new programs this is called "save" — and the `q` means to quit `vi`. So you should be back at the shell prompt.

5. `cat testfile`

   `cat` will display the file you typed on the screen.

Don't remove `testfile`, we'll use it in the next tutorial section.

As you use `vi`, always remember that pressing `ESC` will return you to command mode. So if you get confused, press `ESC` a couple times and start over.

`vi` has an annoying tendency to beep whenever you do something you aren't supposed to, like type an unknown command; don't be alarmed by this.

8.3.2 Editing an existing file

To use `vi`, you only need to read 'Moving around in a file' on this page and 'Deleting text' on the next page. Later sections explain advanced features, but they are not strictly necessary, though often more efficient and less tedious.

Moving around in a file

To move around in a file, Debian's `vi` allows you to use the arrow keys. The traditional keys also work, however; they are `h` for left, `j` for down, `k` for up, and `l` for right. These keys were chosen because they are adjacent on on the home row of the keyboard, and thus easy to type. Many people use them instead of the arrow keys since they're faster to reach with your fingers.

1. `vi testfile`

   Open the file you created earlier with `vi`. You should see the text you typed before.

2. Move around the file with the arrow keys or the `hjkl` keys. If you try to move to far in any direction, `vi` will beep and refuse to do so; if you want to put text there, you have to use an insertion command like `i` or `a`.

3. `:q`

   Exit `vi`.

Deleting text

1. `vi testfile`

   Open your practice file again.

2. `dd`

   The `dd` command deletes a line; the top line of the file should be gone now.

3. `x`

   `x` deletes a single character; the first letter of the second line will be erased. Delete and backspace don't work in `vi`, for historical reasons[1]. Some `vi` variants, such as `vim` will let you use backspace and delete.

4. `10x`

   If you type a number before a command, it will repeat the command that many times. So this will delete 10 characters.

5. `2dd`

   You can use a number with the `dd` command as well, deleting two lines.

6. `:q`

   This will cause an error, because you've changed the file but haven't saved yet. There are two ways to avoid this; you can `:wq`, thus writing the file as you quit, or you can quit without saving:

7. `:q!`

   With an exclamation point, you tell `vi` that you really mean it, and it should quit even though the file isn't saved. If you use `:q!` your deletions will not be saved to `testfile`; if you use `:wq`, they will be.

[1]The keyboard of some very old terminals (from the 60s) had no BackSpace or Delete key

8. `cat testfile`

   Back at the shell prompt, view `testfile`. It should be shorter now, if you used `:wq`, or be unchanged if you used `:q!`.

`:q!` is an excellent command to remember, because you can use it to bail out if you get hopelessly confused and feel you've ruined the file you were editing. Just press `ESC` a few times to be sure you're in command mode and then type `:q!`. This is guaranteed to get you out of `vi` with no damage done.

You now know everything you need to do basic editing; insertion, deletion, saving, and quitting. The following sections describe useful commands for doing things faster; you can skip over them if you like.

Sophisticated movement

There are many motion commands, here's a quick summary:

`w`	Move to the start of the next word
`e`	Move to the end of the next word
`E`	Move to the end of the next word before a space
`b`	Move to the start of the previous word
`0` (zero)	Move to the start of the line
`ˆ`	Move to the first word of the current line
`$`	Move to the end of the line
`RET`	Move to the start of the next line
`-`	Move to the start of the previous line
`G`	Move to the end of the file
`1G`	Move to the start of the file
***`n`*`G`**	Move to line number *n*
`C-G`	Display the current line number
`H`	Top line of the screen
`M`	Middle line of the screen
`L`	Bottom of the screen

***n*|**	Move cursor to column *n*

The screen will automatically scroll when the cursor reaches either the top or the bottom of the screen. There are alternative commands which can control scrolling the text.

C-f	Scroll forward a screen
C-b	Scroll backward a screen
C-d	Scroll down half a screen
C-u	Scroll down half a screen

Repeating commands

As mentioned above you can often prefix a command with a number to repeat that command multiple times. For example, the `l` key moves left; `10l` moves you left 10 positions to the left.

If you wanted to enter a number of spaces in front of the some text you could use a number with the insert command. Enter the number *n* then `i` followed by `SPACE` and `ESC`. You should get *n* spaces.

The commands that deal with lines use a number to refer to line numbers. The `G` is a good example; if you preface it with a number it will go to that line.

Advanced reference

This section gives a more comprehensive list of commands you can use. It is just a reference; if you want, try the commands out to see what they do.

Insertion commands:

a	Append to the right of the cursor
A	Append at the end of the line
i	Insert text to the left of the cursor
I	Insert text to the left of the first non-blank character on current line
o	Open a new line below the current line and insert text
O	Open a new line above the current line and insert text

Deletion commands:

x	Delete the character under the cursor
dw	Delete from the current position to the end of the word
dd	Delete the current line.
D	Delete from the current position to the end of the line

Commands in combination can be more powerful. In particular, `d` followed by a motion command deletes from the cursor to wherever you asked to move. Some examples:

d*n*w	Deletes *n* words (*n*`dw` works too)
dG	Delete from the current position to the end of the file
d1G	Delete from the current postion to the start of the file
d$	Delete from current postion to the end of the line (same as `D`)
d*n*$	Delete from current line the end of the *n*th line

Undo commands:

u	Undo the last command
U	Undo all change to the current line
:e!	"Edit again". Like quitting with `:q!` and restarting — returns you to the last time you did a `:w` to save.

You can undo an undo, so `uu` results in an undone undo, or no change.

Replacement commands:

r*c*	Replace the character under the cursor with *c*
R	Overwrites text
cw	Changes the current word
c$	Changes text from current position to end of the line
c*n*w	Changes next *n* words.(same as *n*cw)
c*n*$	Changes to the end of the *n*th line

C	Changes to the end of the line (same as *c$*)
cc	Changes the current line
s	Substitutes text you type for the current character
***n*s**	Substitutes text you type for the next *n* characters

The commands in the above list which allow you to enter more than a single character of text have to be exited with the `ESC` key, returning you to command mode.

Cut and paste involves first *yanking* (cutting or copying) some text and placing it in a buffer (or "clipboard"); then moving to the desired new location; then pasting the text.

To cut text use the `y` command and its variants:

yy	Yank a copy of the current line
***n*yy**	Yank the next *n* lines
yw	Yank a word
y*n*w	Yank *n* words
y$	Yank the text between the cursor and the end of the line

Paste commands:

p	Paste to the right of the cursor
P	Paste to the left of the cursor
***n*P**	Paste *n* copies to the left of the cursor

When using `vi` within an xterm or using a variant of `vi` that supports X, you can also use the mouse to copy text. See ‘The X Window System’ on page 59 for how to copy and paste in X; be sure you’re in insert mode when you paste, or the pasted text will be interpreted as a command.

When you delete, the deleted text is copied to the buffer (clipboard); you can then use the paste commands. This allows you to cut-and-paste, while the `y` commands result in copy-and-paste.

`vi` has commands to search for text. You can also use these as movement commands, if you want to move to a particular word or character.

The simplest search commands look for characters.

f*c* Find the next character *c* to the right of or below the current position

F*c* Find the next character *c* to the left of or above the current position

t*c* Move right to character before the next *c*.

T*c* Move left to the character following the preceding *c*.

`;` Repeats the last character search command

`,` Same as `;` but reverses the direction of the original command.

If the character you were searching for was not found, `vi` will beep or give some other sort of signal.

`vi` allows you to search for any text, not just a character.

/*text* Searches right and down for the next occurence of *text*.

?*text* Searches left and up for the next occurance of *text*.

n Repeat the last`/` or `?` command

N Repeats the last `/` or `?` in the reverse direction

When using the `/` or `?` commands a line will be cleared along the bottom of the screen. You enter the text to search for followed by `RET`.

The text in the command `/` or `?` is actually a *regular expression*, see 'Regular expressions' on page 65.

Chapter 9

Customizing the shell

9.1 .rc files and `ls -a`

When you type `ls`, files beginning with a dot are not listed. Traditionally, files which contain configuration information, user preferences, and so on begin with a dot; these are hidden and out of your way while you do you day-to-day work. Sample dotfiles are `~/.emacs`, `~/.newsrc`, `~/.bashrc`, `~/.xsession`, `~/.fvwmrc`, etc. .. These are used by Emacs, newsreaders, the Bash shell, the X Window System, and the `fvwm` window manager, respectively. It is conventional to end the dotfile name with `rc`, but some programs don't. There are also directories beginning with a dot, such as `~/.gimp` and `~/.netscape`, which store preferences for the Gimp and Netscape.

Sometimes a program will create a dotfile automatically; for example, Netscape allows you to edit your preferences with a graphical dialog and then it saves your choices. Other times you will create them yourself using a text editor; this is the traditional way to do it, but you have to learn the peculiar format of each file — inconvenient at first, but it can give you a lot of power.

To see dotfiles, you must use the `-a` option to `ls`. The long form of `-a` is `--all`, if you find that easier to remember. You can also use `-A` or `--almost-all`, which includes all dotfiles except `.` and `..` — remember that `.` is the current directory, and `..` is the parent of the current directory; since these are guaranteed to be in every directory, there is no real reason to list them with `ls`. You already know they are there.

9.2 System-wide vs. user-specific configuration

It's important to remember that there are two different kinds of configuration on a Debian system. *System-wide configuration* affects all users. System-wide settings are made in the `/etc` directory. You might configure the way the system connects to the internet, for example; or have web browsers on the system always start on the company home page. Since you want these settings to apply to all users, you make the changes in `/etc`. Sample configuration files in `/etc` include `/etc/X11/XF86Config`, `/etc/lynx.cfg`, and `/etc/ppp/options`. In fact nearly all the files in `/etc` are configuration files.

Normally you must be root to change system-wide settings.

User configuration affects only a single user. Dotfiles are used for user configuration. For example, the file `~/.newsrc` stores a list of which Usenet (discussion group) articles you have read, and which groups you are subscribed to. This allows newsreaders such as `trn` or Netscape to display unread articles in the groups you're interested in. This information will be different for every user on the system, so each user has their own `.newsrc` file in their home directory.

Chapter 10

The X Window System

This chapter describes the X Window System graphical user interface. It assumes that you have already successfully configured X as described in the Installation Manual (again, the install manual is not yet written; for now you will need to use the XFree86 HOWTO, the contents of `/usr/doc/X11`, and this chapter). Once you install X, you can enter the X environment by typing `startx` or via `xdm`, depending on your choice during configuration.

10.1 Starting the X environment

There are two ways of starting X. The first is to start X manually when you feel like using it. To do so, log in to one of the text consoles, and type `startx`. This will start X and switch you to its virtual console.

The second (and recommended) way to use X is with `xdm`, or X Display Manager. Basically `xdm` gives you a nice graphical login prompt on the X virtual console (probably VC 7), and you log in there.

By default, either method will also start an `xterm`, which is a small window containing a shell prompt. At the shell prompt, you can type any commands just as you would on a text VC. So you can follow all the examples in this tutorial using `xterm`; the only difference between an `xterm` and the text console is that you don't have to log on to the `xterm`, since you already logged on to X.

There are also a lot of things you can do only in X, which are covered in this chapter.

One note: the default `xterm` has a smallish font. If you have a small monitor or very high resolution or bad eyesight, you may want to fix this. Follow these steps:

1. Move the mouse pointer into the center of the `xterm` window.
2. Hold down the `Control` key and the *right* mouse button simultaneously. This will give you a font menu.
3. Point to the font you want and release the mouse button.

10.2 Intro: What is X

A GUI (Graphical User Interface) is part and parcel of the Windows or Mac operating systems. It's basically impossible to write an application for those systems which does not use the GUI, and the systems can't be used effectively from the command line. GNU/Linux is more *modular*, that is, made up of many small, independent components which can be used or not according to one's needs and preferences. One of these components is the X Window System, or simply X[1].

X itself is a means for programs to talk to your mouse and video card, without knowing what kind of mouse and video card you have. That is, it's an *abstraction* of the graphics hardware. User applications talk to X, in X's language; X then translates into the language of your particular hardware. This means that programs only have to be written once, and they work on everyone's computer.

In X jargon, the program which speaks to the hardware is known as an *X server*. User applications that ask the X server to show windows or graphics on the screen are called *X clients*. The X server includes a *video driver*, so you must have an X server which matches your video card.

The X server doesn't provide any of the features one might expect from a GUI, such as resizing and rearranging windows. A special X client, called a *window manager*, draws borders and titlebars for windows, resizes and arranges windows, and provides facilities for starting other X clients from a menu. Specific window managers may have additional features.

Window managers available on a Debian system include `fvwm`, `fvwm2`, `icewm`, `afterstep`, `olvwm`, `wmaker`, `mwm`, `twm`, `enlightenment` and others. You'll probably want to try them all and pick your favorite. (For information on window managers, and other fun stuff about X, X11.org (`http://www.x11.org`) is a nice site.)

Neither the X server nor the window manager provide a *file manager*; that is, there aren't any windows containing icons for your files and directories. You can launch a file manager as a separate application; there are many of them available, though unfortunately there aren't yet any good icon-based ones. The GNOME desktop project is developing an icon-based file manager and other GUI facilities, however. See the GNOME project home page (`http://www.gnome.org`) for the latest news on this.

A final feature of X is its *network transparency*, meaning that X clients don't care if they're talking to an X server on the same machine or an X server somewhere on the network. In practical terms, this means you can run a program on a more powerful remote machine, but display it on your desktop computer.

[1]It's also sometimes called X11 or X Window. Please note that "X Windows" is *not* correct and you're likely to offend the purists if you use this incorrect term.

10.3 Basic X operations

10.3.1 The mouse

The mouse in X works pretty much the same as the mouse on other systems, except that it has three buttons. If your mouse only has two, you can simulate the third (middle) button by clicking both buttons simultaneously. This is kind of tricky and annoying, so investing in a $15 3-button mouse probably isn't a bad idea. These are available from just any computer retailer.

The buttons are numbered from left to right, assuming you have a right-handed mouse, so button one is on the left, two in the middle, three on the right. You may see either the numbers or the locations in documentation.

X has a simple built-in copy-and-paste facility. To select text to copy, you click and drag with the left mouse button. This should select the text to copy, assuming the application you're using has copy-and-paste support. To paste the text, you click the middle mouse button in a different X application. For example, if you receive an email containing an URL, you can select the URL with the left button, then click in your web browser's "Location" field with the middle button to paste it in.

10.3.2 X clients

Programs that communicate with the X server are called X clients. Most of these programs will ask the X server to display windows on the screen.

You start an X client the same way you start any other Debian program. Simply type the name of the client on the command line. Try typing `xterm` into an existing xterm, and a new xterm client will appear on the screen.

You may notice that the original xterm is now useless, since your shell is waiting for the second xterm to finish. To avoid this problem, you can run the X client in the background — add a `&` after the command name, like this: `xterm &`. If you forget, you can place a running process in the background. First suspend the process with `C-z`, and then place it in the background with the `bg` command.

If you use a program often, your window manager will generally provide a way to put that program on a convenient graphical menu.

10.3.3 Troubleshooting

Sometimes when you launch an X client from a graphical menu, you won't be able to see any error messages if it fails. You can find any error messages in the file `~/.xsession-errors`.

10.3.4 Leaving the X environment

To leave X, you will need to use a menu. Unfortunately for beginners, this is different for every window manager, and for most window managers can be configured in many ways. If there's an obvious menu, look for an entry like "Exit" or "Close Window Manager". If you don't see a menu, try clicking each of the mouse buttons on the background of the screen. If all else fails, you can forcibly kill the X server by pressing `C-A-Backspace`. Forcibly killing the server will destroy any unsaved data in open applications.

10.4 Customizing your X startup

When you start X, Debian will run some shell scripts which start your window manager and other X clients. By default, a window manager, an `xconsole` and an `xterm` will be started for you.

To customize your X startup, the file `/etc/X11/config` must contain the line `allow-user-xsession`. If it does not, become root and add the line now. Then log back in as yourself and continue the tutorial. [2]

To run the clients of your choice when X starts, you create an executable shell script called `.xsession` in your home directory.

1. `touch ~/.xsession`

   This creates the file.

2. `chmod u+x ~/.xsession`

   Make the file executable.

Once `.xsession` is created, you need to edit it to do something useful with your favorite text editor. You can do anything you want to in this script. However, when the script's process terminates, X will also terminate.

In practical terms this means that you often end the script with a call to `exec`. Whatever program you `exec` will replace the script process with itself, so commands found after the `exec` line will be ignored. The program you exec will become the new "owner" of the script process, which means that X will terminate when this new program's process terminates.

Say you end your .xsession with the line: `exec fvwm`. This means that the `fvwm` window manager will be run when X starts. When you quit the fvwm window manager, your X session will end and all other clients will be shut down. You do not have to use a window manager here; you could `exec xterm`, in which case typing `exit` in that particular xterm would cause the entire X session to end.

[2]You can see how Debian's X startup works in the file `/etc/X11/Xsession`. Note that the behavior of `/etc/X11/Xsession` can be changed by modifying the file `/etc/X11/config`, which specifies a few system-wide preferences.

If you want to run other clients before doing your `exec`, you will need to run them in the background. Otherwise `.xsession` will pause until each client exits, then continue to the next line. See the previous section on running jobs in the background (basically you want to put an ampersand at the end, e.g. `xterm &`).

You can take advantage of this behavior, though. If you want to run commands at the end of your X session, you can have your `.xsession` run a window manager or the like and wait for it to finish. That is, leave off the `exec` and the `&`, just put `fvwm` by itself. Then put the commands of your choice after `fvwm`.

It would probably help to look at a few sample `.xsession` files. In all the examples, replace `fvwm` with the window manager of your choice.

The simplest `.xsession` just runs a window manager:

```
exec fvwm
```

This will run fvwm, and the X session will end when fvwm exits. If you do it without the `exec`, everything will appear to behave the same way, but behind the scenes `.xsession` will hang around waiting for fvwm, and `.xsession` will exit after fvwm does. Using `exec` is slightly better because fvwm replaces `.xsession` instead of leaving it waiting. You can use the `ps` or `top` command to verify this.

A more useful `.xsession` runs a few clients before starting the window manager. For example, you might want some xterms and an xclock whenever you start X. No problem:

```
xterm &
xterm &
xclock &
exec fvwm
```

Two xterms and an xclock start up in the background, and then the window manager is launched. When you quit the window manager, you'll also quit X.

You might try it without the backgrounding just to see what happens. Do this:

```
xterm
xclock
exec fvwm
```

xterm will start, and wait for you to exit it. Then xclock will start; you'll have to exit xclock before fvwm will start. The commands are run in sequence, since the script waits for each one to exit.

You can use sequential execution to your advantage. Perhaps you want to keep track of when you stop working every day:

```
xterm &
xclock &
fvwm
date >> ~/logout-time
```

This will fork off an xterm and an xclock, then run fvwm and wait for it to finish. When you exit fvwm, it will move on to the last line, which appends the current date and time to the file `˜/logout-time`.

Finally, you can have a program other than the window manager determine when X exits:

```
xclock &
fvwm &
exec xterm
```

This script will run xclock and fvwm in the background, and then replace itself with xterm. When you exit the xterm, your X session will end.

The best way to learn how to use `.xsession` is to try some of these things out. Again, be sure you use `chmod` to make it executable — this is a common error.

Chapter 11

Text tools

head, tail, grep, wc, tr, sed, perl and so on

11.1 Regular expressions

A regular expression is a description of a set of characters. This description can be used to search through a file by looking for text that *matches* the regular expression. Regular expressions are analagous to shell wildcards (see ‘Filename expansion ("Wildcards")’ on page 37), but they are both more complicated and more powerful.

A regular expression is made up of text and *metacharacters*. A metacharacter is just a character with a special meaning. Metacharacters include: `.  * [] - \ ^ $`.

If a regular expression contains only text (no metacharacters), then it matches that text. For example, the regular expression ’`my regular expression`’ matches the text ’`my regular expression`’, and nothing else. Regular expressions are usually case-sensitive.

You can use the `egrep` command to display all lines in a file which contain a regular expression. Its syntax is:

`egrep '`*`regexp`*`'` *`filename1`* `...` [1]

For example, to find all lines in the GPL which contain the word GNU, you type:

```
egrep 'GNU' /usr/doc/copyright/GPL
```

`egrep` will print the lines to standard output.

[1]The single quotes are not always needed, but they never hurt.

If you want all lines which contain `freedom`, followed by some indeterminate text, followed by `GNU`, you can do:

```
egrep 'freedom.*GNU' /usr/doc/copyright/GPL
```

The `.` means "any character"; the `*` means "zero or more of the preceding thing," in this case "zero or more of any character." So `.*` matches pretty much any text at all. `egrep` only matches on a line-by-line basis, so `freedom` and `GNU` have to be on the same line.

Here's a summary of regular expression metacharacters:

`.` Matches any single character except newline.

`*` Matches zero or more occurences of the preceding thing. So the expression `a*` matches 0 or more lowercase `a`, and `.*` matches zero or more characters.

`[`*`characters`*`]` The brackets must contain one or more characters; the whole bracketed expression matches exactly one character out of the set. So `[abc]` matches one `a`, one `b`, or one `c`; it does not match 0 characters, and it does not match a character other than these three.

`^` Anchors your search at the beginning of the line. The expression `^The` matches `The` only at the beginning of a line; there can't be spaces or other text before `The`. If you want to allow spaces, you can permit 0 or more space characters like this: `^ *The`.

`$` Anchors at the end of the line. `end$` requires the text `end` to be at the end of the line, with no intervening spaces or text.

`[^`*`characters`*`]` `^` reverses the sense of a bracketed character list. So `[^abc]` matches any single character, *except* a, b, or c.

`[`*`character-character`*`]` You can include ranges in a bracketed character list. To match any lowercase letter, use `[a-z]`. You can have more than one range; so to match the first three or last three letters of the alphabet, try `[a-cx-z]`. To get any letter, any case, try `[a-zA-Z]`. You can mix ranges with single characters and with the `^` metacharacter; for example, `[^a-zBZ]` means "anything except a lowercase letter, capital B, or capital Z."

`()` You can use parentheses to group parts of the regular expression, just as you do in a mathematical expression

`|` | means "or" — you can use it to provide a series of alternative expressions. Usually you want to put the alternatives in parentheses, like this: `c(ad|ab|at)` matches `cad` or `cab` or `cat`. Without the parentheses, it would match `cad` or `ab` or `at` instead

**** Escapes any special characters; if you want to find a literal `*`, you type `\*`. The slash means to ignore `*`'s usual special meaning.

Here are some more examples, to help you get a feel for things:

`c.pe` matches `cope`, `cape`, `caper`

`c\.pe` matches `c.pe`, `c.per`

`sto*p` matches `stp`, `stop`, `stoop`

`car.*n` matches `carton`, `cartoon`, `carmen`

`xyz.*` matches `xyz` and anything after it; some tools, like `egrep`, only match until the end of the line.

`^The` matches `The` at the beginning of a line

`atime$` matches `atime` at the end of a line

`^Only$` matches a line which consists solely of the word `Only` — no spaces, no other characters, nothing. Only `Only` is allowed

`b[aou]rn` matches `barn`, `born`, `burn`

`Ver[D-F]` matches `VerD`, `VerE`, `VerF`

`Ver[^0-9]` matches `Ver` followed by any non-digit

`the[ir][re]` matches `their`, `therr`, `there`, `theie`

`[A-Za-z][A-Za-z]*` matches any word which consists of only letters, and at least one letter. Will not match numbers or spaces

Chapter 12

File tools

12.1 Backup tools

tar, cpio, dump; also large-scale copying, cp -a etc.

(Perhaps something on how to back up only /home and /etc if you only have a floppy drive, since many home users won't have a tape drive)

How to use tar to copy lots of files, or back up your files. Tarballs. I'm thinking this should be a brief section aimed at single-user systems, with a more thorough sysadmin discussion in a different manual.

Backup commands (contributed by Oliver Elphick, section to be cleaned up and elaborated):

dump - dumps one filesystem at a time; its command options assume that you are using half-inch tape (maximum 45Mb per reel) so it's a bit annoying when using DAT (2Gb or more). Probably the best for regular backups. Can't be used for NFS-mounted filesystems.

cpio - 'find [directories] -print | cpio -ovH newc -B >/dev/st0'

tar - 'tar cvf /dev/st0 [directories]'

afio - like cpio; supports pre-compression of files before archiving.

tob - front-end for afio

12.2 File compression with `gzip`

Often it would be nice to make a file smaller: say to download it faster, or so it takes up less space on your disk. The program to do this is called `gzip` (GNU Zip).

1. `cd; cp /etc/profile ./mysamplefile`

Switch to your home directory, then copy an arbitrarily chosen file (`/etc/profile`) to your current directory in the process renaming it `mysamplefile`. This gives us a file to play with using `gzip`.

2. `ls -l`

   List the contents of the current directory. Note the size of `mysamplefile`.

3. `gzip mysamplefile`

   Compress `mysamplefile`.

4. `ls -l`

   Observe the results: `mysamplefile` is now called `mysamplefile.gz`. It's also a good bit smaller.

5. `gunzip mysamplefile.gz; ls -l`

   Uncompress. Observe that `mysamplefile` has returned to its original state. Notice that to uncompress one uses `gunzip`, not `gzip`.

6. `rm mysamplefile`

   Remove the file, since it was just to practice with.

12.3 Splitting files into smaller pieces

Sometimes a file is too big to fit on a disk, or you don't want to send a huge file over the net in a single chunk. You can split the file using the `split` utility, and reassemble it using the `cat` (con*cat*enate) utility.

1. `cd; cp /bin/bash myfile; ls -l myfile`

   Copy the `bash` executable to a file in your home directory called `myfile`. Observe that `myfile` occupies a little over 400,000 bytes, or around 400 kilobytes.

2. `split -b100k myfile myprefix`

   Splits the file into sections of 100 kilobytes, naming the sections `myprefixaa`, `myprefixab`, etc. Type `ls -l` so see the results.

   You can specify any number after the `-b`: choose one that makes sense. If you leave off the `k`, it will be understood as bytes instead of kilobytes. If you use `m` instead of `k`, it will be understood as megabytes.

3. `cat myprefix* > mynewfile`

   Concatenate all the files and write them to `mynewfile`. (The `*` and `>` are tricks you'll learn in another chapter .)

4. `rm myfile mynewfile myprefix*`

   Remove everything.

12.4 Finding files

There are two different facilities for finding files: `find` and `locate`. `find` searches the actual files in their present state. `locate` searches an index generated by the system every morning at 6:42 a.m. (this is a `cron` job, explained elsewhere in this manual). `locate` won't find any files which were created after the index was generated. However, since `locate` searches an index, it's much faster - like using the index of a book rather than looking through the whole thing.

To compare the two ways of finding files, pretend you can't remember where the X configuration file `XF86Config` resides.

1. `locate XF86Config`

   This should be pretty fast. You'll get a list of filenames which *contain* `XF86Config`, something like this:

   ```
   /etc/X11/XF86Config
   /usr/X11R6/lib/X11/XF86Config
   /usr/X11R6/lib/X11/XF86Config.eg
   /usr/X11R6/man/man5/XF86Config.5x.gz
   ```

2. `find / -name XF86Config`

   You will hear a lot of disk activity, and this will take a lot longer. Results will look something like this:

   ```
   /etc/X11/XF86Config
   /usr/X11R6/lib/X11/XF86Config
   find: /var/spool/cron/atjobs: Permission denied
   find: /var/spool/cron/atspool: Permission denied
   find: /var/lib/xdm/authdir: Permission denied
   ```

   Notice that `find` only found files which were named *exactly* `XF86Config`, rather than any files containing that string of letters. Also, `find` actually tried to look in every directory on the system - including some where you didn't have read permissions. Thus the "Permission denied" messages.

   The syntax is different as well. You had to specify what directory to search in — `/` — while `locate` automatically chose the root directory. And you had to specify a search by name, using the `-name` option. You could also have searched for files using many other criteria, such as modification date or owner. To have `find` search for files whose name matches XF86Config, you'd have to use a regular expression: `find / -name '*XF86Config*'`. Like most of the command line tools, `find` accepts regular expressions as arguments.

In general `find` is a more powerful utility, and `locate` is faster for everyday quick searches. The full range of possible searches would take a long time to explain; for more details type `info find`, which will bring up the very thorough info pages on `find` and `locate`.

12.5 Determining a file's contents

Debian comes with a utility which can guess at the contents of a file for you. It is not always correct. However, it is reasonably accurate, and you can use it to explore your system.

1. `file /bin/cp`

   You should see something like this:

   ```
   /bin/cp: ELF 32-bit LSB executable, Intel 386, version 1, stripped
   ```

   Skipping the technical parts, this is an executable file for Intel machines.

2. `file /etc/init.d/boot`

   Gives this response:

   ```
   /etc/init.d/boot: Bourne shell script text
   ```

   Meaning that this is a text file, containing a Bourne shell script.

Chapter 13

Using disks

13.1 Concepts

It's probably a good idea to explain a little theory before discussing the mechanics of using disks. In particular, the concept of a *filesystem*. [1] This is confusing, because it has several meanings.

- *The* filesystem refers to the whole directory tree, starting with the root directory `/`, as described above.
- A "filesystem" in general means any organization of files and directories on a particular physical device. "Organization" means the hierarchical directory structure, and any other information about files one might want to keep track of: their size, who has permission to change them, etc. So you might have one filesystem on your hard disk, and another one on each floppy disk.
- "Filesystem" is also used to mean a *type* of filesystem. For example, MS-DOS and Windows 3.1 organize files in a particular way, with particular rules: filenames can only have 8 characters, for example, and no permissions information is stored. Linux calls this the `msdos` filesystem. Linux also has its own filesystem, called the `ext2` filesystem (version two of the `ext` filesystem). You'll use the `ext2` filesystem pretty much all the time, unless you're accessing files from another operating system or have other special needs.

Any physical device you wish to use for storing files must have at least one filesystem on it. This means a filesystem in the second sense - a hierarchy of files and directories, along with information about them. Of course, any filesystem has a type, so the third sense will come into play as well. If you have more than one filesystem on a single device, each filesystem can have a different type — for example, you might have both a DOS partition and a Linux partition on your hard disk.

[1] Some people spell it as two words, i.e. "fi le system". A quick poll of man pages (`man -k filesystem`, `man -k 'file system'`) reveals about an even split. So I'm spelling it as one word.

It's important to distinguish the filesystem from the low-level format of the disk. In the DOS and Macintosh worlds, the filesystem is called the high-level format. When you format a disk using one of those operating systems, generally you both perform a low-level format and create a file system (high-level format). On GNU and Unix systems, one generally says simply "format" to mean low-level format, and "making a filesystem" to mean high-level format.

Formatting has to do with the particulars of the physical device, such as the exact physical location of your data on a floppy disk (on the edge or near the center of the disk for example). The filesystem is the level of organization you have to worry about — names of directories and files, their sizes, etc.

13.2 `mount` and `/etc/fstab`

This section describes how to mount a floppy or Zip disk, the `/dev` directory, and distributing the directory tree over multiple physical devices or partitions.

13.2.1 Mounting a filesystem

On a GNU/Linux system there's no necessary correspondence between directories and physical devices, as there is in Windows where each drive has its own directory tree beginning with a letter (such as `C:\`).

Instead, each physical device such as a hard disk or floppy disk has one or more filesystems on it. In order to make a filesystem accessible, it's assigned to a particular directory in another filesystem. To avoid circularity, the root filesystem (which contains the root directory `/`) is not contained by any other filesystem — you have access to it automatically when you boot Debian.

A directory in one filesystem which contains another filesystem is known as a *mount point*. A mount point is a directory in a first filesystem on one device (such as your hard disk) which "contains" a second filesystem, perhaps on another device (such as a floppy disk). To access a filesystem, you must mount it at some mount point.

So, for example, you might mount a CD at the mount point `/cdrom`. This means that if you look in the directory `/cdrom`, you'll see the contents of the CD. The `/cdrom` directory itself is actually on your hard disk. For all practical purposes the contents of the CD become a part of the root filesystem, and when typing commands and using programs it doesn't make any difference what the actual physical location of the files is. You could have created a directory on your hard disk called `/cdrom`, and put some files in it, and everything would behave in exactly the same way. Once you mount a filesystem, there's no need to pay any attention to physical devices.

However, before mounting a filesystem, or to actually create a filesystem on a disk that doesn't have one yet, it's necessary to refer to the devices themselves. All devices have names, and these are located in the `/dev` directory. If you type `ls /dev` now, you'll see a pretty lengthy list of every possible device you could have on your Debian system.

Possible devices include:

2

- `/dev/hda` is IDE drive A. In general, this will be a hard drive. IDE refers to the type of drive - if you don't know what it means, you probably have this kind of drive, because it's the most common. Your DOS/Windows `C:\` partition is likely to be on this drive.
- `/dev/hdb` is IDE drive B, as you might guess. This could be a second hard drive, or perhaps a CD-ROM drive. Drives A and B are the first and second (master and slave) drives on the primary IDE controller. Drives C and D are the first and second drives on the secondary controller.
- `/dev/hda1` is the first *partition* of IDE drive A, usually called `C:\` on a DOS or Windows system. Notice that different drives are lettered, while specific partitions of those drives are numbered as well.
- `/dev/sda` is SCSI disk A. SCSI is like IDE, only if you don't know what it is you probably *don't* have one of these drives. They're not very common in home Intel PC's, though they're often used in servers and Macintoshes often have SCSI disks.

  3
- `/dev/fd0` is the first floppy drive, generally `A:\` under DOS. Since floppy disks don't have partitions, they only have numbers, rather than the letter-number scheme used for hard drives. However, for floppy drives the numbers refer to the drive, and for hard drives the numbers refer to the partitions.
- `/dev/ttyS0` is the first of your serial ports (`COM1:` under DOS). `/dev` contains the names of many devices, not just disk drives.

To mount a filesystem, we want to tell Linux to associate whatever filesystem it finds on a particular device with a particular mount point. In the process, we might have to tell Linux what kind of filesystem to look for.

13.2.2 Example: Mounting a CD-ROM

As a simple demonstration, we'll go through mounting a CD-ROM, such as the one you may have used to install Debian. You'll need to be root to do this, so be careful; whenever you're root you have the power to mess up the whole system, rather than just your own files. Also, these commands assume there's a CD in your drive; you should put one in the drive now.

[2] This isn't a comprehensive list. Generally the documentation for a particular device or program will tell you what device name you want to use. There are hundreds of different device names. A pretty complete (through not very detailed) list should be on your system in the fi le `/usr/src/linux/Documentation/devices.txt`.

[3] SCSI devices have a more complicated naming scheme than IDE devices, mostly because SCSI has more uses. The partitions of a SCSI disk have the form `/dev/sda[1-9a-f]`, that is, `/dev/sda` (or `sdb` or `sdc`, etc.) followed by a number or letter from `1--9` or `a--f`. `/dev/scd0` is the fi rst SCSI CDROM device; general devices such as scanners might look like `/dev/sg0`; `/dev/st0` is a SCSI tape drive.

1. `su`

   If you haven't already, you need to either log in as root or gain root privileges with the `su` (super user) command. If you use `su`, enter the root password when prompted.

2. `ls /cdrom`

   See what's in the `/cdrom` directory before you start. If you don't have a `/cdrom` directory, you may have to make one using `mkdir /cdrom`.

3. `mount`

   Typing simply `mount` with no arguments lists the currently mounted filesystems.

4. `mount -t iso9660` *`CD device`* `/cdrom`

   For this command, you should substitute the name of your CD-ROM device for *`CD device`* in the above command line. If you aren't sure, `/dev/cdrom` is a good guess since the install process should have created this symbolic link on the system. If that fails, try the different IDE devices: `/dev/hdc`, etc. You should see a message like:

   ```
   mount: block device /dev/hdc is write-protected, mounting read-only
   ```

   The `-t` option specifies the type of the filesystem, in this case `iso9660`. Most CDs are `iso9660`. The next argument is the name of the device to mount, and the final argument is the mount point. There are many other arguments to `mount`; see the manual page for details.

   Once a CD is mounted, you may find that your drive tray will not open. You must unmount the CD before removing it.

5. `ls /cdrom`

   Confirm that `/cdrom` now contains whatever is on the CD in your drive.

6. `mount`

   Look at the list of filesystems again, noticing that your CD drive is now mounted.

7. `umount /cdrom`

   This unmounts the CD. It's now safe to remove the CD from the drive. Notice that the command is `umount` with no "n", even though it's used to u*n*mount the filesystem.

8. `exit`

   Don't leave yourself logged on as root. Log out immediately, just to be safe.

13.2.3 `/etc/fstab`: Automating the mount process

The file `/etc/fstab` (it stands for "file system table") contains descriptions of filesystems that you mount often. These filesystems can then be mounted with a shorter command, such as `mount /cdrom`. You can also configure filesystems to mount automatically when the system boots. You'll probably want to mount all of your hard disk filesystems when you boot.

Look at this file now, by typing `more /etc/fstab`. It will have two or more entries that were configured automatically when you installed the system. It probably looks something like this:

```
# /etc/fstab: static file system information.
#
# <file system>     <mount point>   <type>  <options>   <dump >  <pass>
/dev/hda1            /               ext2    defaults    0       1
/dev/hda3            none            swap    sw          0       0
proc                 /proc           proc    defaults    0       0

/dev/hda5            /tmp            ext2    defaults    0       2
/dev/hda6            /home           ext2    defaults    0       2
/dev/hda7            /usr            ext2    defaults    0       2

/dev/hdc             /cdrom          iso9660 ro,noauto   0       0
/dev/fd0             /floppy         auto    noauto,sync 0       0
```

The first column lists the device the filesystem resides on. The second lists the mount point, the third the filesystem type. The line beginning by `proc` is a special filesystem . Notice that the swap partition (`/dev/hda3` in the example) has no mount point, so the mount point column contains `none`.

The last three columns may require some explanation.

The fifth column is used by the `dump` utility to decide when to back up the filesystem. In most cases you can put `0` here.

The sixth column is used by `fsck` to decide in what order to check filesystems when you boot the system. The root filesystem should have a `1` in this field, filesystems which don't need to be checked (such as the swap partition) should have a `0`, and all other filesystems should have a `2`. It's worth noting that the swap partition isn't exactly a filesystem in the sence that it does not contain files and directories, but is just used by the Linux kernel as secondary memory. However, for historical reasons, the swap partitions are still listed in the same file than the filesystems.

Column four contains one or more options to use when mounting the filesystem. Here's a brief summary (some of these probably won't make much sense yet — they're here for future reference):

`async` and `sync` Do I/O synchronously or asynchronously. Synchronous I/O writes changes to files immediately, while asynchronous I/O may keep data in buffers and write it later, for efficiency reasons.

`ro` and `rw` Mount the filesystem read-only or read-write. If you don't need to make any changes to the filesystem, it's a good idea to mount it read-only so you don't accidentally mess something up. Also, read-only devices (such as CD-ROM drives and floppy disks with write protection tabs) should be mounted read-only.

`auto` and `noauto` When the system boots, or whenever you type `mount -a`, `mount` tries to mount all the filesystems listed in `/etc/fstab`. If you don't want it to automatically mount a filesystem, you should use the `noauto` option. It's probably a good idea to use `noauto` with removable media such as floppy disks, because there may or may not be a disk in the drive. You'll want to mount these filesystems manually after you put in a disk.

`dev` and `nodev` Use or ignore device files on this filesystem. You might use `nodev` if you mount the root directory of another system on your system — you don't want your system to try to use the devices on the other machine.

`user` and `nouser` Permit or forbid ordinary users to mount the filesystem. `nouser` means that only root can mount the filesystem. This is the normal arrangement. You might use the `user` option to access the floppy drive without having to be root.

`exec` and `noexec` Allow or do not allow the execution of files on this filesystem. Probably you won't need these options.

`suid` and `nosuid` Allow or do not allow the suid bit to take effect. Probably you won't need these options.

`defaults` Equivalent to: `rw`, `dev`, `suid`, `exec`, `auto`, `nouser`, `async`. You can specify `defaults` followed by other options to override specific aspects of `defaults`.

13.2.4 Removable disks (floppies, Zip disks, etc.)

Add the following lines to your `/etc/fstab` file:

```
/dev/sda1  /mnt/zip  ext2    noauto,user     0 0
/dev/sda4  /mnt/dos  msdos   noauto,user     0 0
```

From then on, you'll be able to mount the DOS formated Zip disks with the command `mount /mnt/dos`, and Linux formated Zip disks with the command `mount /mnt/zip`. [4]

[4] If you have SCSI hard disks in your system, you'll have to change `sda` by `sdb` or `sdc`, etc. .. in the example above.

13.3 PPP

13.3.1 Introduction

If you connect to the internet over a phone line, you'll want to use PPP (Point-To-Point Protocol). This is the standard connection method offered by ISPs (Internet Service Providers). In addition to using PPP to dial your ISP, you can have your computer listen for incoming connections — this lets you dial your computer from a remote location.

This section is a quick-start no-frills guide to setting up PPP on Debian. If it turns out that you need more details, see the excellent PPP HOWTO (`http://metalab.unc.edu/LDP/HOWTO/PPP-HOWTO.html`) from the Linux Documentation Project. The HOWTO goes into much more detail if you're interested or have unique needs.

13.3.2 Preparation

Configuring PPP on GNU/Linux is straightforward once you have all the information you'll need. Debian makes things even easier with its simple configuration tools.

Before you start, be sure you have all the information provided by your ISP. This might include:

- Username or login
- Password
- Your static IP (Internet Protocol) address, if any (these look like `209.81.8.242`)
- Bitmask (this will look something like `255.255.255.248`)
- The IP addresses of your ISPs name server (or DNS).
- Any special login procedure required by the ISP.

Next, you'll want to investigate your hardware setup: whether your modem works with GNU/Linux, and which serial port it's connected to.

There's a simple rule which determines whether your modem will work. If it's a "WinModem" or "host-based modem", it won't work. These modems are cheap because they have very little functionality, and require the computer to make up for their shortcomings. Unfortunately, this means they are complex to program, and manufacturers generally do not make the specifications available for developers.

If you have a modem with its own on-board circuitry, you should have no trouble at all.

On GNU/Linux systems, the serial ports are referred to as `/dev/ttyS0`, `/dev/ttyS1`, and so on. Your modem is almost certainly connected to either port 0 or port 1, equivalent to `COM1:` and `COM2:` under

Windows. If you don't know which your modem is connected to, `wvdialconf` can try to detect it (see below); otherwise just try both and see which works.

If you want to talk to your modem or dial your ISP without using PPP, you can use the `minicom` program. You may need to install the minicom package before the program is available.

13.3.3 The Easy Way: `wvdial`

The simplest way to get PPP running is with the `wvdial` program. It makes some reasonable guesses and tries to set things up for you. If it works, you're in luck. If it guesses wrong, you'll have to do things manually.

Be sure you have the following packages installed:

- `ppp`
- `ppp-pam`
- `wvdial`

When you install the `wvdial` package, you may be given the opportunity to configure it. Otherwise, to set up `wvdial`, follow these simple steps:

1. Login as root, using `su` as described in an earlier chapter
2. `touch /etc/wvdial.conf`

   `touch` will create an empty file if the file doesn't exist — the configuration program requires an existing file.
3. `wvdialconf /etc/wvdial.conf`

   This means you're creating a configuration file, `/etc/wvdial.conf`
4. Answer any questions that appear on the screen. `wvdialconf` will also scan for your modem and tell you which serial port it's on; you may want to make a note of this for future reference.
5. `/etc/wvdial.conf` should look something like this now:

   ```
   [Dialer Defaults]
   Modem = /dev/ttyS1
   Baud = 115200
   Init1 = ATZ
   Init2 = ATQ0 V1 E1 S0=0 S11=55 +FCLASS=0
   ; Phone = [Target Phone Number]
   ; Username = [Your Login Name]
   ; Password = [Your Password]
   ```

Just replace the information in brackets with the proper information and remove the semicolons from the beginning of those lines and you're done! Here is what a completed `wvdial.conf` file should look like:

```
[Dialer Defaults]
Modem = /dev/ttyS1
Baud = 115200
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 S11=55 +FCLASS=0
Phone = 5551212
Username = beavis
Password = password
```

Now that `wvdial.conf` is set up, to connect to your ISP just type `wvdial`. If it doesn't work, you'll probably have to delve into manual PPP configuration.

13.3.4 Doing It Manually

This still isn't all that difficult, though it's slightly harder than `wvdial`. The quick-and-easy summary: type `pppconfig` as root, answer the questions, then type `pon` to log on, and `poff` to log off. We'll go into a little more detail though.

Chapter 14

Removing and installing software

14.1 The `dpkg` package utility

14.2 What a package maintenance utility does

An application or utility program usually involves quite a few files. It might involve libraries, data files like game scenarios or icons, configuration files, manual pages and documentation. When you install the program, you want to make sure you have all the files you need in the right place.

You'd also like to be able to uninstall the program. When you uninstall, you want to be sure all the associated files are deleted. However, if a program you still have on the system needs those files, you want to be sure you keep them.

Finally, you'd like to be able to upgrade a program. When you upgrade, you want to delete obsolete files and add new ones, without breaking any part of the system.

The Debian package system solves these problems. It allows you to install, remove, and upgrade software *packages*, which are neat little bundles containing the program files and information that helps the computer manage them properly. Debian packages have filenames ending in the extension `.deb`, and they're available on the ftp site or on your official Debian CD-ROM.

14.3 Apt

14.3.1 Confi guring Apt

Debian now supplies a tool named Apt (for "A Package Tool") to help the administrators to add or remove software more easily. Your first task will be to customize the `/etc/apt/sources.list` configuration file. This package resource list is used to locate archives of the package distribution system in use on the system. The source list is designed to support any number of active sources and a variety of source media. The file lists one source per line, with the most preferred source listed first. The format of a `sources.list` entry is:

```
deb uri distribution [component1] [component2] [...]
```

The URI for the deb type must specify the base of the Debian distribution, from which APT will find the information it needs. distribution can specify an exact path, in which case the components must be omitted and distribution must end with a slash (/). This is useful for when only a particular sub-section of the archive denoted by the URI is of interest. If distribution does not specify an exact path, at least one component must be present.

The currently recognized URI types are cdrom, file, http, and ftp.

`file` The file scheme allows an arbitrary directory in the file system to be considered an archive. This is useful for NFS mounts and local mirrors or archives.

`cdrom` The cdrom scheme allows APT to use a local CDROM drive with media swapping. Use the apt-cdrom(8) program to create cdrom entires in the source list.

`http` The http scheme specifies an HTTP server for the archive. If an environment variable $http_proxy is set with the format http://server:port/, the proxy server specified in $http_proxy will be used. Users of authenticated HTTP/1.1 proxies may use a string of the format http://user:pass@server:port/. Note that this is an insecure method of authentication.

`ftp` The ftp scheme specifies an FTP server for the archive. APT´s FTP behavior is highly configurable; for more information see the apt.conf(5) manual page.

`copy` The copy scheme is identical to the file scheme except that packages are copied into the cache directory instead of used directly at their location. This is usefull for people using a zip disk to copy files around with APT.

A few examples:

```
deb http://www.debian.org/archive stable main contrib
```

Uses HTTP to access the archive at www.debian.org, and uses the stable/main and stable/contrib areas.

```
deb ftp://ftp.debian.org/debian unstable main contrib non-free
```

Uses FTP to access the archive at ftp.debian.org, under the /debian directory, and uses the unstable/main, unstable/contrib and unsunstable/non-free areas.

```
deb ftp://ftp.debian.org/debian stable main
```

Uses FTP to access the archive at ftp.debian.org, under the /debian directory, and uses the stable/main area.

If this line appears as well as the one in the previous example in `sources.list`, a single FTP session will be used for both resource lines.

```
deb file:/home/vincent/debian stable main contrib non-free
```

Uses the archive stored locally (or NFS mounted) at /home/vincent/debian for stable/main, stable/contrib, and stable/non-free.

14.3.2 Using apt-get.

`apt-get` is the command-line tool for handling packages, and may be considered the user's "back-end" to apt. `apt-get` is very straightforward to use.

```
apt-get [options] [command] [package ...]
```

Where *command* is one of:

`update` update is used to resynchronize the package overview files from their sources. The overviews of available packages are fetched from the location(s) specified in `/etc/apt/sources.list`. For example, when using a Debian archive, this command retrieves and scans the Packages.gz files, so that information about new and updated packages is available. An update should always be performed before an `upgrade dist-upgrade`.

`upgrade` upgrade is used to install the newest versions of all packages currently installed on the system from the sources enumerated in `/etc/apt/sources.list`. Packages currently installed with new versions available are retrieved and upgraded; under no circumstances are currently installed packages removed, or packages not already installed retrieved and installed. New versions of currently installed packages that cannot be upgraded without changing the install status of another package will be left at their current version. An `apt-get update` must be performed first so that apt-get knows that new versions of packages are available.

`dist-upgrade` *dist-upgrade*, in addition to performing the function of *upgrade*, also intelligently handles changing dependencies with new versions of packages; apt-get has a "smart" conflict resolution system, and it will attempt to upgrade the most important packages at the expense of less important ones if necessary. The `/etc/apt/sources.list` file contains a list of locations from which to retrieve desired package files.

`install` *install* is followed by one or more packages desired for installation. Each package is a package name, not a fully qualified filename (for instance in a Debian GNU/Linux system, *lsdo* would be the argument provided, not *ldso_1.9.6-2.deb*). All packages required by the package(s) specified for installation will also be retrieved and installed. The `/etc/apt/sources.list` file is used to locate the desired packages. If a hyphen is appended to the package name (with no intervening space), the identified package will be removed if it is installed. This latter feature may be used to override decisions made by *apt-get*'s conflict resolution system.

`remove` *remove* is identical to install except that packages are removed instead of installed. If a plus sign is appended to the package name (with no intervening space), the identified package will be installed.

`check` *check* is a diagnostic tool; it updates the package cache and checks for broken packages.

`clean` *clean* clears out the local repository of retrieved package files. It removes everything but the lock file from `/var/cache/apt/archives/` and `/var/cache/apt/archives/partial/`.

The most usefull options are:

`-m` Ignore the missing packages if any.

`-d` Only downloads the necessary packages, but don't install them.

`-f` Fix broken dependancies; Run `apt-get -f install` to just automagically repair the dependancy problems on your system.

14.4 Using dselect.

14.5 Using dpkg manually

The simplest way to install a single package you've downloaded is the command `dpkg -i` (short for `dpkg --install`). Say you've downloaded the package `icewm_0.8.12-1.deb` and you'd like to install it. First log on as root, then type:

```
dpkg -i icewm_0.8.12-1.deb
```

and icewm version 0.8.12 will be installed. If you already had an older version, dpkg will upgrade it rather than installing both versions at once.

If you want to remove a package, you have two options. The first is most intuitive:

```
dpkg -r icewm
```

This will remove the icewm package (`-r` is short for `--remove`). Note that you give only the ’icewm’ for `--remove`, while `--install` requires the entire `.deb` filename.

`--remove` will leave configuration files for the package on your system. A configuration file is defined as any file you might have edited in order to customize the program for your system or your preferences. This way, if you later reinstall the package, you won’t have to set everything up a second time.

However, you might want to erase the configuration files too, so dpkg also provides a `--purge` option. `dpkg --purge icewm` will permanently delete every last file associated with the icewm package.

14.6 Compiling software yourself

You’ll have to have -dev packages installed.

Put it in /usr/local, /opt, or your home directory.

The configure –prefix; make; make install routine.

14.7 Proprietary software not in .deb format

What to do with this annoying stuff (wrapper packages, /usr/local)

Chapter 15

Troubleshooting

15.1 Debian is frozen or crashed!

If your Debian system freezes up, it is possibly a bug and the system has crashed. However, it's fair to say that this doesn't happen very often — many people go for months or even years without a crash, even under heavy use. GNU/Linux is extremely reliable. If you have problems more often than this, something is probably wrong; ask for help (see 'Getting help from a person' on page 28).

However, it is much more common for the screen to freeze due to a bug in the X server, especially when switching to and from the X virtual console. The newer X servers are more likely to do this, or simply crash; it's hard for the XFree86 project to keep up with the latest features of every video card available.

If the screen freezes, the system is usually still alive, you simply can't interact with it from the console. If you're on a network, you should be able to log in from another workstation. Once logged in you can reboot in an orderly way (i.e. type `reboot`). If you aren't on a network, pressing `C-A-DEL` will work most of the time. If neither of these works, you may have to simply shut down the system and let `fsck` repair the disk on your next boot.

15.2 My terminal isn't behaving properly

Sometimes you terminal will get stuck in reverse video mode, will refuse to move to a new line when you press return, will print garbage characters, or otherwise not work properly. This is almost always the result of attempting to view a binary file with `cat` or `more` (sometimes it's also caused by a bug in a program). You can try to recover by entering the `reset` command. If it did nothing, try typing `echo C-vESCc` (ie: press C-v, then ESC, then 'c'). If that fails, log out and back in.

The problem is that terminals are controlled with special *control sequences* which tell them to enter reverse video mode, position the cursor, etc. If you send a binary file to the terminal, the file may happen to contain these control sequences. Thus the terminal will get confused.

15.3 The computer beeps when I press a key, and my screen shows a text file and/or a lot of ~ symbols.

You've inadvertently gotten stuck in `vi`. See 'Creating and editing a text file with `vi`' on page 49.

Chapter 16

Advanced topics

(Should advanced topics be here? I think it would be nice to have some of these, just to show people the possibilities and give some conceptual explanation that won't really be in a reference manual. Also it always feels nice to make it to a chapter called "advanced topics". Self-esteem booster for the newbie. :)

16.1 Introduction to shell scripting

16.1.1 What and why

Automate simple tasks.

16.1.2 A simple example

Ideas?

16.2 Advanced files

16.2.1 The real nature of files: hard links and inodes

Each file on your system is represented by an *inode* (for Information Node; pronounced "eye-node"): an inode contains all the information about the file. However, the inode is not directly visible. Instead, each inode is linked into the filesystem by one or more *hard links*. Hard links contain the name of the file, and the inode number. The inode contains the file itself, i.e., the location of the information being stored on disk, its access permissions, the type of the file, and so on. The system can find any inode once it has the inode number.

A single file can have more than one hard link. What this means is that multiple filenames refer to the same file (that is, they are associated with the same inode number). However, you can't make hard links across filesystems: all hard references to a particular file (inode) must be on the same filesystem. This is because each filesystem has its own set of inodes, and there can be duplicate inode numbers between filesystems.

Since all hard links to a given inode are referring to *the same file*, you can make changes to the file, referring to it by one name, and then see those changes when referring to it by a different name. Try this:

1. `cd; echo "hello" > firstlink`

   `cd` to your home directory and create a file called `firstlink` containing the word "hello". What you've actually done is redirect the output of `echo` (`echo` just echoes back what you give to it), placing the output in `firstlink`. See the chapter on shells for a full explanation.

2. `cat firstlink`

   Confirm the contents of `firstlink`.

3. `ln firstlink secondlink`

   Create a hard link: `secondlink` now points to the same inode as `firstlink`.

4. `cat secondlink`

   Confirm that `secondlink` is the same as `firstlink`

5. `ls -l`

   Notice that the number of hard links listed for `firstlink` and `secondlink` is 2.

6. `echo "change" » secondlink`

   This is another shell redirection trick - don't worry about the details. We've appended the word "change" to `secondlink`. Confirm this with `cat secondlink`.

7. `cat firstlink`

   `firstlink` also has the word "change" appended! It's because `firstlink` and `secondlink` refer to *the same file*. It doesn't matter what you call it when you change it.

8. `chmod a+rwx firstlink`

   Change permissions on `firstlink`. Do `ls -l` to confirm that permissions on `secondlink` were also changed. This means that permissions information is stored in the inode, not in links.

9. `rm firstlink`

   Delete this link. This is a subtlety of `rm` — it really removes links, not files. Now type `ls -l` and notice that `secondlink` is still there. Also notice that the number of hard links for `secondlink` has been reduced to one.

10. `rm secondlink`

    Delete the other link. When there are no more links to a file, Linux deletes the file itself, that is, its inode.

All files work like this — even special types of files such as devices (e.g. `/dev/hda`).

A directory is simply a list of filenames and inode numbers, that is, a list of hard links. When you create a hard link, you're just adding a name-number pair to a directory. When you delete a file, you're just removing a hard link from a directory.

16.2.2 Types of files

One detail we've been concealing up to now is that the Linux kernel considers nearly everything to be a file. That includes directories and devices: they're just special kinds of files.

As you may remember, the first character of an `ls -l` display represents the type of the file. For an ordinary file, this will be simply `-`. Other possibilities are:

- `d` (directory)
- `l` (symbolic link)
- `b` (block device)
- `c` (character device)
- `p` (named pipe)
- `s` (socket)

Symbolic links

Symbolic links (also called symlinks or soft links) are the other kind of link besides hard links. A symlink is a special file that "points to" a hard link on any mounted filesystem. When you try to read the contents of a symlink, it gives the contents of the file it's pointing to rather than the contents of the symlink itself. Since directories, devices, and other symlinks are types of files, you can point a symlink at any of those things.

So a hard link is a filename and an inode number. A file is really an inode: a location on disk, file type, permissions mode, etc. A symlink is an inode that contains the name of a hard link. A symlink pairs one filename with a second filename, while a hard link pairs a filename with an inode number.

All hard links to the same file have equal status. That is, one is as good as the other; if you perform any operation on one it's just the same as performing that operation on any of the others. This is because the hard links all refer to the same inode. Operations on symlinks, on the other hand, sometimes affect the symlink's own inode (the one containing the name of a hard link) and sometimes affect the hard link being pointed to.

There are a number of important differences between symlinks and hard links:

- Symlinks can cross filesystems. This is because they contain complete filenames, starting with the root directory, and all complete filenames are unique. Since hard links point to inode numbers, and inode numbers are unique only within a single filesystem, they would be ambiguous if the filesystem wasn't known.
- You can make symlinks to directories, but you can't make hard links to them. Each directory has hard links — its listing in its parent directory, its `.` entry, and the `..` entry in each of its subdirectories — but to impose order on the filesystem, no other hard links to directories are allowed. Consequently, the number of files in a directory is equal to the number of hard links to that directory minus two (you subtract the directory's name and the `.` link).
- You can only make a hard link to a file that exists, because there must be an inode number to refer to. However, you can make a symlink to any filename, whether or not there actually is such a filename.
- Removing a symlink removes only the link. It has no effect on the linked-to file. Removing the only hard link to a file removes the file.

Try this:

1. `cd; ln -s /tmp/me MyTmp`

   `cd` to your home directory. `ln` with the `-s` option makes a symbolic link; in this case, one called `MyTmp` which points to the filename `/tmp/me`.

2. `ls -l MyTmp`

   Output should look like this:

   ```
   lrwxrwxrwx   1 havoc    havoc           7 Dec  6 12:50 MyTmp -> /tmp/me
   ```

   The date and user/group names will be different for you, of course. Notice that the file type is `l`, indicating that this is a symbolic link. Also notice the permissions - symbolic links always have these permissions. If you attempt to `chmod` a symlink, you'll actually change the permissions on the file being pointed to.

3. `chmod 700 MyTmp`

   You will get a "No such file or directory" error, because the file `/tmp/me` doesn't exist. Notice that you could create a symlink to it anyway.

4. `mkdir /tmp/me`

   Create the directory `/tmp/me`.

5. `chmod 700 MyTmp`

   Should work now.

6. `touch MyTmp/myfile`

   Create a file in `MyTmp`.

7. `ls /tmp/me`

   The file was actually created in `/tmp/me`.

8. `rm MyTmp`

   Remove the symbolic link. Notice that this removes the link, not what it points to. Thus you use `rm` not `rmdir`.

9. `rm /tmp/me/myfile; rmdir /tmp/me`

   Clean up after ourselves.

Device files

Device files refer to physical or virtual devices on your system, such as your hard disk, video card, screen, or keyboard. An example of a virtual device is the console, represented by `/dev/console`.

There are two kinds of devices: *character devices* can be accessed one character at a time, that is, the smallest unit of data which can be written to or read from the device is a character (byte).

Block devices must be accessed in larger units called blocks, which contain a number of characters. Your hard disk is a block device.

You can read and write device files just as you can from other kinds of files, though the file may well contain some strange incomprehensible-to-humans gibberish. Writing random data to these files is probably a Bad Idea. Sometimes it’s useful, though: for example, you can dump a postscript file into the printer device `/dev/lp0`, or send modem commands to the device file for the appropriate serial port.

`/dev/null` `/dev/null` is a special device file that discards anything you write to it. If you don’t want something, throw it in `/dev/null`. It’s essentially a bottomless pit. If you read `/dev/null`, you’ll get an end-of-file (EOF) character immediately. `/dev/zero` is similar, only if you read from it you get the `\0` character (not the same as the number zero).

Named pipes (FIFOs)

A named pipe is a file that acts like a pipe. You put something into the file, and it comes out the other end. Thus it's called a FIFO, or First-In-First-Out: the first thing you put in the pipe is the first thing to come out the other end.

If you write to a named pipe, the process which is writing to the pipe doesn't terminate until the information being written is read from the pipe. If you read from a named pipe, the reading process waits until there's something to read before terminating. The size of the pipe is always zero — it doesn't store data, it just links two processes like the shell `|`. However, since this pipe has a name, the two processes don't have to be on the same command line or even be run by the same user.

You can try it by doing the following:

1. `cd; mkfifo mypipe`

   Makes the pipe.

2. `echo "hello" > mypipe &`

   Puts a process in the background which tries to write "hello" to the pipe. Notice that the process doesn't return from the background; it is waiting for someone to read from the pipe.

3. `cat mypipe`

   At this point the `echo` process should return, since `cat` read from the pipe, and the `cat` process will print `hello`.

4. `rm mypipe`

   You can delete pipes just like any other file.

Sockets

Sockets are similar to pipes, only they work over the network. This is how your computer does networking: you may have heard of "WinSock", which is sockets for Windows.

We won't go into these further, because you probably won't have occasion to use them unless you're programming. However, if you see a file marked with type `s` on your computer, you know what it is.

16.2.3 The `proc` filesystem

The Linux kernel makes a special filesystem available, which is mounted under `/proc` on Debian systems. This is a "pseudo-filesystem" — it doesn't really exist on any of your physical devices.

The `proc` filesystem contains information about the system and running processes. Some of the "files" in `/proc` are reasonably understandable to humans (try typing `cat /proc/meminfo` or `cat /proc/cpuinfo`) while some others are arcane collections of numbers. Often, system utilities use these to gather information and present it to you in a more understandable way.

People frequently panic when they notice one file in particular — `/proc/kcore` — which is generally huge. This is (more or less) a copy of the contents of your computer's memory. It's used to debug the kernel. It doesn't actually exist anywhere, so don't worry about its size.

If you want to know about all the things in `/proc`, type `man 5 proc`.

16.2.4 Advanced aspects of file permissions

Using numeric arguments with `chmod`

Earlier in this chapter, we briefly mentioned that you can set file permissions using numbers. The numeric notation is called an absolute mode, as opposed to the symbolic notation (e.g. `u+rx`) which is often called a relative mode. This is because the number specifies an exact mode to set, and the symbol just specifies a change to make (e.g. "add user read and execute permissions").

The numeric mode is a series of four octal digits or twelve binary digits. Each octal (base eight) digit represents three binary digits: one octal digit and three binary digits are two ways to represent the decimal digits 0 through 7.

Deriving a particular mode is pretty straightforward. You simply add up the modes you want to combine, or subtract modes you don't want. For example, user permissions, with only read permission turned on, would be `100` in binary. User permissions with write only would be `010` binary. User permissions with read and write both turned on would be `100 + 010 = 110`. Alternatively, you could put it in octal: `4 + 2 = 6`.

For the full mode, simply add up digits from this table:

0001	others, execute
0002	others, write
0004	others, read
0010	group, execute
0020	group, write
0040	group, read
0100	user, execute
0200	user, write
0400	user, read
1000	sticky bit
2000	set group id
4000	set user id

To use the table, first decide what permissions you want to set. Then add up the numbers for those permissions. The total is your mode. For example, to get mode `0755`:

```
  0001   o=x
  0004   o=r
  0010   g=x
  0040   g=r
  0100   u=x
  0200   u=w
+ 0400   u=r
-------
  0755  u=rwx go=rw
```

You'd actually call this mode simply `755`, without the leading `0`, because `chmod` automatically adds zeroes at the beginning of the mode — `7` means mode `0007`.

To set a file to `755`, you'd type `chmod 755 myfile`.

`755` is a very common mode for directories, as it allows anyone to use the directory but only the owner to create and delete files in the directory. `644` is the analogous mode for files, and it is also very common. It allows anyone to use the file but only the owner can change it. For executable files, `755` is a common mode; this is just `644` plus execute permissions (`644 + 111 = 755`).

16.2.5 chattr

A useful tip?

16.2.6 Large-scale copying

cp -a and variants on the theme.

how to copy an old system to a new one.

FIXME whoops, I also listed this topic under Backup Tools. need to decide.

16.2.7 Other concepts not yet covered, but should be

fsck, dd, fdisk, etc.

what package is a file in?

MSDOS vs. Mac vs. Unix text files

sync

16.3 Compiling the kernel

How, what, and why

16.4 A few words on security

The basics of security from a user standpoint. Maintaining one's privacy. What other users can see of your account.

16.5 Programming on Linux

Something about the Linux programming environment. Aimed at, say, people taking CS101. Nothing on *how* to program, just Emacs, gcc, gdb, ddd, etc. as programming tools.

Likely based on debug.tex

Chapter 17

Where to from here?

17.1 Other Debian manuals to read

17.2 Other resources

A few good URLs, directions to meta-documents.

Chapter 18

Contributing to Debian: How can I help?

18.1 Submit bug reports

How to do that with the "bug" package

18.2 Other things

Whatever we want to say here.

Appendix A

A brief survey of available applications

(Would this be useful? Or does dselect really do the job? It seems like there are a significant number of debian-user and newsgroup questions to the effect of "I'm looking for an application that does X". We could just list what exists, more or less, maybe recommend one option over another when there's an obvious choice.)

A.1 (Subheadings could copy the structure of the menu system)

Appendix B

Summary of GNU/Linux system components

You may be a little confused about exactly what GNU/Linux is. What's the Linux kernel, vs. the GNU utilities, vs. the X Window System? Here's a little ASCII art diagram to show you how it all relates (apologies, it's not very attractive).

```
 Window Manager ----- Gnome Programs/Libraries (in development)
 |   |     ----------/                     \
 |   |     /                                 ---------\
 | X Clients         Programs/Applications ---- Shared Libraries
 |   |   \                    |                 /
  \  |    ---------------\    |     /-----------
  X Server -------------- Linux Kernel
      |                   /    |
      |   /---------------     |
  Video Card -------- Other Computer Hardware (Memory, disk, etc.)
```

Note that the X Server and X Clients *are* applications, but they are applications which interact in special ways with the video hardware and other applications.

The window manager is a special X Client, and thus also an application. All the X programs use the Shared Libraries, but no line appears in the diagram.

When the Gnome (GNU Network Object Model Environment) project is completed, it will help to unify the X environment with a special set of programs and libraries, interacting with the window manager and other X Clients.

The GNU utilities are small programs that do simple tasks: `mv`, `cp`, `tar` for example.

Appendix C

Appendix C: Booting the system

This appendix describes what happens during the GNU/Linux boot process.

How you boot your system depends on how you set things up when you installed Debian. Most likely, you just turn the computer on. But you may have to insert a floppy disk first.

Linux is loaded by a program called LILO, or LInux LOader. LILO can also load other operating systems, and ask you which system you'd like to load.

The first thing that happens when you turn on an Intel PC is that the BIOS executes. BIOS stands for Basic Input Output System. It's a program permenantly stored in the computer on read-only chips. It performs some minimal tests, and then looks for a floppy disk in the first disk drive. If it finds one, it looks for a "boot sector" on that disk, and starts executing code from it, if any. If there is a disk, but no boot sector, the BIOS will print a message like:

```
Non-system disk or disk error
```

Removing the disk and pressing a key will cause the boot process to continue.

If there isn't a floppy disk in the drive, the BIOS looks for a master boot record (MBR) on the hard disk. It will start executing the code found there, which loads the operating system. On GNU/Linux systems, LILO, the LInux LOader, can occupy the MBR, and will load GNU/Linux.

Thus, if you opted to install LILO on your hard drive, you should see the word LILO as your computer starts up. At that point you can press the left `Shift` key to select which operating system to load - press `Tab` to see a list of options. Type in one of those options, and press return. LILO will boot the requested operating system.

If you don't press the `Shift` key, LILO will automatically load the default operating system after about 5 seconds. If you like, you can change what system LILO loads automatically, which systems it knows how to load, and how long it waits before loading one automatically.

If you didn't install LILO on your hard drive, you probably created a *boot disk*. The boot disk will have LILO on it. All you have to do is insert the disk before you turn on your computer and the BIOS will find it before it checks the MBR on the hard drive. To return to a non-Linux OS, take out the boot disk and restart the computer (from Linux, be sure you follow the proper procedure for restarting: see 'Shutting down' on page 12 for details.)

LILO loads the Linux kernel from disk, and then lets the kernel take over. (The kernel is the central program of the operating system, in control of all other programs.) The kernel discards the BIOS and LILO.

On non-Intel platforms, things work a little differently. But once you boot, everything is more or less the same.

Linux looks at the type of hardware it's running on. It wants to know what type of hard disks you have, whether or not you have a bus mouse, whether or not you're on a network, and other bits of trivia like that. Linux can't remember things between boots, so it has to ask these questions each time it starts up. Luckily, it isn't asking *you* these questions—it's asking the hardware! While it boots, the Linux kernel will print messages on the screen describing what it's doing.

The query process can cause problems with your system, but if it was going to, it probably would have when you first installed GNU/Linux. If you're having problems, consult the installation instructions, or ask on a mailing list.

The kernel merely manages other programs, so once it is satisfied everything is okay, it must start another program to do anything useful. The program the kernel starts is called `init`. After the kernel starts `init`, it never starts another program. The kernel becomes a manager and a provider of services.

Once `init` is started, it runs a number of scripts (files containing commands), which prepare the system to be used: they do some routine maintenance and start up a lot of programs which do things like display a login prompt, listen for network connections, and keep a log of the computer's activities.

Appendix D

Miscellaneous

This chapter contains interesting information that didn't fit in the rest of the manual, such as historical notes. It may be moved to another manual in the future, or made into a coherent chapter.

D.1 Unix History

In 1969, Bell Telephone Laboratories (Bell Labs, a division of AT& T) was working with General Electric and Project MAC of MIT to write an operating system called Multics. To make a long story slightly shorter, Bell Labs decided the project wasn't going anywhere and broke out of the group. This left Bell Labs without a good operating system.

Ken Thompson and Dennis Ritchie decided to sketch out an operating system that would meet their needs. Bell Labs had an unused PDP-7 computer that Thompson wanted to put to use, so he implemented the system they had designed on that machine. As a pun on Multics, Brian Kernighan, another Bell Labs researcher, gave the system the name Unix. The group was able to get funding to buy a better computer, a PDP-11, by proposing a plan to write a word processing system. Rather than write the word processor from scratch, they made it an application that ran under Unix, which they ported to the PDP-11.

Later, Dennis Ritchie invented the "C" programming language. In 1973, Unix was rewritten in C instead of the original assembly language.[1] In 1977, Unix was moved to a new machine through a process called *porting* away from the PDP machines it had run on previously. This was aided by the fact Unix was written in C since much of the code could simply be recompiled and didn't have to be rewritten.

In the late 1970's, AT& T was forbidden from competing in the computing industry, so it licensed Unix to various colleges and universities very cheaply. It was slow to catch on outside of academic institutions but was eventually popular with businesses as well. The Unix of today is different from the Unix of 1970. It

[1]"Assembly language" is a very basic computer language that is tied to a particular type of computer. It is usually considered a challenge to program in.

has two major variations: System V, from Unix System Laboratories (USL), a subsidiary of SCO[2], and the Berkeley Software Distribution (BSD). The USL version is now up to its forth release, or SVR4[3], while BSD's latest version is 4.4. However, there are many different versions of Unix besides these two. Most proprietary versions of Unix derive from one of the two groupings. The versions of Unix that are actually used usually incorporate features from both variations.

Current proprietary versions of Unix for Intel PCs cost between $500 and $2000, with the exception of Solaris x86 which has been crushed by free Unix clones and forced to lower prices.

D.2 GNU/Linux History

Debian traces its roots to the founding of the GNU project in 1984 by Richard M. Stallman. GNU (GNU's Not Unix) is a project of the Free Software Foundation; their goal was and is to replace the Unix operating system with free software. They had written almost an entire operating system by the early 1990s, but the kernel was missing. Fortunately, Linux appeared to fill this gap.

The primary author of the Linux kernel is Linus Torvalds. Since his original versions, it has been improved by countless numbers of people around the world. It is a clone, written entirely from scratch, of the Unix operating system. Neither USL, nor the University of California, Berkeley, were involved in writing Linux. One of the more interesting facts about Linux is that development occurs simultaneously around the world. People from Australia to Finland contributed to Linux and will hopefully continue to do so.

Linux began with a project to explore the 386 chip. One of Linus's earlier projects was a program that would switch between printing `AAAA` and `BBBB`. This later evolved into Linux.

Linux has been copyrighted under the terms of the GNU General Public License (GPL). This is a license written by the Free Software Foundation (FSF) that is designed to keep software free. In brief, it says that although you can charge as much as you'd like for a copy, you can't prevent the person you sold it to from selling it, giving it away, or modifying it. It also means that the source code[4] must also be available. This is useful for programmers. Anybody can modify Linux and even distribute their modifications, provided that they keep the code under the same copyright — the GPL.

Debian is called GNU/Linux because it is a product of two massive efforts, the Linux kernel and the GNU project. Still, focusing on only these two contributions leaves out tens of thousands of contributors. It's impossible to keep track of everyone who's made Debian what it is today.

The following two lists, of leaders and release milestones, are copyrighted by Software in the Public Interest and may be redistributed but not modified.

Debian has had several leaders since its beginnings in 1993.

[2]Previously, USL was owned by AT& T and later Novell

[3]A cryptic way of saying "System Five, Release Four".

[4]The *source code* of a program is what the human programmer reads and writes. It is later translated into machine code that the computer interprets.

- Ian Murdock founded Debian in August 1993 and led it until March 1996. This effort was sponsored by the FSF's GNU project for one year (November 1994 to November 1995).
- Bruce Perens led Debian from April 1996 until December 1997
- Ian Jackson led Debian from January 1998 until February 1999.
- Wichert Akkerman is the current Debian leader since February 1999.

Here are a few of the major Debian release milestones:

- 0.01-0.90 were released between August 1993 and December 1993
- 0.91 released January 1994 (around 30 developers, primitive packager)
- 0.93R5 released in March 1995 (dpkg makes its first appearance)
- 0.93R6 released in November 1995 (around 60 developers, a.out, first dselect)
- 1.0 was never released. It later became version 1.1
- 1.1 'Buzz' released June 1996 (474 packages, 2.0 kernel, fully ELF, dpkg)
- 1.2 'Rex' released December 1996 (848 packages, 120 developers)
- 1.3 'Bo' released July 1997 (974 packages, 200 developers)
- 2.0 'Hamm' released July 1998 (1500+ packages, 400+ developers, glibc2)
- 2.1 'Slink' is scheduled for release in early March 1999.

D.3 The Linux kernel's version numbering

The first number in Linux's version number indicates truly huge revisions. These change very slowly: right now version 2 is the latest. The second number indicates less major revisions. Even second numbers signify more stable, dependable versions of Linux while odd numbers are developer's versions that are more prone to bugs. The final version number is the minor release number—every time a new version is released that may just fix small problems or add minor features, that number is increased by one. Right now the stable kernel is 2.0, and the developer's kernel is 2.1. When 2.1 is ready, it will become stable kernel 2.2. The latest version of the stable kernel is currently 2.0.35, though that may well change by the time you read this. The 2.2 stable release is expected soon.