THE advanced linux POCKETBOOK

This Pocketbook is for Advanced users of Linux. If you find the information over your head, we recommend that you look at the 2003 Edition Linux Pocketbook, which can be found on last month's cover CD. If you missed out, then don't worry... we can help you! Call **136 116** or visit **www.magshop.com.au** to order your back issue for $9.80 + $6.50 for postage within Australia.

May the source be with you!

THE 2003 edition linux

POCKETBOOK

Editorial

Hello and welcome to the 2003 Edition of the Advanced Linux Pocketbook. Last month we gave you the updated Linux Pocketbook, so by now you should be an old hand when it comes to the basics; installing software, configuring settings, and connecting to other machines. You should be at least savvy when it comes to the command line, even if you can't do much more than manipulate files. Grep should be in your bag of tricks, and your fingers should gravitate to the 'l' and 's' keys automatically when you sit idle at the console. If you're keen to learn how to make the most out of Linux, you're in the right place. While the first book dealt with how to get Linux up and running on your system, this Advanced Pocketbook develops your skills from there. Remember back to when you installed your first Linux distribution and looked up to the geeks that seemed to know so much more about this obscure yet powerful operating system. They dazzled you with talk of Python scripts, pages built using PHPNuke, and staying up late at night, fuelled by litres of Jolt cola, to recompile their kernels. Well, here's your chance to become one of those geeks. The APC Advanced Linux Pocketbook 2003 edition will raise your level of Linux awaredness to a higher plane. If you're after enlightenment, read on. Yes, that really was a bad Linux pun. As always, may the source be with you.

Matt Overington

First published May 2001.

The Advanced Linux Pocketbook is published by ACP Tech, a division of ACP Publishing Pty Ltd (ACN 053 273 546).

Contents

The next step

The Advanced Linux Pocketbook is, as its title suggests, the follow-up to *The Linux Pocketbook*. In this edition we'll cover a variety of advanced topics, including how to configure new hardware and set up a multitude of network services, an introduction to the power of scripting, some handy optimisation tips and basic good security habits.

Being an advanced book, however, there is a certain amount of knowledge we assume you already have. You should know how to install, configure and tailor Linux to your liking, as well as how to edit configuration files and install new programs — all the information that was featured in the previous Linux Pocketbook. We don't explain how to do simple tasks in this book; rather we aim to build on your current knowledge. This means covering a far broader range of Linux's capabilities and getting right to the core of our subjects.

Moreover, because we don't go over the basic knowledge you learned in *The Linux Pocketbook*, much of which was based around getting Linux up and running, we can dedicate the whole book to helping you get the most out of Linux.

If, however, you consider yourself still fairly new to Linux, just keep *The Linux Pocketbook 2003 edition* handy as a reference guide.

Since *The Advanced Linux Pocketbook* follows on from its predecessor, much of the content in this Pocketbook focuses on the Red Hat and Mandrake distributions. Many commands and programs will operate the same way on other distributions, but you still may find the odd Red Hat and Mandrake-specific quirk here and there. If you're wondering why Mandrake and Red Hat operate in such a similar way remember that Mandrake started out based on Red Hat so they set up the system in a similar way.

A NEW DISTRIBUTION

The print versions of the Linux Pocketbook series provided an introduction to a variety of distributions — multiple versions of Red Hat, Caldera OpenLinux and Mandrake Linux. *The Advanced Linux Pocketbook* introduces you to one more — Debian.

Debian is a very well respected and popular distribution, but it doesn't get quite the same amount of press as distributions like Red Hat or Mandrake. Part of the reason for this is that its insistence on stability and security means it adopts new technologies more slowly than the mainstream distributions, which doesn't endear it to new users who want the latest and greatest. Additionally, Debian currently uses a traditional text mode installation procedure, which again isn't as inviting to new users as the graphical installation mechanisms provided by Red Hat, Mandrake and others.

That said, Debian — like this Pocketbook — assumes you already have a grounding in Linux. It's an advanced, clean and professional distribution for advanced users. If you don't already know

how to edit configuration files or start the X window system from the command line, then you might find yourself quickly lost in Debian. Also, because this *is* the advanced book, the install guide we've provided is brief; we're assuming you are already familiar with the basics of installing a new distribution.

You can pick it up from Debian.org or everythinglinux. If you think you're geek enough to give it a try, we strongly recommend you set up a new partition in which to install Debian. This way you can try it out and see if you like it. Be warned though that if your only experience of Linux is Red Hat or Mandrake, Debian will be a challenge for you — some directory structures differ in Debian, and it has a completely different (but very powerful) package management system. Hence, you can see why we chose it for this Pocketbook — an experienced distribution for experienced users!

If you find it too hard at first, stick with it a while. There's a reason developers of distributions such as Caldera and some embedded Linux derivatives chose Debian as their base: it delivers on what it promises — stability, security and consistency across the distribution and installed programs.

As stated above, however, this book focuses primarily on Red Hat and Mandrake distributions, as these are the distributions we've shown you how to configure and use from the start. All the packages mentioned in this book can be found online easily using Debian's inbuilt package management tool, which can find and fetch programs for you and for which we provide an overview in Chapter 1. And, aside from the location of a configuration file or two (which the Debian package of an application will install to the correct location for you) you will, as an experienced user, be able to apply all of the information contained in this book to Debian.

If Debian doesn't seem your cup of herbal tea right now, keep an eye on it (and all other distributions) and check it out at a later date. As distributions evolve you may find yourself growing fond of a new distribution. Choice is, as always, a wonderful thing!

Introducing Debian

Welcome to Debian GNU/Linux

Linux geeks can get pretty fanatical about their chosen distribution. Some of the most outspoken advocates in this arena are Debian users.

What is it that makes Debian such a much-loved distribution? The short answer is this: Debian handles Linux the right way.

Debian has often been cast as a power-user's distribution and difficult to use. While it doesn't have a drag-and-drool install process, it is by no means hard to use and the small investment in time of learning the Debian Way pays significant dividends later on.

As part of your experience of Linux, and to offer you an experience of Linux you may not have encountered before, we've chosen to include this short guide as part of *The Advanced Linux Pocketbook 2003 edition.*

WHY IS DEBIAN COOL?

The first thing that sets Debian apart from other general-purpose distributions is that it's a project, not a company. This means there is a significant focus on doing things right without the commercial pressures of getting out a release. As a result, significant periods of time have passed between releases while people worked on fixing bugs. Debian is renowned for being behind other mainstream distributions, more often than not when it comes to implementing new Linux technology, such as a new kernel or more advanced FTP server. But the payoff is that every Debian release has been worked over with a fine-tooth comb and is guaranteed to be stable and as bug-free as possible.

When bugs are found, fixes are released within 48 hours and update packages made available online. As a project, Debian relies on hundreds of volunteers to build and test the packages it includes. This means you tend to get someone who actually uses the software, packaging it and adding their own little tweaks to make it work better. Package maintainers are generally very receptive to suggestions and criticisms, taking great pride in their work and spending countless hours making their packages excel. The great variety of packagers in the Debian system means that packages tend to be better configured out of the box, taking into account the average user's requirements and setting sensible defaults. Of course, you can still configure the software any way you like without Debian interfering with your setup.

The other significant cool-factor feature of Debian is an excellent and much-loved piece of updating software called apt. This handy tool makes upgrading and updating a Debian system so easy and well automated that it's the main Debian feature every other distribution is looking to emulate.

STRONG POLICY

The Debian project has a strong policy on licences and file locations. If the policy is violated it is treated as a serious bug, which means an entry is made in the bug database and the author is sent periodic emails requesting that they fix it.

If a piece of software doesn't conform to the Debian Free Software Guidelines, it is placed in the non-free section of the distribution which isn't considered part of the Debian system. Licences that are considered free include the GNU Public License (GPL), the BSD licence and Perl's Artistic License.

An important and often overlooked issue for distributions is the location of files. It may seem like a minor issue but when you have hundreds of pieces of software on a computer, finding the various configuration and documentation files can be troublesome.

Most distributions have some form of policy on file locations. The problem is that many of them rely on 'contributed' packages which are considered outside the distribution and have no well-used, central location for reporting policy violations. Policy violations are also not often considered critical.

[A] significant cool-factor feature of Debian is an excellent and much-loved piece of upgrading software called 'apt'.

In Debian, configuration files are stored in /etc. No ifs, no buts. If you need to configure a package, that's where you look. Documentation is supplied as man or info pages with each package, and additional documentation, such as README files, is in /usr/share/doc/packagename. If you find a package that does not adhere to these and the other standards in the Debian Policy Manual, feel free to report it as a bug. Debian is serious about quality and bug reports are given the highest priority.

DEBIAN IS A LIVE DISTRIBUTION

Debian releases don't happen very often. With over 8,700 separate packages, it takes a lot to marshal them together into a mostly bug-free version ready for release. Debian won't make a release until everyone involved is satisfied that it's ready. That said, few Debian users actually run the released version for longer than a few days.

Debian is, in reality, three distributions rolled into one. These distributions are known as 'stable', 'unstable' and 'testing'. Stable is the current released system, which at the moment is 3.0r0 (codenamed 'Woody'). A stable release receives no updates

apart from security and bug fixes. No new features or functionality will be added to a stable release (which contributes to the distribution being stable!).

Unstable is the current release which is being further developed, with all the latest packages freshly uploaded by the package maintainers — bugs and all. Ironically, Unstable is actually generally quite stable. However, at times when major changes are in progress it can be broken and unusable for days or weeks at time. If you plan to run unstable, you need to keep an eye on the mailing lists and news sources to avoid updating your system while major changes are underway.

Testing is the middle ground between stable and unstable. Packages that have been in unstable for a while without any serious bugs or new versions are eventually pushed into testing. This means that testing packages are the latest usable, though not necessarily bug-free, versions of each package. Testing is probably the place to be while you're getting to know your way around Debian.

So what is the point of the different versions? Debian isn't just a distribution on CD with occasional upgrades. Debian is a live system that is constantly being updated with the latest versions, bug fixes and upgrades. As packages are added to the mirror sites around the world, the central packages list is updated. The packages list contains information about all the packages currently available, including which other packages they depend on. You can upgrade your system using the apt-get command as often as you care to run it, and your entire system will be automatically upgraded, all dependency issues being resolved by apt-get. It's literally a set-and-forget task, and is one of the major attractions of Debian.

If a new stable release of Debian hits the Web, don't download new ISOs or wait for it to appear on a magazine, just run apt-get overnight (if you've got a slow link) and in the morning you'll be running the latest Debian system. Most Debian users run apt-get at least once a week to make sure they're running the latest versions of everything they need. Details of how to drive apt-get and its front-end dselect can be found later in this chapter.

Debian is, in reality, three distributions rolled into one. These are known as 'stable', 'unstable' and 'testing'.

DEBIAN'S PACKAGE MANAGEMENT SYSTEM

Debian packages come in .deb format and are manipulated by a tool called dpkg**.** dpkg **is called by front-ends like** dselect **and** apt-get **when a package needs to be installed. The format handles all the dependencies, installation and removal issues when adding software to your Debian system. Each Debian package contains three main sections:**

- **The executables, documentation and associated files which are installed when you select the package.**
- **A control file which sets up the dependencies, conflicts, suggested additional packages and description of the package.**
- **A series of scripts which are run before and after installation and removal of the package from the system.**

A special type of package file is a Meta Package. These packages contain no files themselves but depend on a variety of other packages, making it easier to install large groups of packages related to a specific program. An example is the Netscape **Meta Package which installs the various Netscape packages to get a working Netscape installation going.**

Another special package is a Virtual Package, which specifies a type of application. Some packages, such as one that requires a working mail transport agent (MTA), will depend on a Virtual Package such as mail-transport-agent**. Any MTA you install will provide this Virtual Package meeting the dependency.**

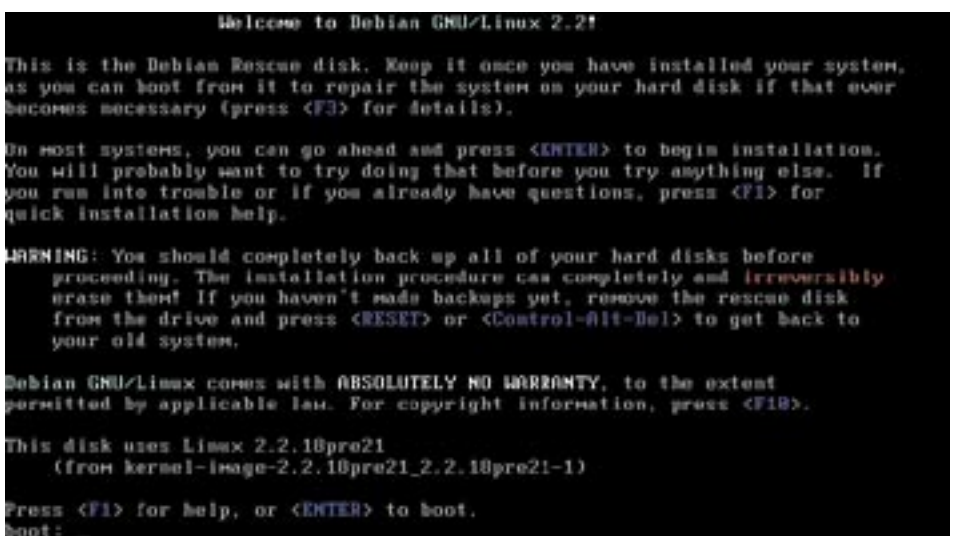
Installing Debian

Debian might not have as slick an install as Red Hat or Mandrake, but its features and performance make it well worth the effort.

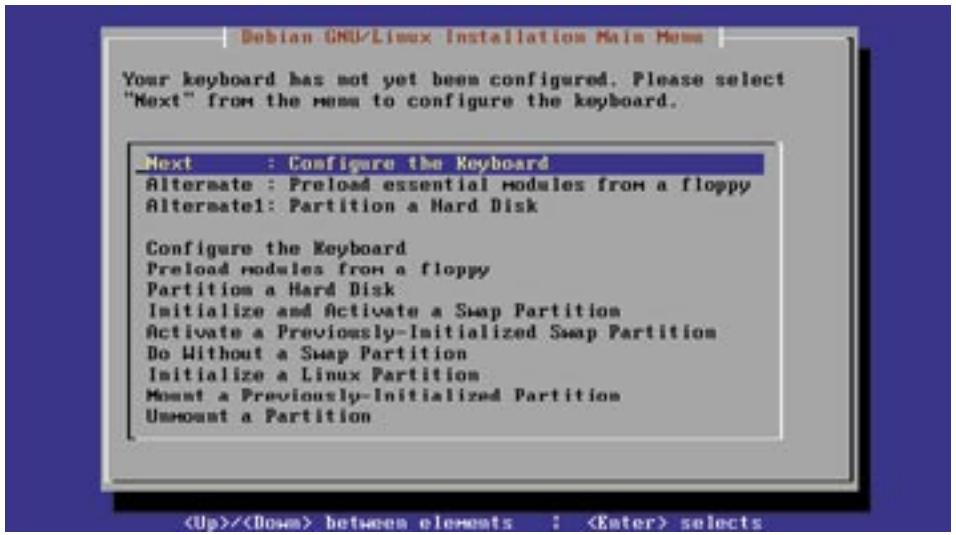
If you have a spare partition to try out Debian, we highly recommend you do so. Be warned that if you're used to Red Hat or Mandrake, Debian won't initially be so easy to install, or even to use. But, as we stated earlier, the learning curve for Debian pays off thanks to greater flexibility and extremely useful tools such as apt. At this stage Debian *is* a distribution for advanced Linux users, so if you think you're up for it, give it a spin to see just how much distributions can differ from each other.

To begin installing Debian, throw disc one into your CD-ROM drive and boot up, assuming you have a bootable CD drive. If you don't, read the documentation on the CD about booting from a floppy. No help on how to do this here — this *is* the *advanced* book.

You'll be presented with a range of options. Unless you have some wacky hardware you should be able to just press Enter to continue.



The text-mode Debian install screen for 'Potato'.



Follow the prompts through the install process, choosing the next option or backtracking to answer previous prompts. At any point you can view the progress of the installation process with Alt-F3, or jump to a console prompt with Alt-F2. Alt-F1 takes you back to the installation procedure.

When you get to the Device Driver Modules section you can choose any kernel modules you might need for the installation and thereafter, such as devices to get your network running, sound, and so on. After this you'll be asked to select the medium to use for the install. Choose 'cdrom' and go for the default settings.

Follow the onscreen directions and reboot when prompted.

Eventually you will get to the point where the installer scans your CD. This means it is grabbing the package list off the CD. There are three Debian CDs in the Woody release, so put the second CD in when asked and get it to scan the packages on the second disc as well.

At this point you can add other sources, like a pre-compiled collection of new updates which contains Debian updates. If you have a permanent Internet connection, you could at this point add a Debian mirror to your sources list. How to do this will be covered later, so for now stick with the CDs and continue installing Debian.

For packages you can choose either the Simple or Advanced installation. The Simple option allows you to choose from a list of install types,

The main Debian installation procedure screen. You can jump to any installation step from here.

UPDATING LISTS

Many parts of the Debian system are designed to have packages plug into them without damaging existing settings. This is one of the major advantages of a strong, enforced policy in the distribution. Some parts of the system use update-something **tools to push any changes you make into use. It's important to use these tools instead of manipulating the files directly or you may end up breaking the way Debian packages work with them. The standard Linux** /etc/modules.conf **should never be directly modified. Instead you can place a file in /etc/modutils with, for example, module loading details for your soundcard. Then all you would need to do is run** update-modules **to update the real** /etc/modules.conf **file, letting Debian manage your system.**

The menus taking the function of the Windows Start Bar in X window managers all tend to use different file formats to add and remove items that appear in them. To handle this, Debian introduced a common file format for all menuing systems and the update-menus **command to update them with translation methods for each menuing system from scripts in** /etc/menu-methods**. These files are stored in /usr/lib/menu. Have a poke around at the other 'update' programs on your system to explore the tasks they perform.**

adding the types of application you think you might need. The Advanced install drops you straight into the package management interface of dselect where you can select each component individually. Either one is quite easy to use, but Simple will be much quicker.

After you have made your selections apt will start installing your packages, telling you how much disk space is required and what packages will be installed and removed. Get used to this as it is your window into the Debian world. Press Enter and put the first CD in to start installing.

Once apt has collected all the packages from the CD it will start the configuration process. Each package is installed in the appropriate place and the installer will ask you questions for packages that require manual configuration. Eventually, after all packages have been installed and configured, you will be dropped into Linux at a login prompt. Login as root and you're ready to enter the exciting world of Debian.

Getting comfy with Debian

Now that you have Debian installed, what should you do next? It's time to update your distribution, get connected to the Web and configure apt-get**.**

INSTALLING UPDATES FROM THE INSTALLATION CD

The most common way that you will be upgrading Debian will be via CD.

To install the updates from the CD, run dselect and choose Access. Select 'cdrom' and enter the location of the CD-ROM drive — for example, /dev/cdrom. dselect will search for the updates on the CD and offer the location of the files for you. Choose the defaults in the questions that follow until you are returned to the dselect main menu.

Next choose Update to retrieve the packages list from the CD. Finally choose Install and go ahead with the upgrade. Once completed, you'll have an up-to-date Debian Woody installation.

After finishing the upgrade from the CD, go back to the Access menu item and choose 'apt' to re-enable updates over the Net (which we'll get onto next).

Now would be a good time to check out the user interface. Because Debian doesn't have a 'standard' the way other distributions do, your default interface will be whichever window manager and desktop environment you chose to install. To start it, simply run startx.

SETTING UP PPP

Before you're able to update your Debian system with the latest changes from mirrors located around the world you'll need to have a working Internet connection. We'll cover getting a PPP connection up and if you're lucky enough to have something faster we suggest you use one of the many excellent HOWTO documents. Telstra cable modem users will find the Debian 'bpalogin' package will handle their needs.

Debian sets up PPP connections using a text-based utility called pppconfig. This tool runs automatically when you install the PPP packages. It is useful to guide you through the PPP configuration options.

Run pppconfig, choose Create and enter the name of your provider. Next you need to choose whether to use static or dynamic DNS. Most ISPs send the name server details each time you connect, so choose Dynamic. Now select the method of authentication for your PPP connection. Almost all modern ISPs will use PAP,

Debian and the WindowMaker desktop.

although you can set up a Chat session which allows you to enter a login and password the old fashioned way.

Enter your username and password and choose your modem port speed. If your modem and serial port can handle it, choose 115200. This is the speed at which your machine talks to the modem, not your modem's speed. With compression your modem may sometimes go faster than its rated speed so it is useful to have some headroom. Lastly, enter the number for your ISP's dialup lines.

pppconfig next tries to find the serial port your modem is installed on. Make sure the modem is switched on and choose Yes. If it works, your modem will be found. If not, you may need to enter it yourself. If you do need to, remember that /dev/ttyS0 is COM1 under Windows, /dev/ttyS1 is COM2, and so on.

When you're ready, choose Finished to write the PPP configuration files. From now on you should be able to connect to the Internet by typing pon [providername] where 'providername' is the name you gave for your ISP. To disconnect use poff [providername]. You can, of course, use graphical PPP login tools, but which one you use depends on which window manager and desktop environment you prefer (Window Maker, Gnome, KDE, and so on). Hence, it's always handy to know how to get a connection up and running irrespective of the desktop environment.

If you have a permanent connection, such as cable, in pppconfig you can choose the Persist option in the Advanced settings and the connection will automatically redial. If you want it to connect automatically when your computer boots into Linux, move the file /etc/ppp/no_ppp_on_boot to /etc/ppp/ppp_on_boot.

APT AND DSELECT

Now it's time to set up apt to update your system automatically from your local Debian mirror. This is not only handy to do now in order to grab the latest updates that will have appeared after the production of this book, but also once configured you can update your entire system at any time with one simple command.

First, look up your local mirror at *www.debian.org/misc/README.mirrors*. If you didn't install a text mode Web browser, type the following to install one:

```
apt-get install links
```

and insert the appropriate disk to install the 'links' text browser so you can view the mirrors list.

For most Australians the closest mirror will be *ftp.au.debian.org* which we'll use in this example. If your machine is on a university network you may find there is a closer mirror on your campus. Some ISPs also run Debian mirrors, so hunt around for the fastest one.

The next step is to tell apt where it can find Debian packages. The next time you run dselect or apt-get you can update the package listing and update any out-of-date packages or add new ones. You can have multiple mirrors listed if you like to ensure you get the latest version of everything.

Add the following two lines to the file /etc/apt/sources.list with your favourite text editor:

```
deb http://ftp.au.debian.org/pub/
  debian stable main contrib non-free
deb http://ftp.au.debian.org/pub/
  debian-non-US stable/non-US main
  contrib non-free
```

These two lines add the local mirror to your sources specifying that you want the 'stable' distribution and the packages in the main, contrib and non-free sections. The second line specifies the non-US packages, generally encryption software and other packages that cannot be exported from the US.

You can leave in the lines for your CD installation, which means that if the package you request is up-to-date on the CDs it will install from there instead of over your Internet connection. If you put the CD lines first it will prefer your CD, saving you download time.

If you need to use a proxy server, add something like the following to /etc/apt/ apt.conf according to your proxy settings:

```
Acquire
{
    http
```

NEED A CONSOLE TEXT EDITOR?

You did install a text editor you can use didn't you? Well if you're not a vi **person you can always just install another editor from the command line. For example, to install Easy Edit just type:**

apt-get install ee

Because apt **already knows about your CD installation, it will prompt you for the appropriate CD and install the editor for you.**

```
    {
    Proxy "http://192.168.3.1:3128/";
    };
};
```

Now run dselect. This is your control centre for adding and removing Debian packages. First choose Update to grab the latest packages list. For a simple upgrade of the existing packages you could now go straight to Install, but we're sure you're itching to get right into package selection.

Go to Select. The first page you see is a help listing, which is quite handy. Go through and read up on all the options. When you've finished reading, press Space to get down to the nitty gritty.

You'll be confronted with quite a massive list of packages: every package in the Debian system discovered when you updated your packages list. You can use the arrow keys, Page-up and Down and the Home and End keys to move around. Each package has a brief blurb at the bottom of the screen.

To the left of the package name are four status flags. The first is the error flag, which will either be a space for no error or **R** for a serious error. Next is the installed state. Space means not installed, **+** means installed and - means not installed but configuration files were left when it was last removed.

The next two columns tell you the actions that will be taken. The final mark determines what is currently selected as the action for this package. * means marked for installation or upgrade. - means marked for removal but configuration files will be kept. **=** means on hold, the package will not be changed — which is handy if you are running the unstable distribution and a package is broken. _ means the package will be purged, removing the files and the configuration files. **n** means the package is new and hasn't been marked for install or removal.

If all this seems a bit confusing, that's because it is. You can press V to display a more user friendly version of the status.

To add a package, select it in the list and press **+**. If the package has any dependencies, a new screen will be displayed with any dependent or suggested package. Check you really want to install those dependent packages and press Enter. Continue through the list until you are happy with your choices and press Enter to finish. If there are any remaining dependencies or suggestions you will be prompted for them.

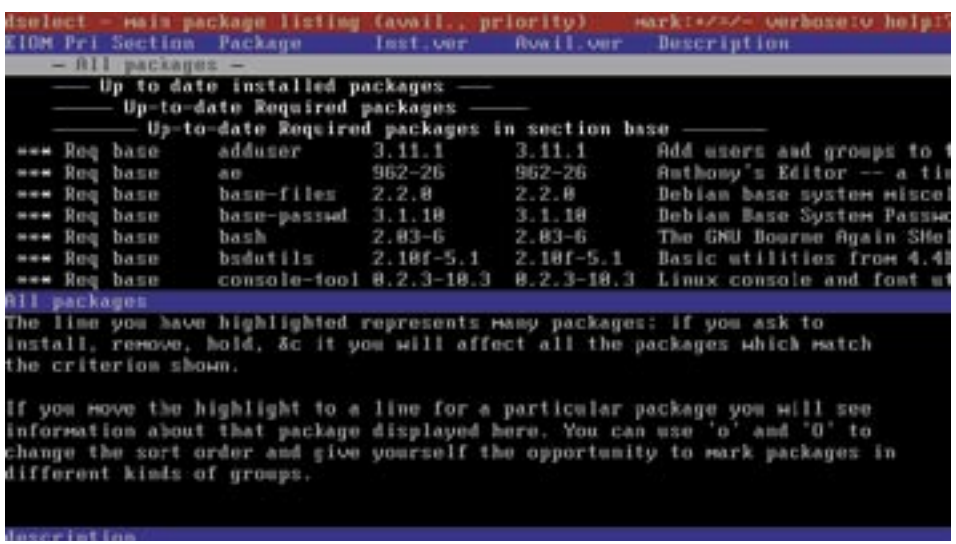
To remove packages you have two options: to remove them completely (purge) or to remove them but leave the configuration files. To purge, press _ and to remove and leave the config files press -.

When you have finished adding and removing packages, press Enter and any other dependencies will be displayed. When the dependencies are all resolved, you will be returned to the main dselect screen. Select Install to begin downloading and installing the packages. When they have finished downloading, dselect will prompt for any packages requiring configuration and install all the packages. Finally, from the main menu choose Config and Remove. Most packages do their configuration and removal in the main Install phase but some require this extra step.

GETTING INVOLVED

Debian has a real sense of community and not a small amount of civic pride in what has been achieved so far. To really get the most out of Debian you need to keep an eye on the various news sources. At the least you need to know what releases are coming up and how they might affect you.

The local mirror of the Debian home page is *www.au.debian.org* and



dselect ***is Debian's all-singing, all-dancing package utility.***

is your core information source for all things Debian. A Slashdot-style news page is maintained at *www.debianplanet.org*, which is useful for many of the issues that arise.

Of course, being a distributed project without a central physical location, all the real work — and discussion — gets knocked out on the many mailing lists associated with the project. If you find Debian growing on you, check out Debian Weekly News at *www.debian.org/News/weekly*.

While Debian is, at first impression, harder to install and configure than the mainstream distributions, its features, dedication to stability and high level of security is sure to endear some of you, just as it has many other Linux users in the past.

If Debian isn't your can of coke, that's well and good — the whole point of having multiple distributions is choice. Explore the different distributions available, and stick with the one you like best.

REINSTATING LILO

You've installed Debian, checked it out, and regardless of whether you're keeping it installed you'd like to be able to boot your main Linux distribution — and it seems to have disappeared from your LILO menu.

Remember when you installed your current distribution and LILO (or GRUB) was installed as a boot manager? Debian has done the same, but it's probably only aware of itself, so your currently installed LILO configuration is only pointing to your new Debian installation. Restoring your old configuration is easy. Follow these steps while in Debian:

- **Create a new directory under /mnt and mount your main Linux distribution partition**
- **Run LILO thus:** lilo -r /mnt/[dir]

LILO will automatically install itself again using the configuration file found in /mnt/[dir]/etc. Of course, your Debian installation isn't listed in your original lilo.conf **file, so if you want to be able to boot your new Debian partition you'll need to edit** lilo.conf **and add a new entry for it. Do this once you've booted your normal distribution and *only* after you've mounted the Debian partition. LILO won't install itself if it can't find the kernel for your Debian entry, so you'll need to provide a full path.**

Of course, if you made a boot disk during your original Linux installation, you can use this to restore your LILO configuration — now you know why you dug up a floppy to make that disk!

Handling hardware

Modularising hardware

It's likely that you know *of* kernel modules, but not what they're all about and how you can use them. In this section we'll help you to master modules.

Historically Linux drivers were created exclusively by open source developers (read: hackers) with very little help from hardware manufacturers. Companies were worried that if they released the information necessary for building drivers to the open source community, competitors would use the information to clone their products, or sue them over patent violations.

This situation has changed significantly in the past year and many companies, such as ATI, Matrox, and Creative, are paying their own staff or external groups to make their products work with Linux. The resulting drivers are generally open source. Things have changed on the Linux side too, with the addition of USB keyboard and mouse support as of 2.2, and full USB and Firewire support in Linux 2.4. As a result, these days it's safe to say that the majority of hardware will indeed work on Linux.

Linux hardware support increases in leaps and bounds every month and new drivers surface all the time.

However, it's still worth doing some research before making purchasing decisions, and there's a stack of sites that are useful for doing this as well as for support information for hardware you already have. Many hardware vendors have information about Linux support listed on their own sites, and many distributions have a Hardware Compatibility List available online. Finally, sites dedicated to information on specific aspects of the OS (such as XFree86 and ALSA) are full of and dedicated to information on the latest developments in various graphics and sound drivers. Keep in mind that even if your research bears no fruit, there's still a good chance your piece of hardware may work either now or in the future — Linux hardware support increases in leaps and bounds every month and new drivers surface all the time.

Of course, if you're buying new hardware, you can save even more time and effort by letting someone else do the dirty work for you. Companies like EverythingLinux (*www.everythinglinux.com.au*) provide entire online stores listing hardware that is known to work under Linux, as well as information about the different drivers available and instructions on how to install them.

Gaming sites such as Linux Games (*www.linuxgames.com*) also provide benchmarks and reviews of many multimedia Linux devices.

A module is a piece of code that is separate from the kernel that can be loaded or unloaded on demand.

HOW LINUX DRIVERS WORK

Most drivers in Linux are implemented as part of the Linux kernel, with the exception of video cards, which require a separate X driver (and may also require a kernel driver). Kernel drivers may be either built into the kernel itself, or exist as a *module*. A module is a piece of code that is separate from the kernel that can be loaded or unloaded on demand. When loaded, the module is part of the running kernel and it can do anything that kernel code can do. If you used *The Linux Pocketbook* to compile your own kernel you would have had the option to build many drivers either as part of the kernel (the 'y' option), as a module (the 'm' option), or not at all.

The default kernels on many modern Linux distributions come with modular drivers for most popular hardware. If you're using the kernel that came with your distribution, there should be very little need to recompile it to get support for some new hardware item. All those geeky messages that fly by when your system starts up — before your services start — show when Linux probes your hardware and loads the appropriate drivers. You can view this information in your own time by typing dmesg at the console, piping it through more if necessary.

Besides loading the necessary modules for your hardware, there may also be some extra steps required — for example, making sure the driver for your new soundcard can be referred to as just 'sound' if you want it to. Or loading a driver other than the one Linux uses by default for your hardware. Either way, Red Hat and Mandrake can use the hardware setup service kudzu to help you do this. When first started, kudzu scans the computer's hardware for any new devices. If it finds one it then sets up any additional configuration files, installs any packages needed to support that hardware, and then loads the relevant module. kudzu generally works well and makes plugging in new network cards (and similar tasks) a breeze. But if you're a Debian user, or you want (or need) to get your hands dirty, we'll show you how to configure hardware manually.

DEALING WITH MODULES

Most Linux drivers come in the form of modules. But why does it matter if a driver is compiled as a module or as part of the kernel? If drivers are built into the kernel then they are assigned in the order that the kernel probes them: things like I/O addresses, IRQ or DMA numbers are fixed by the linkage order.

With modules, you can control the load order and thus the probe sequence. This is especially noticeable to multihomed machines, as well as hardware that is frequently plugged and unplugged from the computer — USB and PC cards, for example. You can also specify options for drivers, such as whether a device will function in full or half-duplex mode, or what specific IRQ or DMA it should use.

As licensing for the Linux kernel dictates that any code added into the kernel must be licensed by the GNU General Public License, closed source drivers (such as nVidia GeForce drivers) are only available as kernel modules.

Modules also allow you to share ports between multiple devices which use different drivers — for example, if you have a parallel port zip drive and a printer plugged into a port, modules allow you to use them both simultaneously.

The kernel should load drivers dynamically for new hardware, but you can tell it to load extra modules.

For these reasons and more, module drivers are generally preferable to kernel-compiled drivers. They are loaded automatically in the background whenever their functionality is requested — using kmod, the kernel module loader — and unloaded when no longer required, to save memory. It's also possible to add and remove them manually using insmod, modprobe and rmmod.

The modules your system has compiled exist in your /lib/modules directory. Here you can see drivers for different filesystems, network cards, USB devices and SCSI devices. When you run make modules && make modules_install while compiling your kernel, your modules are installed to the /lib/modules directory.

One common term you'll hear when referring to drivers is block and character devices. In summary, a character device (like a tape drive, or a serial port) needs data to be written to it in order. A block device (such as a hard disk) doesn't. The difference is between sequential and random access storage.

TELLING LINUX TO LOAD ADDITIONAL MODULES

While the kernel should load drivers dynamically for new hardware, you can manually tell Linux to load additional modules using the /etc/modules file. This is useful in situations where you'd like to load a different driver to one the Linux kernel has loaded. As you'll see later, you can use this feature to make sure IDE CD burners use a driver other than the regular IDE driver.

SPECIFYING OPTIONS FOR PARTICULAR MODULES

You can configure various options for your modules using a handy file called /etc/modules.conf. This file configures every aspect of modules except for which modules to actually load. This includes:

- Options for individual modules. For example, starting a driver to access a piece of hardware with a specific I/O, DMA, and IRQ, or to enable nifty hardware features, such as full duplex on a network card.
- Aliases, which map generic names to specific drivers. When Quake3 wants to load your soundcard, it tells Linux to load sound. How does Linux know that 'sound' means your SoundBlaster Live 128? It looks at the aliases in the /etc/modules.conf file to see which module should be loaded for 'sound'.
- Things that need to be done before (pre) or after (post) a module is loaded. For example, its best to start the pcmcia service before loading the *pcmcia-core* driver.

Your /etc/modules.config file will probably already contain some entries, most of which will have been generated by kudzu when you installed Linux. In addition there could be entries generated from driver install scripts, such as those included with the official nVidia drivers.

INITIALISING, REMOVING AND LISTING MODULES MANUALLY

insmod and modprobe initialise (load) a module of a particular name, providing information on the console about any matching hardware that the driver attached itself to. These commands are useful when the correct driver is compiled and exists on the system, but hasn't yet been run by Linux. If you get a bunch of text telling you that a card was detected and loaded, then you have a working driver. Alternatively, check the bottom of /var/log/messages for the same information.

lsmod lists all currently loaded modules. You can also type: cat /proc/modules for the same information.

rmmod removes a particular module.

modinfo -d [modulename] checks a module file for description information. This is useful for troubleshooting and to check whether a particular module file is in fact the driver you were looking for. Unfortunately, not all modules provide this information. You can also use the '-p' switch to request module parameters.

PLUG AND PLAY

Support for generic plug and play devices is included in the 2.2. and 2.4 kernels. However, users of kernel 2.2 should turn **off** the setting in their BIOS called 'OS is PnP aware' to allow the BIOS to configure PnP devices itself. Kernel 2.4 users should leave this feature turned on. **This is important!** Linux 2.2 needs this information to configure most hardware, while

HANDY HARDWARE SITES

Linux Hardware Database
lhd.datapower.com

Linux Hardware Net
www.linuxhardware.net

Linux.com
www.linux.com

Graphics cards

Everyone loves graphics cards — they provide the wonderful visuals that drive our 2D desktops and 3D gaming. So how do you use the latest cards in Linux?

3D ACCELERATION

As mentioned in the previous Linux Pocketbook, 3D support for Linux has increased dramatically. One of the biggest changes was the release of XFree86 4.0, which contains many advancements, including a new modular architecture, support for closed source drivers, and the Direct Rendering Infrastructure. DRI — which allows certain applications direct access to the graphics card bypassing the X protocol — allows for much faster performance in graphics-intense applications, especially 3D ones such as games.

The new Xrender system provides transparency and smooth text on the Linux desktop.

This comes at the expense of losing the ability to run these applications remotely using standard X protocol. However, another technology known as GLX (a better, remote graphics protocol designed for 3D applications) will become part of future DRI implementations and will provide excellent 3D application serving across networks at speed.

GLX was invented by the 3D graphics ninjas at SGI. Another notable contribution from SGI is OpenGL, the basis for 3D graphics on most computer platforms including Linux, Windows, MacOS and Playstation2.

There have been other advancements in X besides DRI and GLX. One of the most exciting is Xrender. Xrender provides font anti-aliasing, which allows text on screen to be smoothed out. This has been available on Windows and MacOS for some time, but Xrender also has other, less common features — most notably varying levels of transparency for every window on the desktop.

Xrender is included in Xfree86 4.0 onwards, but needs support from the two main Linux application toolkits, QT and GTK. The latest versions of QT support the extension, whereas GTK applications won't be accelerated until GTK 2.0 comes out late this year. Xrender will also be able to take advantage of some 3D accelerated video cards.

Besides the usual resources (your video card supplier's Web site, and your distribution's hardware compatibility list), XFree86's own Web site and especially its driver status documents provide a brilliant resource to help you configure your existing card or guide your next video card purchase. Check them out at *www.xfree86.org*.

There have been other advancements in X besides DRI and GLX. One of the most exciting is Xrender.

ATI

ATI is currently the 3D card of choice for Linux. Its Radeon cards provide comparable frame rates to GeForce2 cards at higher resolutions, an excellent set of open source drivers that integrate well with the Direct Rendering Infrastructure of XFree86 4.0, and support for nifty things like XVideo (see the DVD section later in this chapter).

The older Mach64, Rage and Rage 128 cards also have excellent XFree86 4.0 drivers and have been known to run faster than their Windows counterparts. This includes the Rage 128 Mobility which is used on many laptops.

Radeons use the *radeon* driver, Rage 128s use the *r128* driver and other cards mentioned above use the *ati* driver. Owners of all these cards should use the latest release of XFree86 — especially Radeon users and those with dual-head cards. Users who own All-in-Wonder cards with TV input or DVD hardware can use the GATOS project to provide support for these features.

Owners of older ATI cards, namely the Rage 3D and Rage Fury, can get accelerated 3D using the 3.3 release of XFree86 and drivers provided by the Utah GLX project.

3DFX

Now that all of 3dfx's bases belong to nVidia, all future support for its cards will come from the new parent company. XFree86 drivers for all of 3dfx's cards remain in existence, and nVidia has promised to maintain the Voodoo 3, 4 and 5 drivers. These are provided by a package typically called 'glide_V3'. This offers a way to convert standard OpenGL to the Glide language and is used internally on all Voodoo cards.

In terms of XFree86 releases, you'll probably want to use at least version 4.02, as this improves the speed of certain mission-critical graphics applications (Unreal Tournament, for example).

As mentioned in the last Linux Pocketbook (and unfortunately unchanged at the time of writing) current Voodoo 4 and 5 drivers only support one of the two GPUs on board the card, thus lowering performance

For 2D graphics, nVidia cards work straight out of the box on most modern distributions.

significantly on these cards. However, 3dfx's new drivers will hopefully be released in the future by nVidia.

INTEL 810, 815

If you've got one of Intel's embedded 815e graphics cards, support is provided by the *i810* driver in XFree86 4.x. The driver requires the use of *agpgart*, the extended AGP memory driver, which should already be part of your distribution — check for its existence under /lib/modules.

NVIDIA

For 2D graphics, nVidia cards work straight out of the box on most modern distributions. Drivers for GeForce, Quadro and TNT/TNT2 are included in the open source *nv* driver that comes as part of XFree86 4.01 and GeForce2 drivers are included in the 4.02 version of the same driver. GeForce 4 support is found in version 4.2 and later.

3D requires some extra steps, as it uses a closed source driver from nVidia (essentially, a Linux version of the current Detonator drivers for Windows). The driver requires XFree86 4.01 or later, but uses its own form of direct rendering rather than the XFree864 DRI system. It also requires kernel 2.2.12 or later, which shouldn't be a problem.

OpenGL is used in Quake 3 to draw the game's beautiful graphics.

The drivers consist of two main parts: a binary kernel module, which is packaged as 'NVIDIA_kernel', and an X driver packaged as 'NVIDIA_GLX'.

NVIDIA_kernel RPMs are available for various Linux distributions, and if yours isn't listed on the nVidia site, you can easily build your own by running rpm --rebuild on the Source RPM, or compiling the source code tarball. The install scripts will automatically load the kernel module and update your system so it's loaded at boot time.

To use the nVidia driver over the XFree86 driver you'll need to edit your /etc/X11/XF86Config-4 file and update it according to the documentation supplied.

NVIDIA_GLX is an OpenGL driver which replaces some files from the Mesa OpenGL implementation used on Linux — so you may need to force install the packages using '--force' for RPM-based distributions. It's important to read the documents, as you may have to manually remove some Mesa files to stop them interfering with nVidia's OpenGL files.

NVIDIA_kernel RPMs are available for various distributions, and if yours isn't listed on the site you can build your own.

GRAPHICS AND GAMING LINKS

ATI Linux drivers
support.ati.com/faq/linux.html

GATOS project
www.linuxvideo.org/gatos

nVidia Linux drivers
www.nvidia.com/Products/Drivers.nsf/Linux.html

Matrox drivers
www.matrox.com/mga/support/drivers/latest

3dfx
www.3dfx.com

Evil Smoking Dude's Guide to 3DFXs under Red Hat 7
www.evil3d.net/articles/linux/howto/3dfx/redhat7

XFree86 Home Page
www.xfree86.org

Utah-GLX
utah-glx.sourceforge.net

Xrender preview
www.xfree86.org/~keithp/render

TransGaming
www.transgaming.com

The Windows game **Sacrifice** *runs under Linux using TransGaming's DirectX-enabled Wine.*

MATROX

Old and new Matrox cards are both very well supported, with accelerated support for the MGA2064W (Millennium I), MGA1064SG (Mystique), MGA2164W (Millennium II, PCI and AGP), G100, G200, G400 and G800 provided by the *mga* driver. Dual head support also exists within XFree86 for cards that support it, and so does TV out, via a closed source driver (due to Matrox support for Macrovision, which is designed to stop consumers recording DVDs onto video cassettes).

OTHER CARDS

Number Nine and S3 Savage cards are now supported by Xfree86 4.0, using the *i128* and *savage* drivers.

DIRECTX GAMES AND WINE

Since the last Linux Pocketbook, Wine has also had a major update. Newcomer TransGaming has developed a version of Wine which is intended to have full DirectX capabilities. TransGaming provides its customised version of Wine for free, but allows users with a paid subscription to suggest gaming titles they would like TransGaming to suppport.

Network cards

In case 'kudzu' doesn't manage to detect and properly configure any new network cards you install, here's how to configure them manually.

Linux is a network-based OS — it needs networking to function properly, and industrial-level firewalls, proxy servers and Internet connection sharing software is ready to go out of the box in most distributions. Linux supports nearly all off-the-shelf network cards, as well as less common devices, such as wireless LAN cards.

Perhaps the most important thing to remember is that the name on the card isn't as important as the chipset that is used. Linux drivers are based on chipsets — for example, a NetGear network card might use an Intel EtherExpress chipset, and thus use the *eexpress* driver under Linux.

Finding out which chipset your card uses is very easy — the documentation for the card should specify this information. Otherwise, simply look at the network card itself.

Somewhere (usually on one of the larger chips) you'll find the name of the actual chipset — which will more than likely match one of the network drivers on your system. The drivers themselves live in your /lib/modules/[kernel version]/net directory.

Common cards and chipsets include: Compaq Tulip (which uses the *tulip* driver), 3Com (which uses various *3c* drivers), and Intel EtherExpress chips (which use *eexpress* and *eepro* drivers).

Most cheaper generic network cards use RealTek 8139 chipsets and the *rtl8139* driver that goes with it.

> It's worth remembering that the name on the card isn't as important as the chipset that is used.

If none of the above work, try the *ne2000* driver; it will work with most cards.

Nearly all network cards are detected by the Linux kernel upon bootup, and tools such as kudzu also automatically handle the task of setting up the aliases (for example, mapping 'eth0' to your first network card) which we discussed at the start of this chapter.

Debian users or those who like to do things by hand will have to add the appropriate line to modules.conf, in the format:

alias [interface name] [driver]

For example:

alias eth0 ne2k-pci

for the first network card,

alias eth1 rtl8139

for the next, and so on. If there are two or more cards of the same type installed in the machine (for example, if the second line reads alias eth1 rtl8139) Linux will assign one alias per piece of hardware, in the order they were detected at boot time.

Linux calls network devices (such as network cards and PPP connections) 'interfaces'. You may have noticed terms like 'eth0' and 'ppp0' in your travels; these refer to the first Ethernet card and first PPP connection. PPP runs using another device, such as a serial port (for a modem) or a network card (for some ADSL connections).

To see currently enabled network interfaces use the command ifconfig as root.

DIGITAL SUBSCRIBER LINE

The earlier Linux Pocketbooks DSL service has become available from a number of providers.

Most business-class providers offer their own instructions for connection (including Linux-based ones), but the process itself is fairly basic since most DSL providers use one of two common connection methods.

The first is plain DHCP, used on corporate networks and with cable ISPs. If this is the case, you can simply set up your Internet connection using the DHCP option in Linuxconf, DrakConf, or any similar program (or by hand, if you're a real Linux geek).

The second is via PPP over Ethernet (or PPPoE), which uses the PPP protocol commonly used on dialup modems to authenticate users. For this you'll need the 'rp-pppoe' package (surf to *www.roaringpenguin.com/pppoe* to grab the latest version, and be sure to read the requirements — you may need an updated pppd **if you're using a 2.4 kernel). Simply install the package and then run** adsl-setup **in a command terminal to go through a question-and-answer style setup program. It's all common sense but keep in mind when you are prompted about basic firewalling options that we provide information on setting up a secure firewall in the Networking chapter.**

Digital versatile disc

DVDs under Linux? With the help of DeCSS and MPEG decompressors, you can now play the DVDs you bought on the DVD drive you bought.

Chances are you'll already have most of what it takes to support DVDs under Linux. Unlike CD-ROMs, which use the ISO9660 filesystem (or a variant of it such as Microsoft's Joliet for longer filenames), most DVDs use their own filesystem called MicroUDF (though some DVDs are formatted wth ISO9660). They also handle I/O differently than CD-ROMS. Support for both of these features requires kernel 2.2.16 or later, compiled with MicroUDF and DVD ioctl support.

Users of Mandrake 7.2 and Red Hat 7.0 (and above) have such support already built into their distributions, whereas Debian 2.2 users may not. If you're unsure, check for the existence of the MicroUDF driver (*udf.o*) in /lib/modules/[kernel version]/fs/fs. If it exists, you're ready to go. For DVD-ROM support, this is basically all you need. To mount a DVD, simply run:

mount -t udf /dev/cdrom /mnt/cdrom

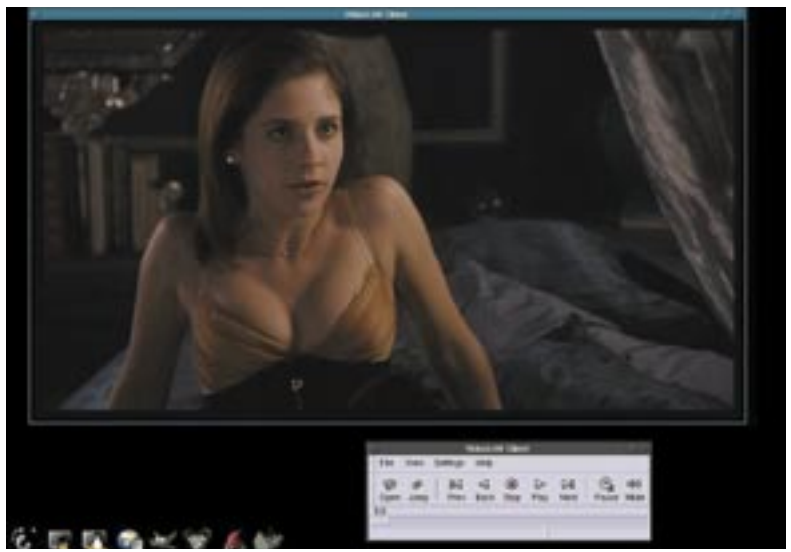
— assuming your DVD-ROM drive is your only DVD/CD drive in the system. If you have both a CD-ROM and a DVD-ROM, you'll need to specify a device such as '/dev/hdc' (for the first IDE device on the second IDE port) or '/dev/scd1' (for the second CD-ROM like device).

Many DVDs can also be mounted successfully with the ISO9660 driver, even if they are formatted in UDF. You might wish to try this yourself — that way, you won't have to replace iso9660 with udf in your filesystem table, /etc/fstab.

DVD MOVIES

One feature which is useful, but not mandatory, for DVD movie playback is Xvideo (or xv) support in your XFree86 driver. DVD movies store information about colour in a different format (YUV) than your monitor does (RGB). To play a movie, your system needs to convert between the two. It will be faster if your video card supports xv, a method of allowing applications to write directly to an overlay (such as an area onscreen where a movie is playing) with the colour conversion performed in hardware.

You can check the /var/log/XFree86.0.log file for a mention of the word xvideo to see if your system is



VideoLAN DVD playback on Linux. Oh, and Buffy again.

using it (try cat /var/log/XFree86.0.log |grep xvideo). Only some video cards are capable of this feature — check the XFree86 Web site's driver status documents to find out if your card supports it.

Most players require /dev/dvd to point to your DVD drive. Create a symlink from /dev/dvd to the location of your DVD drive (probably the same location that /dev/cdrom points to):

```
ln -s /dev/dvd /dev/[drive]
```

Most new PCs are fast enough to play a DVD without specialist decoding hardware, and it seems this trend will continue in the future. But if you do have DVD decoding hardware, then Linux support is available to varying degrees depending on the hardware. Creative DXR2 and DXR3, Hollywood Plus, various ATI cards, Matrox G200 and G400 cards are all supported, and nVidia and 3dfx decoder support is on the way. But since every different hardware DVD decoder works differently, support is moving more slowly than software decoding. The most popular hardware decoders currently work with the open source OMS player.

At the moment there are around five software decoders (players) available for Linux. Two are closed source Linux ports of the excellent Windows-based PowerDVD and WinDVD (the Linux version of WinDVD is LinDVD). Though only available to embedded manufacturers at the time of writing, both companies are planning consumer releases around the time you read this. Since both companies are experienced with writing software decoders, the requirements for the players are much lower than the open source versions, allowing 333MHz Celerons or greater to play back movies full screen.

The open source players are currently in beta, but are working better each day. A 450MHz PII with 128M of RAM is the recommended minimum for full-screen software decoding, though these requirements will drop over time as the code improves. As well as the aforementioned OMS (which also features software decoding), there are the software-only Xine and VideoLAN projects. Packages for these are included on cover disc one. Xine uses much the same backend as OMS, but is designed to be easier to configure. VideoLAN is a completely separate project designed for sending streaming video across a LAN, including DVD (though it can also play local files, of course).

Closed applications currently lead in terms of features (such as interactive support, subtitles, slow motion and skins). However, open source players have some unique features of their own. To prevent unauthorised players and enforce market control, the DVD Content Control Association (a consortium of film studio representatives) ensures

The Xine DVD player.

DVD LINKS

LinDVD
www.intervideo.com

PowerDVD
www.gocyberlink.com

Open Media System
www.linuxvideo.org

VideoLAN
www.videolan.org

Xine
US site: ***xine.sourceforge.net***
Non US site: ***xine.cjb.net***

most DVDs encrypt their content. Players (which decode the content to play it) require decryption keys, and must conform to certain rules to obtain a key. They won't play overseas content, ensuring movie enthusiasts pay inflated local prices for DVDs released far later than they are overseas. Neither do they allow viewers to press the fast forward button during copyright messages and DVD Consortium advertising.

It is already annoying to be forced to sit through the same copyright messages in English, Malyasian, Spanish, Portuguese and every other language within Australia's region, and will be more so in the future when the inevitable DVD version of the long-winded VHS anti-piracy messages appears. Since the open source players acquired their keys through simple reverse engineering, they are not subject to these agreements and do not implement these rules, so viewers can actually watch a movie they own in the way they want to watch it.

CD burning

CD burning is fun for lots of reasons — it allows you to burn your own music CDs, create portable MP3 collections and back up your system's software, to name just a few.

Everyone has a burner these days, so how do you use it under Linux? Whether you go with SCSI or the cheaper IDE option, it's almost guaranteed your CD burner is Linux compatible. Burning under Linux is extremely easy to set up, especially if you have a SCSI device — skip the next section on IDE burners if you do.

EXTRA STEPS FOR IDE BURNERS

IDE CD burners use ATAPI, a SCSI-like command set which works over a regular IDE bus. Thus it uses a different driver (*ide-scsi*) from the regular Linux *ide* driver that is typically used for IDE hard disks and CD ROMs. When Linux boots it will automatically load the regular *ide* driver for all existing IDE devices — so you'll need to tell it to use the *ide-scsi* driver for your CD burner.

The first step is to tell the Linux kernel that your CD burner uses the *ide-scsi* driver. You can make this permanent by adding the following command to your boot 'string':

For LILO:
Edit /etc/lilo.conf
Add the line:
append=" hdx=ide-scsi"
(Yes, that's a space before the 'h')

For Grub:
Edit /boot/grub/menu.lst
Add the line:
hdx=ide-scsi

where 'x' is your IDE CD burner. Then run lilo or grub to initialise the changes.

Next, you need to set up an alias in modules.conf to allow ide-scsi to be referred to as your SCSI adapter. By default, Red Hat and Mandrake will use kudzu to create the necessary entries the next time you boot, but if you want to do it manually add:

```
alias scsi-hostadapter ide-scsi
```

Then reboot. If this sounds unnecessary, it is — you could run some commands to start the drivers for the burner right away. But you'll probably want to make sure the changes are permanent and that the burner works automatically as soon as the machine is booted, so now is a good time to test it.

GETTING READY TO BURN

Make sure that the 'mkisofs', 'cdrecord' and 'cdda2wav' (or 'cdparanoia') packages are installed on your system. The first, mkisofs, is used to make ISO images of data or music, ready for burning to a CD. The second is the actual recording program that burns data to a CD. The last is useful when you want to rip music from a CD and store it on your hard drive (for later listening or for burning to CD).

When you're ready, run the following:

```
cdrecord --scanbus
```

to check Linux can see your burner. You should get a listing of available SCSI (or IDE-SCSI) devices on your system, including your CD-R. If this is the case, your CD-R is set up and all you need to do is pick some recording software.

RECORDING SOFTWARE

Time to choose your burning software. If you want to do it the geek way you can create ISO images of, for example, a data hierarchy as follows:

```
mkisofs -A '[disc name]' -J -L -no-bak -r -o [ouput file.iso] [source dir]
```

X-CD-Roast makes creating data and audio CDs an easy process.

INSTALLING A NEW BURNER

Installing and using a new CD burner is easy. Besides all the steps here, you'll need to make a mountpoint to read CDs in your new drive. (Many, but not all, distributions are smart enough to make one for you.) Make a new directory under /mnt (for example, /mnt/burner) and add a line in /etc/fstab **to complete the installation of your new drive. Both SCSI or IDE-SCSI CD burner devices will be referred to as** /dev/scd0**,** dev/scd1**, and so on. The** /etc/fstab **line should look like this:**

```
/dev/scd0 /mnt/burner iso9660 defaults 0 0
```

to create a Joliet (reads perfectly under Windows machines) and Rock Ridge (reads perfectly under Linux systems) ISO image. To see what all the options mean, type man mkisofs. And, to burn the ISO image, use:

```
cdrecord dev=[device] speed=[speed]
-eject [ISO file name]
```

Simple!

Of course, the many GUI front-ends available don't just make setting up to burn a simple affair, they can also help automate complex processes — such as ripping and then burning music. Most Linux distributions come with a variety of these applications — popular names include gcombust, xcdroast, gnometoaster and kisocd.

Soundcards

After graphics, the next component we adore the most in our beloved PCs is the soundcard, and all the musical joy it brings. That, and the battle cries from a fast-paced FPS game!

As mentioned in the previous Linux Pocketbook, sound on Linux is currently undergoing a major change, from the older and more popular Open Sound System (used by Red Hat and Debian), to the new Advanced Linux Sound Architecture (used by Mandrake), which is also backward compatible. ALSA uses more of the features of modern soundcards, such as multidirectional audio, multiple DSPs and hardware mixing, as well as providing handy features like diagnostic files in /proc.

In future versions of the Linux kernel, ALSA will be standard. Until then, it's available out of the box in Linux Mandrake, or can be compiled into the kernels of Red Hat and Debian.

Compiling sound support into the kernel.

SOUNDCARD SUPPORT SITES

Creative Labs
opensource.creative.com

4Front Technologies
www.opensound.com

ALSA project
www.alsa-project.org

Open Sound System
www.oss.org

There's a wide variety of soundcards available with a staggering range of features. Tools like Red Hat's setup and Mandrake's Hardware Configuration do an excellent job of automatically setting up these options for you.

If you'd like to do things yourself, there are a fair number of options to configure that will depend on the specifics of your card — it's best to check online for the various details.

Like network cards, Linux support for soundcards is based on their chipset rather than their model. For example, the SoundBlaster Live cards use an EMU 10K chipset, and are thus supported under the *emu10k* driver. The Creative Vibra128 uses the Ensoniq 1371 chipset, and thus uses the *ens1371* driver. Creative commands around 80% of the soundcard market and produces its own open source drivers. Linux does support other manufacturers such as ESS and Yamaha, but Creative products definitely work well. Many sites provide information on the chipset your card uses, whether to use *ALSA* or *OSS* drivers, and the options necessary to make the most of your card's features — usually set up in modules.conf. Check the box above for sites that provide this information.

Basic (non card-specific) sound support is provided by a driver called *soundcore*. Many applications that require sound on Linux look for another driver called *sound*, which is typically an alias to the actual driver configured in /etc/modules.conf (there is a non-aliased driver called *sound*, but the alias overrides it). *soundcore* needs to load before the soundcard alias driver does. Hence, for the Soundblaster Live card we mentioned earlier, modules.conf would contain:

```
alias sound emu10k1
pre-install emu10k1 insmod soundcore
post-remove emu10k1 rmmod soundcore
```

Again, the options and modules vary for each card vendor, so check online for card-specific details.

CPUs and SMP

It is often said that two heads are better than one. So what are you waiting for? Build your own symmetric multiprocessing system (SMP) and start reaping the benefits.

LIES, DAMN LIES, AND SCALABILITY

Scalability is one of the biggest buzzwords in today's IT industry. The term describes the ability of a system to expand to handle larger demands. In terms of operating systems, this usually means the ability to run on more powerful hardware as demands on performance increase.

One of the biggest users of the term is Microsoft, which claims triumphantly that the latest versions of Windows are truly scalable, and thus ready to compete with the proprietary Unix OSes that run large-scale business and government departments — that is, 'enterprise ready'. Oddly enough, Linux is more scalable than any version of Windows, because unlike Windows, which exists only on Intel-compatible architectures for server and desktop versions, Linux is portable to high-end computing architectures such as PowerPC, Compaq Alpha, AS/400 and UltraSparc. While most of these systems are proprietary in nature and more expensive than generic Intel hardware, they are capable of handling loads and tasks undreamt of on their cheaper competitors.

Under Microsoft's limited definition, scalability means the ability to run multi-CPU Intel machines. When the Linux kernel 2.2 was the newest kid on the block, Microsoft was right — kernel 2.2's multiprocessing support was quite immature and poor compared to Windows. Beyond four processors, the benefit of any additional processors dropped off considerably, as each new processor gave less and less return.

That was 2 years ago. Now kernel 2.4 has been released, and in the words of Dave Miller, chief Linux SMP guru: "I'd like to thank Microsoft, without which this would have never been possible". Partially as a result of Microsoft's benchmarks, Linux 2.4's SMP is top notch, especially on the IA32 systems that dominate Microsoft's limited scalability views. Linux now holds the record as the fastest Web server on two, four and eight-processor IA32 machines on SpecWeb — the benchmark Microsoft has used for the last five years proclaiming the brilliant performance of IIS. Performance under the 2.4 kernel also scales linearly to 16 processors.

PENTIUM 4 AND THE I1586 IDENTIFIER

Operating systems identify a particular processor using a function called CPUID. According to CPUID, an Intel 386 is an i386, a 486 an i486, a Pentium i586, Pentium II i686, and a Pentium III an i786. As a combined marketing tactic and hilarious joke, Intel decided to return 'i1586' (Roman numerals, geddit?) for its Pentium 4 chips. As a result, older Linux kernels won't boot properly on Pentium 4 systems, because they don't recognise the CPU. Of course, it didn't take long for this little change to make it to the kernel, and 2.2 users can upgrade to 2.2.18 or later in order to boot on Pentium 4 systems. Kernel 2.4 will run just fine.

FUNKY THREADS

We took a look at processes with the ps command in *The Linux Pocketbook 2003 edition*. Although your processors can only execute one instruction at a time, your system is capable of doing multiple tasks simultaneously. The kernel achieves this by switching between processes extremely quickly. This kernel task is called scheduling.

Processes are often made up of more than one thread. Threads are the smallest unit of execution and form the basis of multitasking (running more than one program simultaneously) on most modern operating systems.

Threads are often called lightweight processes'; there's more to them than that, but unless you're about to sign up to the the Linux kernel mailing list, suffice it to say that using the threads concept reduces overhead by sharing fundamental parts between the threads, allowing scheduling to occur more frequently and efficiently.

Operating systems can implement threads in two ways: at user level (where most ordinary programs run) and kernel level (where many device drivers run). User level threads manage threading tables themselves; this is called cooperative multitasking. Each thread gives up the CPU by calling an explicit switch, thereby allowing another thread to use the CPU for a while.

> Linux now holds the record as the fastest Web server on two, four and eight-processor IA32 machines.

User space threads are not distributed among processors. Hence, if an an out-of-control thread decides it doesn't feel like calling the switch, it can use all the CPU all the time (or monopolise the timeslice). This usually results in a crash. Hence cooperative multitasking (and user space threads) essentially . . . well, suck. Older versions of MacOS (OS9 and earlier) and Windows for Workgroups used this approach.

Kernel level threads are much nicer. The Linux kernel, the Unix-like kernels used in closed source Unix, BSD and MacOS X and even that funny kernel32.dll thing used in another OS, all use a program called

a scheduler to determine which processes get to use the CPU at any one time. The scheduler uses your CPU's own clocks, and because this is at the kernel level the scheduler can override what the thread itself wants to do. So even if a thread wanted to, it couldn't monopolise the timeslice and starve other applications of CPU time. This is called true multitasking, and it's been on Unix via the POSIX threads implementation for many, many years.

In any SMP system, granularity occurs at the thread level. In English, this means a single thread won't run faster on a dual CPU system, but two instances of that thread can be run simultaneously. Thus, applications must be designed with this in mind, and use multiple processes wherever possible.

WHAT YOU'LL NEED

While Linux SMP also works with IA64, Sledgehammer, Sparc, PowerPC, Compaq Alpha and other high-end architectures common in government and large business installations, we'll be focusing on garden variety i386 multiprocessing here.

Many vendors (such as Dell and IBM) ship ready-to-go Linux-based SMP machines; but if you're shopping for components yourself, you'll need an SMP motherboard. This will typically support a 'base two' number (1, 2, 4, 8, 16, and so on) of Intel-based chips, which must be identical in both model and stepping (which refers to the specific chip revision).

Speed is less of a concern, though you will have to underclock a faster chip to use it with a slower one. If you'd like to find these details for your current CPU, take a look at /proc/cpuinfo. If you're buying a new machine, it may be wise not to use every CPU slot on your motherboard. This allows you to leave the purchase of additional CPUs until later when the cost will have dropped significantly.

Intel claims ownership of the common APIC SMP scheme, and no other company may use it unless licensed from Intel. While AMD supports the OpenPIC SMP standard, no motherboards existed at the time of writing.

Now you've got the hardware, you'll also need an SMP-enabled kernel.

BUILDING A KERNEL FOR SYMMETRIC MULTIPROCESSING SUPPORT

Modern Linux distributions come with a variety of kernels, and will install prebuilt SMP kernels on SMP-based systems. If you decide to build your own kernel, you'll need to make sure the following options are checked:

- Symmetric Multiprocessing (SMP) support to support multiple CPU systems.
- MTRR (Memory Type Range Register) support to solve problems with buggy BIOSes which only activate cache memory for the first processor. Incidentally, this is also useful for non-SMP systems as it can double the performance of some video transfers.
- Enhanced RTC (real-time clock) support — which the scheduler needs.

Linux distributions come with a variety of kernels, and will install prebuilt SMP kernels on SMP-based systems.

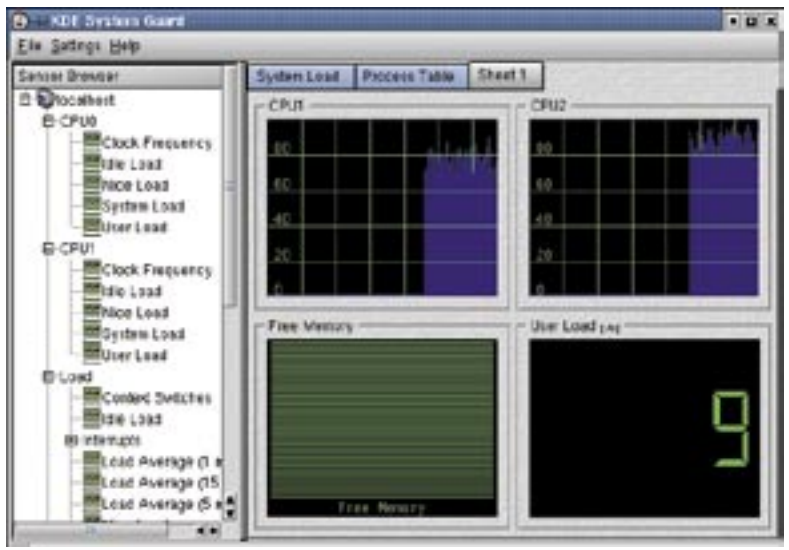
Make sure Advanced Power Management is disabled. This isn't because your multiprocessor beast should never be starved of power, but because the current implementation of APM under Linux is not compatible with SMP.

Also, remember to rebuild not only your kernel but any modules as well — uniprocessor modules won't load on SMP systems. Uninstalling and recompiling LILO with LARGE_EDBA support may also be necessary in some circumstances. Just remember to keep a copy of your older kernel around in your LILO menu in case anything goes wrong.

When your spanking new SMP kernel boots, you can tell if you're firing on both barrels by taking a look at /proc/cpuinfo.

Besides the improved general SMP performance we mentioned earlier, Linux kernel 2.4 has another important SMP feature. All major subsystems (networking, VFS, VM,

Catting /proc/cpuinfo ***to see both processors functioning.***



ksysguard ***monitors CPU loads and process distribution.***

I/O, block/page caches, scheduling, interrupts, signals, and more) are fully threaded, and can thus take better advantage of your SMP system.

KEEPING EVERYTHING NICE

In the 2.2 and 2.4 kernel series, each process has a priority which determines how much of the timeslice it can use. If one process has higher priority than another, it gets to use more of the CPU. The default priority of a process is 0, the highest possible being 20 and the lowest -19.

You can change the priority of programs using nice. On its own it just prints the current default priority for new programs. The command nice -n [adjustment value] [command] runs a new program with a specific priority, for example: nice -n 10 service httpd start will start Apache with a priority of 10.

Be careful when re-nicing programs like this. If you give too much priority to programs, they might hog the system. Likewise, reducing the priority of certain critical tasks will, obviously, be detrimental to your system.

MONITORING CPU USAGE

Since most SMP systems are used in business and research environments, it's important to keep track of your system's performance so that you can address bottlenecks and optimise your machines.

Many modern PCs feature motherboards that come with National Semiconductor LM78 or similar chips which can measure fan speed, temperature and CPU voltage. For businesses this means they can ensure the environment in your server room is stable, and for desktop power users, they make sure your overclocked Athlon isn't about to melt.

A nifty package called 'lm_sensors' uses modules called sensors and i2c-core to measure and report the status of your system. There are, of course, a variety of graphical programs which will read this information and display it on your desktop, such as GnomeSensors and KHealthCare. Carry out a search on *freshmeat.net* for a selection.

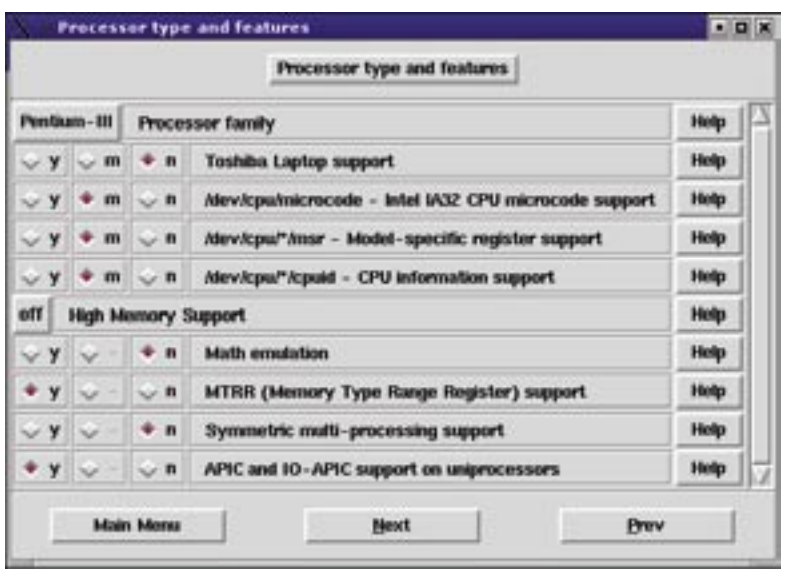
USEFUL APPLICATIONS

ksysguard

A process monitoring and management applet which is also capable of monitoring processes and threads across different processors in a graphical and comparative manner. It has five counters available for each CPU and a simple drag-and-drop interface.

GKrellM

This app has an SMP CPU monitor (which can also view all CPUs as a single virtual chip), current processes and a CPU/mainboard temperature display for the sensor chips installed on most modern SMP motherboards. Oh, and it's got some groovy skins, too.



Configuring CPU options in the kernel.

MULTITHREADED APPLICATIONS

To really get the most from an SMP system, you need to have system-intensive applications separated into multiple threads. This allows them to make use of both processors. Some of the better-known programs that support this are:

MesaGL
The 3D API used for hardware and software 3D rendering functions on Linux.

GKrellM has many system monitoring capabilities, including SMP monitoring and CPU temperature.

Quake III
id Software's first-person deathmatch extravanganza is one of the few games that support SMP.

The GIMP
The popular Linux photo manipulation application supports SMP for performing complex tasks on big images.

To find more, try doing an online search.

Network services

Networking

Linux is developed with, and helps create, the marvel that is networked communication. In this chapter we will show you how easy it is to use Linux to power your home or business.

NETWORKING AND LINUX, ONE AND THE SAME

Networking and related services are Linux's forte. The Internet was built on and still relies on open source Unix-based applications. Programs such as Apache, sendmail, wu-ftpd, Squid and BIND dominate their respective fields of Web, mail transport, FTP, caching and DNS serving. It's quite ironic that Microsoft makes claims about open source being a threat to innovation, and publishes these claims over an Internet that wouldn't exist without it.

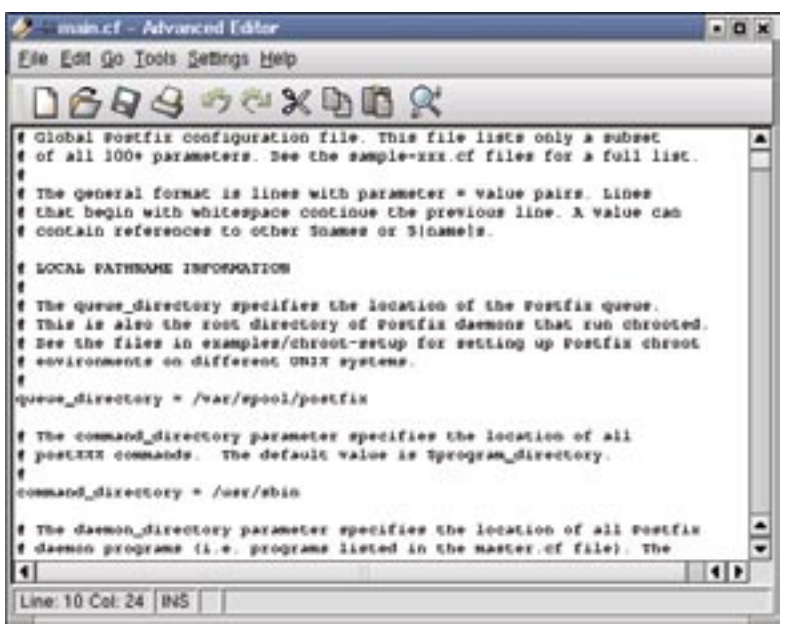
In the previous Linux Pocketbook we described how to operate a basic Linux desktop system. If you'd like to take your knowledge to the next level, and be the system administrator of your own network, this chapter is for you. Whether you're a home user taking your first steps into networking or a business person looking for a stable platform on which to build your IT infrastructure, you'll find Linux not only capable of but superior in performing the functions of most proprietary operating systems. This includes everything from dial-in PPP connections through to some of the fastest Web servers in the world. In addition, there are services (sometimes called 'daemons') that handle interoperating with Windows, MacOS, and other Unix-like OSes, so Linux will fit in well with your existing systems.

We have received a lot of requests from readers along the lines of 'please show us how to set up a Web server/FTP server/file server'. Here we'll do just that. This chapter is structured around the scenario of setting up a Linux box to support a number of network services. We will cover, in order: Web-based administration, Web serving, database serving, dynamic Web content serving, FTP serving, FTP and Web caching, mail serving, DHCP serving, Telnet and SSH (its modern equivalent), remote desktop access with VNC (virtual network computing), and time synchronisation with NTP clients.

Lastly, we'll take a stroll through the inbuilt firewalling in Linux 2.2 and 2.4, to make sure all these new servers are accessible only to the people you want to use them. While we're on the topic, run your distribution's updating tool (up2date, MandrakeUpdate, or apt-get for Red Hat, Mandrake, and Debian) to make sure the servers you'll be configuring are up to date with the latest security fixes.

Feel free to skip any section and configure only as many services as you require. For example, if, as is rapidly becoming the norm, you want to set up a machine to share a cable or ADSL service in a small business or home, you can just follow the steps required to configure firewalling (and Network Address Translation), proxy caching, DHCP serving, FTP serving and SMB serving. This will create an installation that is able to act as a firewall to protect your network, as a masquerader to share the Internet connection, speed up Web access through proxy caching, serve files over FTP, share files to networked Windows machines over SMB, and dole out IP addresses to the local network. Even a very low-end Pentium will have no trouble performing all these services on a single machine.

Each section of this chapter is dedicated to a single service and has a box detailing important information, a plain English description of what it does and the name of the particular server software we used in our tutorial — to let you know there's more than one way to skin a cat. We won't show you how to install each service, because that's just a matter of running rpm -ivh on the RPM file you can get from your distribution CDs or download fresh from the Internet (we'll provide Web



You can configure Linux services by editing their configuration file directly, or by using a GUI tool such as Webmin.

sites you can grab them from). In addition, we'll be providing the following information for each service:

Configuration files The files, usually in /etc, that you (or a configuration program acting on your behalf) edit to modify the settings of the service. As this is *The Advanced Linux Pocketbook*, and modern Linux applications have configuration files that are well commented and easy to edit, we'll be setting up most of our services by editing these files directly, then restarting the service to make the changes take effect. However, if you'd prefer to use a graphical tool, the information we'll provide is easy to apply to such tools.

Webmin configuration Whereabouts in Webmin to configure the service. Webmin is a Web-based, SSL (Secure Sockets Layer) secured remote configuration tool which we'll be setting up in the next section. Sometimes a Webmin module for a particular feature won't exist, in which case you'll need to venture into a configuration file. Check the Webmin site (*www.webmin.com*) often, however, as new administration modules are regularly added.

Packages The generic package names to look for if the service and its dependencies aren't already installed. Package names of common applications and services sometimes differ between distributions, but generally the main package name will always be present (for example, 'apache' in the name of any package that belongs to Apache). You can search for these on your distribution CDs or online.

Log files The files (in /var/log) you can examine to monitor the service. Unix servers use plain text file formats for logs, and append new entries at the end. At specific intervals (by default once a day), the logs are rotated — that is, some of the older entries at the top of the log are removed to make way for more entries on the bottom — this keeps the size of the logs down. The logs are also handy for troubleshooting — you can watch them update in realtime using tail -f [logfile] in a terminal.

Service configuration The method used to start the service. There are two ways to do this. The first option is for a permanently running server; when a server is started it attaches to a port and waits for connections. This is controlled by /etc/init.d/[servicename] start|stop|restart on all Linux distributions. Mandrake and Red Hat users can, however, simply type: service [servicename] start|stop|restart to do the same thing.

The second method is known as 'start on demand'. A single permanently running service called inetd or xinetd listens to ports on behalf of all such services, and starts them whenever a request is made to use the service. So, unlike permanently running services, those started through inetd and xinetd don't occupy any memory and CPU cycles while waiting for connections, because they are turned off. These 'super servers' are configured through /etc/inetd.conf and /etc/xinetd/xinetd.conf. To make the changes take effect, simply restart the inetd or xinetd service. Many services can be started through either method. Alternatively, you can also use ntsysv under Red Hat and Mandrake to select services to be started at boot time.

Access control Information about how the service decides who gets to use it. This is pretty important from a security point of view — you don't want the whole world accessing the confidential information on your intranet server. Nor do you want to provide a cache of popular overseas Web sites for everyone this side of the Pacific to use for free. Many services can be further locked down using TCP wrappers (the hosts.allow and hosts.deny files found in /etc).

Network ports There is more to consider than just an application's own access control when it comes to determining who has permission to use a service — any service can be limited to specific hosts, networks and protocols using a firewall. This section contains ports and protocols the service uses, and is important for configuring your firewall. The firewall should reject all incoming requests apart from those necessary for the services you'd like to expose to the untrusted network (typically, the Internet). Linux actually comes with a document describing the most commonly used network ports and the protocols that use them — take a look at your /etc/services file.

Now that we've covered what you'll find, it's time to explore the powerful networking features of Linux.

Webmin

Function	**Web-based administration**
Description	**Allows almost complete remote management through a browser**
Application used	**Webmin**
Packages	**webmin, perl, perl-devel**
Service name	webmin
Configuration file(s)	**/etc/webmin directory**
Log files	**/var/log/webmin directory**
Webmin configuration	**Webmin**
Service configuration	**Permanently running**
Access control	**Webmin users and capabilities menu**
Network port	**10000/TCP**

Webmin is a Web-based interface for remote administration. It comes with its own Web server and works in any modern browser that supports tables and forms. It handles a wide variety of common system administration tasks — including configuring most services in this chapter (and itself). There are also Java-based modules for file management and Telnet. It supports SSL for encrypted administration and can even be used to partition hard drives, set up RAID arrays, back up and burn CDs remotely, and install and remove software from a Linux system.

All Webmin requires is Perl 5.0 which, on any standard Linux system, will already be installed. If it isn't, check your distribution CDs for the relevant packages ('perl' and 'perl-devel') and install them. To use Webmin launch your favourite browser and point it to:

https://127.0.0.1:10000

Note the 'https'! Webmin uses SSL — if you get an error message about your connection being denied (or similar), then you've probably just typed http (or nothing at all). Of course, if your machine is on a network accessible over the Internet you need to type in the IP address of the machine when administering remotely — 127.0.0.1 is known as the 'localhost' (check your /etc/hosts file to see where this is set) and is only visible and usable for accessing services locally (that is, from the same machine).

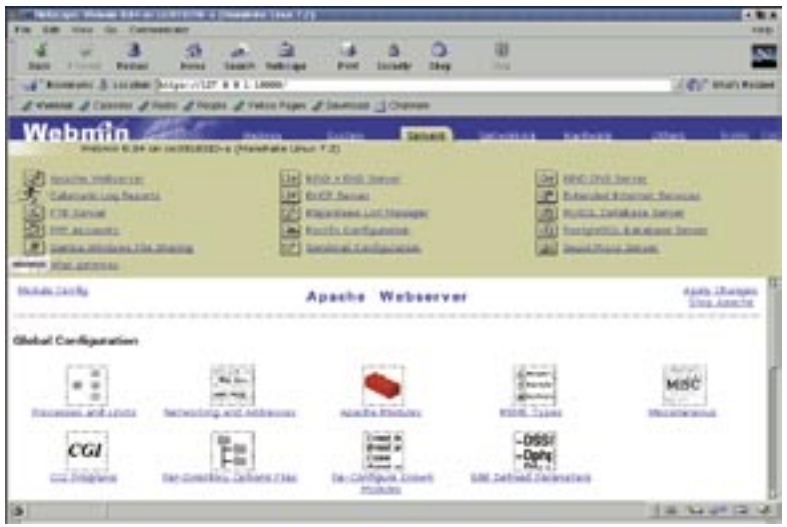
Your Web browser may give you some information about the site and present you with a digital certificate signed by an authority not known to the browser. Since we trust the Webmin developers (after all, they've given us their source to check), just click Yes.

You'll be presented with Webmin and its HTML user interface. Most of the options we'll be configuring in this chapter lie under Servers, though Webmin also provides options for system monitoring, hardware setup, and more. If you decide to use Webmin to set up the services in the rest of this chapter, click through until you find the service, modify the details for the service, and then click OK to submit your changes. Webmin checks the values you enter and will report on any bad entries. It checks your input more stringently than Linuxconf or similar tools, and generally makes it hard to misconfigure a service. We like this!

INSTALLING NEW MODULES

Webmin is modular in nature and many additional modules are available as downloads from *www.webmin.com*. Modules ranging from Java shells to SMS phone messaging are available, and many more are under development. Once you have downloaded and extracted a new module (Webmin uses the .WBM extension), enter the Webmin Configuration module and click on the Webmin Modules button. Then use the form at the top of the

Depending on your browser, you'll be asked to approve Webmin's digital signature when you first access the page.



Using Webmin to configure the Apache Web server.

page to install the module either from the local filesystem of the Webmin server, or uploaded from the client your browser is running on.

CONFIGURING WEBMIN

Webmin itself is configured through the front page of your Webmin site. One of the handiest options is using the Webmin 'users' option to delegate a limited set of administrative privileges to a user. This makes it easy to delegate administrative tasks (such as modifying user accounts) to other users without giving them full control of the system.

The Webmin configuration menu is also particularly handy, allowing you to modify the port Webmin runs at, what sort of information to log, and even themes (we've used the Caldera theme in our screenshot). It's even possible to tell Webmin to upgrade itself to the latest version over the Web.

FURTHER INFORMATION

Webmin home page
www.webmin.com

Caldera Systems (Webmin project sponsors)
www.caldera.com

TROUBLESHOOTING

Aside from https:// problems, the only notable error seen in Webmin is that some distributions use ports other than 10000. 1000 is used on many Linux variants — check your /usr/share/doc/[webmin version]/README file.

Web serving

Function	**HTTP serving**
Description	**Web serving**
Application used	**Apache**
Packages	**apache, apache-common, apache-manual, apache-devel, mod_ssl**
Service name	httpd
Configuration file(s)	/etc/httpd/httpd.conf
Log files	**/var/log/httpd directory**
Webmin configuration	**Servers ⟶ Apache Webserver**
Service configuration	**Permanently running (default) or start on demand**
Access control	**Configuration file directory permissions,** .htaccess **files, TCP wrappers**
Network port	**80/TCP, 443/TCP (SSL)**

The World Wide Web provides a simple method of publishing and linking information across the Internet, and is responsible for popularising the Internet to its current level. In the simplest case, a Web client (or browser), such as Netscape or Internet Explorer, connects with a Web server using a simple request/response protocol called HTTP (Hypertext Transfer Protocol), and requests HTML (Hypertext Markup Language) pages, images, Flash, and other objects.

In more modern situations, the Web server can also generate pages dynamically based on information returned from the user. Either way, setting up your own Web server is extremely simple.

CHOOSING A WEB SERVER

There are many choices available for Web serving under Linux. Some servers are very mature, such as Apache (which we'll show you how to set up here), and are perfect for small and large sites alike. Other servers are programmed to be light and fast, and to have only a limited feature set to reduce complexity. A search on *freshmeat.net* will reveal a multitude of servers, but here we'll just look at two to give you an idea of how performance and features can differ between them.

TUX

Tux is a new Web server from Red Hat which sits on top of Linux kernel 2.4's own inbuilt Web server, khttpd. Putting a Web server in the kernel dramatically increases performance, but can also expose a system to security risks. khttpd avoids this by only serving static content, passing any dynamic processing functions up to another Web server that lives in regular user space, like Tux.

khttpd and Tux perform well together — very well in fact. Identically configured Dell machines benchmarked on SPECweb99 (the benchmark used by Microsoft to promote IIS until this set of figures came out) revealed Tux outperformed Microsoft's IIS product by a factor of three. Tux is currently the fastest Web server for two, four or eight processors on IA32 compatible systems.

APACHE

Apache is the world's most popular Web server, serving over 65% of hosts online. It is open source and primarily Unix-based, though versions do exist for other platforms. It is also well supported in terms of add-on application software, available either as separate modules, or compiled into Apache itself for improved performance. These include SSL, URL spelling mistake fixing, various authentication modes and even FrontPage Server Extensions. For readers needing to create dynamic content, Apache also supports the widest range of server side scripting languages.

Tux is currently the fastest Web server for two, four or eight processors on IA32 compatible systems.

Apache isn't as fast as Tux, but its wide range of modules, good security and standards compliance more than make up for this. Additionally, various performance optimised versions exist from companies like IBM and SGI, including one that ships by default in Mandrake.

CONFIGURING APACHE

In this section we'll install a basic Apache setup with SSL, which can be done in a matter of minutes — the packages come with almost all distributions, the single configuration file is simple to read and well commented, and the default options work well.

Let's start by examining the configuration file, httpd.conf which is found in /etc/httpd or /etc/httpd/conf depending on your distribution. There are two types of entries in the file — regular settings, which appear on a single line in the format setting: currentvalue, and sections, which work similarly to HTML tags. These are opened and closed with <sectioname> and </sectioname>, with various settings between.

APACHE IN A LINUX/WINDOWS ENVIRONMENT

If you're using Apache or ProFTPd in a mixed Linux/Windows environment and you need to authenticate users based on credentials stored on a Windows NT server, Apache can use the SMB Pluggable Authentication Module to do this — see *www.apache.org* for more information.

You won't need to edit many settings, but here are some of the more important ones:

ServerAdmin This is the email address for the site administrator. It is important to enter a valid email address that you frequently check, in order for visitors to be able to report any problems.

Documentroot Indicates where on your disk your Web site will be located. This folder becomes the base from which Apache will serve files. For example, with the default document root of /var/www/html, a file called /var/www/html/index.html would appear on the Web site as /index.html. The default value for this varies — popular locations include /var/www, /home/www and /home/httpd.

Port Apache can listen on a port other than the default of 80, though Web browsers will have to specify this port to connect.

ServerName The hostname of your Apache server.

Hostname Same as Servername. Only used by certain modules.

Userdir By turning on (uncommenting) and defining Userdir, each user on your system will be able to put documents in /home/[username]/ [userdir] and have them appear on /[hostname]/ ~[username]/ on the Web site.

LoadModule and **AddModule** These statements configure Apache modules, defining the module name and object file, and the order in which to load modules, respectively.

AddHandler Sets up MIME types for the server, to match files of a certain type to applications which run them.

Errordocument A custom error page should also be set up, so visitors are presented with something more helpful than the standard '404 not found' errors (like the email address of someone to fix the bad link that sent your visitor there). For example:

Errordocument 404 /lame_excuses/not_found.html.

Servertype Sets Apache as permanently running (standalone) or start on demand (inetd).

LogLevel Determines how much information to log. There are eight

types of feedback from the server, labelled according to their importance: debug, info, notice, warn, error, crit, alert and emerg. LogLevel is defined as the minimum level of importance to log — for example, setting LogLevel warn logs warn, error, alert and emerg information.

Child processes This dictates the amount of child server processes to start — and should be adjusted based on the load of the Web server.

AccessFileName The name of the files used to specify access control options for a particular virtual directory, overriding options set by httpd.conf if they are set (and if httpd.conf allows these overrides). The default name is .htaccess.

HostnameLookups Defines whether to look up the hostnames of the client machines using your Web server, to give you a better idea of who's viewing your site. This defaults to off for privacy reasons. Once you've got the configuration file tweaked to your liking, restart the httpd service to make the changes take effect.

The Apache default page.

Then open a Web browser and type in your hostname to see the content in your document root — the Welcome to Apache' page that comes supplied with the Web server should pop up. You can now replace this with your own content, and if you're stuck for ideas, read on to the next few sections of this chapter.

SERVER STATUS

Apache comes with a facility to set up an inbuilt Web page that displays statistics from the actual server. You can access the page at http://[host name]/server-status from your own machine (remote access isn't allowed by default). It is configured as follows:

```
<Location /server-status>
SetHandler server-status
order deny,allow
deny from all
allow from localhost, 127.0.0.1
#allow from .your_domain.com
</Location>
```

ACCESS CONTROL

Apache supports TCP wrappers, but unlike most other applications that do so, it applies hosts.deny before hosts.allow. From a security standpoint, this isn't the best idea — access being forbidden should always override access being allowed. Furthermore, this is different from your other applications, and you only get one set of files. When you declare a directory (such as /), simply add the line order allow,deny to the section. The change will affect all subdirectories unless specified otherwise.

VIRTUAL HOSTING

Name-based virtual hosting allows a single IP address to serve more than one Web site. This ability is courtesy of HTTP 1.1, which is supported by almost all Web browsers. When a compliant Web browser asks for a page, it will also provide the name of the server it was looking for. The Web server will then use this information to provide content from the appropriate virtual host. There are other ways of providing virtual hosting, but the name-based method is by far the most popular, as it only requires a single IP address for all the virtual sites.

Virtual hosts are set up in httpd.conf. The options that can be set up for each virtual host are the same as those for the default host. Any options specified for a virtual host will override those set by the default host. A section may look like this:

```
<Virtualhost
pocketbook.example.com>
        Serveradmin webmaster@po
cketbook.example.com
        Documentroot /var/www/
pocketbooks.example.com
        Servername
pocketbooks.example.com
</Virtualhost>
```

TROUBLESHOOTING

Apache errors are usually simply typos in the configuration file. When

FURTHER INFORMATION

Local documentation for Apache
/home/httpd/html/manual/index.html

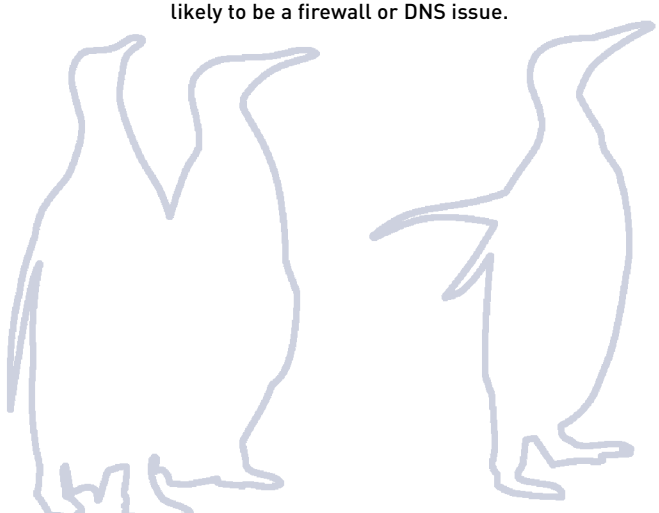
Apache Server FAQ
www.apache.org/docs/misc/FAQ.html

SGI Apache performance patches
oss.sgi.com/projects/apache

Tux Project Web site
www.redhat.com/products/software/ecommerce/tux

you restart the httpd service, it will check its configuration file and report any errors and their line numbers — open up httpd.conf and check the specified line to find and fix what's wrong.

If the server does start but seems difficult to access, firewall and DNS issues may be the problem. Try using a Telnet client to reach port 80 (telnet 127.0.0.1:80 — yes, you can telnet to a Web server). If you're able to login, and typing GET / returns a listing of your document root before exiting, then it's likely to be a firewall or DNS issue.



Database serving

Function	**Structured Query Language-based database serving**
Description	**Allows information to be fetched and entered into databases over networks**
Application used	**MySQL**
Packages	**mysql, mysql-shared, mysql-common**
Service name	**mysql**
Configuration file(s)	/etc/mysql
Log files	/var/lib/mysql/mysql.log
Webmin configuration	**Servers ⟶ MySQL database server**
Service configuration	**Permanently running**
Access control	**Permissions on databases, fields and tables**
Network port	**3306 TCP/UDP**

CHOOSING A DATABASE

Most databases are supported under Linux, including Oracle, DB2, Sybase, Informix, MySQL, PostgreSQL, InterBase and Paradox. Databases, and the Structured Query Language they work with, are complex, and this chapter has neither the space or depth to deal with them. Read the next section on PHP to learn how to set a dynamically generated Web portal in about five minutes.

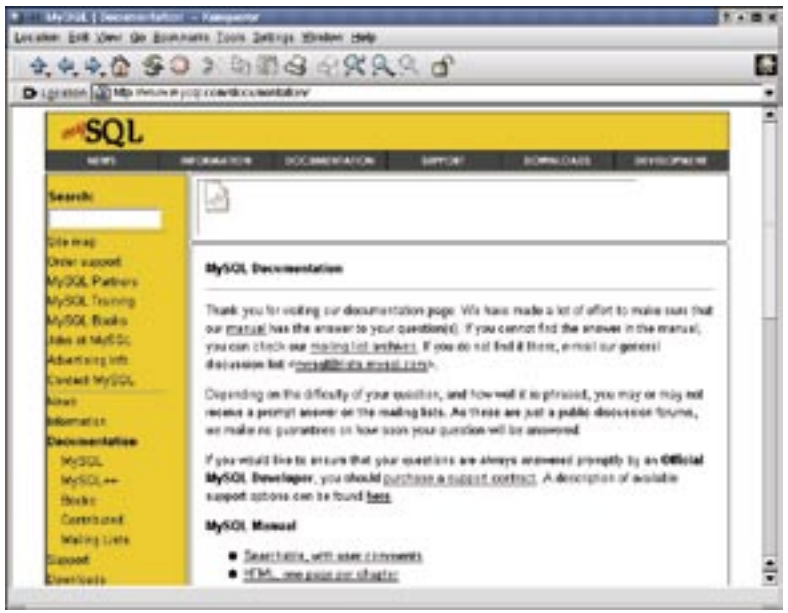
We'll be using MySQL because it's extremely fast, capable of handling large databases (200G databases aren't unheard of), and has recently been made open source. It also works well with PHP. While currently lacking transaction support (due to speed concerns), a future version of MySQL will have this option.

INSTALLING MYSQL

MySQL is typically distributed as three packages — 'mysql' for the database server, 'mysql-client' for the client machine, and 'mysql-shared' for both. Since our Web server will be on the same machine as our SQL server, it's important that you make sure that all three are installed.

TROUBLESHOOTING

See troubleshooting MySQL in conjunction with Apache and PHP in the next section.



For more information on MySQL, a full online manual is available for free at the MySQL Web

FURTHER INFORMATION

MySQL Web site
www.mysql.com

PostgreSQL
www.postgresql.org/index.html

Oracle
www.oracle.com

Borland InterBase
www.borland.com/interbase

IBM DB2
www-4.ibm.com/software/data/db2/linux

Sybase Adaptive Server Enterprise and SQL Anywhere
www.sybase.com/linux

Informix Dynamic Server — Linux Edition Suite
www.informix.com

Dynamic Web content genera-

Function	**Dynamic content generation**
Description	**Creates Web content based on information from databases**
Application used	**PHP Hypertext Preprocessor**
Packages	**php, mod_php, php-mysql (requires MySQL to be installed)**
Service Name	**N/A**
Configuration file(s)	/etc/php.ini **and configuration files for Web server**
Log files	**Same as Web server**
Webmin configuration	**Same as Web server**
Service configuration	**Started with Web server**
Access control	**Based on Web server, Web application, and database server permissions**
Network port	**Same as Web server**

CHOOSING A WEB SCRIPTING LANGUAGE

Web scripting languages are even more common on Linux than databases — basically, every language is available. This includes CGI, PHP 3 and 4, Perl, JSP, ASP (via closed source applications from Chili!Soft and Halcyon Software) and ColdFusion.

INTRODUCING PHP

PHP is an open source scripting language designed to churn out dynamically produced Web content ranging from databases to browsers. This includes not only HTML, but also graphics, Macromedia Flash and XML-based information. The latest versions of PHP provide impressive speed improvements, install easily from packages and can be set up quickly. PHP is the most popular Apache module and is used by over two million sites, including Amazon.com, US telco giant Sprint, Xoom Networks and Lycos. And unlike most other server side scripting languages, developers (or those that employ them) can add their own functions into the source code to improve it. Supported databases include all those mentioned in the Database serving' section and most ODBC-compliant databases.

The language itself borrows its structure from Perl and C. This book won't cover the scope of PHP programming, so we'll be setting up the ready-to-go PHP-Nuke Web portal software in this chapter. If you're interested in developing your own PHP code, see 'Further information' at the end of this section.

INSTALLING PHP

We'll be installing PHP as a Dynamic Shared Object (DSO) for Apache — this means it can be installed without having to recompile Apache. PHP also works with many other Web servers, including AOLServer, Roxen, Zeus and Netscape Enterprise Server.

Binary versions of PHP 4 are typically provided as three packages: one called 'php', which provides core PHP functionality, 'mod_php' which provides the Apache DSO module to link with it and a database connection module for the language — in our example 'php-mysql' for use with MySQL. The last package obviously depends on having the MySQL database installed, so install the necessary packages (see the previous section) first.

CONFIGURING APACHE

Open httpd.conf in your favorite editor and make the following changes.

You'll need a LoadModules statement to make the DSO module available to Apache. So add the following to the end of the existing LoadModule statements:

```
LoadModule php4_module modules/libphp4.so
```

The AddModule section covers both DSOs and static modules compiled into Apache.

Whenever you add a new module, you have to change the AddModule section, so Apache can load them in the correct order. At the end of all the AddModule statements, add:

```
AddModule mod_php4.c
```

Then, to associate .php files with PHP itself, add this to the bottom of your AddType statements:

```
AddType application/x-httpd-php4 .php4 .phtml .php
```

You'll now need to restart Apache to make the changes take effect (service httpd restart). Then test your mod_php installation works by creating a new file called phpinfo.php in your Apache document root (for example, /var/www/html). Its contents should simply read:

Viewing the build information of your PHP installation.

```
<? phpinfo() ?>
```

Save the file, check everyone has permission to read it, open a Web browser, and type http://[hostname]/phpinfo.php. With any luck, you'll see a page that looks a little like the screenshot above.

PHP-NUKE

PHP-Nuke is a powerful set of prewritten PHP scripts aimed at creating portal sites. It combines originally produced content with syndicated information from partner sites, and lets users customise what they want to see. This way, visitors are attracted to your site because it gives them content they're interested in. As well as news displays, Nuke provides comments and moderation, surveys, access statistics and referrer tracking, user registration, page theming, banner advertising management, a search engine, Web directories, headline generation in Netscape RDF format, content syndication, moderated

forums, content management, banner advertising control, and more. Nuke is administered through a friendly Web-based GUI.

INSTALLING NUKE

You'll find Nuke online at www.phpnuke.org . It should be unarchived to your Apache document root (typically, /var/www, but check your httpd.conf). To do this, change into that directory and run:

```
tar -zxvf /[location of file]/PHP-Nuke-4.3.tar.gz
```

Then change to the newly created /sql directory. A file called nuke.sql should be found which contains the commands to build the database structure, tables and the default data. Create a new database called, for example, 'nuke':

```
mysqladmin create nuke
```

Then fill the databases with the tables in the nuke.sql file with:

```
mysql nuke < nuke.sql
```

CONFIGURING NUKE

Point your browser to http://[hostname]/admin.php. Login with Nuke's default username and password:

```
AdminID: God
Password: Password
```

Nuke will present you with a range of administrative options. Keep in mind you can click the 'Online manual' button within any screen for a pop-up menu with help. Go into Preferences and tweak away — remember to change the basic details like site name, administrator email address, and the address to alert for incoming article submissions. While you're at it, customise the site theme, logos, slogans and other details to match your site. Lastly, change the default password!

PHP-Nuke makes building a fully featured interactive Web site simple.

TROUBLESHOOTING

Most of the errors that can occur installing PHP and MySQL are fairly simple. If your Web server reports:

```
Unsupported or undefined function mysql_pconnect
```

php_mysql isn't installed. Alternatively, if something like the following appears:

```
Warning: Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.sock' (111) in /var/www/html/mainfile.php on line 19. Unable to select database.
```

then MySQL isn't installed or running.

FURTHER INFORMATION

The official site of the PHP project
www.php.net

PHP Builder
www.phpbuilder.net

Zend, authors of the PHP 4 core engine
www.zend.com

Wired's Web site development resource
www.webmonkey.com

PHP Nuke
www.phpnuke.org

CNet's Web authoring and development site
www.builder.com

FTP serving

Function	**FTP serving**
Description	**A common Internet-based file transfer method that's more efficient than Web serving**
Application used	**ProFTPd**
Packages	**proftpd**
Service name	proftpd
Configuration file(s)	/etc/proftpd.conf
Log files	**By default logs to various files in /var/log/daemons**
Webmin configuration	**N/A**
Service configuration	**Permanently running (default), or start on demand**
Access control	**Configuration file directory permissions,** .ftpaccess **files**
Network port	**20/TCP, 21/TCP**

File Transfer Protocol (FTP) is an efficient way to transfer files between machines across networks. Clients and servers exist for almost all platforms, making FTP the most convenient (and therefore popular) method of transferring files.

There are two typical modes of running an FTP server — either anonymously or account-based. Anonymous FTP servers are by far the most popular; they allow any machine to access the FTP server and the files stored on it with the same permissions. No usernames and passwords are transmitted down the wire. Account-based FTP allows users to login with real usernames and passwords. While it provides greater access control than anonymous FTP, transmitting real usernames and passwords unencrypted over the Internet is generally avoided for security reasons. Thus we will be setting up anonymous FTP in our example.

CHOOSING AN FTP SERVER

WU-FTPD

The most popular FTP server on the Internet, the Washington University FTP daemon, comes with most Linux distributions, but has a rather poor reputation security-wise. Unlike ProFTPd which we will be using, a module for WU-FTPd comes with Webmin, which allows easy configuration through Webmin.

PROFTPD

ProFTPd is designed with greater security than WU-FTPd and offers better performance. It is configured in the often familiar style of the Apache Web server, so it's easy to get to grips with if you already understand Apache.

USING PROFTPD

It is possible that packages for WU-FTPd (including 'wu-ftpd' and 'anonftpd') may already be installed on your system. Since you are unable to have more than one FTP server at one time, uninstall them before installing the single ProFTPd package.

Like Apache, ProFTPd is just about ready to run out of the box. The server won't start unless your local hostname and IP address are defined in /etc/hosts, so make sure you check this first.

ProFTPd's document root is the home directory of the ftp user — typically /var/ftp. A file called /var/ftp/README would show up as /README to your FTP users.

Open /etc/proftpd.conf in your favorite editor. Like Apache configuration, ProFTPd uses HTML-like sections for some parts of its configuration file. The section which actually establishes your anonymous server should be similar to the following:

```
<Anonymous ~ftp>
        User ftp
        Group ftp
        UserAlias Anonymous ftp
        Requirevalidshell off
        Maxclients 10
        Displaylogin welcome.msg
        DisplayForstChdir .message
        <Limit WRITE>
                Denyall
        </Limit>
</Anonymous>
```

Some of the other configuration options are described below:

Servername The fully qualified domain name of your FTP server.

Servertype Can be set to standalone or inetd.

Port The standard FTP port is 21. You can change this, as long as you tell your FTP clients to connect to this port.

Maxclients The maximum number of anonymous logins — don't set it too high, just in case someone posts your latest software release to Slashdot . . .

Checkforvalidloginshell You should turn this option off, so that your anonymous FTP user (probably called 'ftp') can use FTP even if they don't have a valid system login shell.

Useralias allows additional names to be given to a single account. In our example above, Anonymous is an alias to the user ftp.

DisplayLogin This allows you to configure a welcome message displayed after a successful login. The default value is welcome.msg in your document root — modify the file as you please.

DisplayFirstChdir This message is displayed whenever a user enters a new directory. The default value is a file called .message in your document root.

Once you have the configuration file tweaked to your liking, start the service (if it isn't already running) and use an FTP client on your new server to login directly. A simple command line FTP client provides a good test, so type:

```
ftp 127.0.0.1
```

You'll be asked for a username — type anonymous, and then for a password just hit Enter. If your login was successful, your FTP server is up and ready to go.

FURTHER INFORMATION

ProFTPd official site
www.proftpd.net

WU-FTPd site
www.wu-ftpd.org

TROUBLESHOOTING

If your anonymous login is rejected, you may have forgotten to turn off CheckForValidLoginShell in your configuration file. Also, check to make sure the service started properly by looking for 'proftpd' with ps ax |grep proftpd.

Proxy caching

Function	**Web and FTP proxy cache**
Description	**A central store for frequently downloaded Web and FTP data**
Application used	**Squid**
Packages	**squid**
Service name	squid
Configuration file(s)	/etc/squid/squid.conf
Log files	**/var/log/squid directory**
Webmin configuration	**Servers ⟶ Squid proxy server**
Service configuration	**Permanently running**
Access control	**Access Control Lists in configuration files**
Network port	**3128/TCP**

INTRODUCING SQUID

When a Web browser retrieves information from the Internet, it stores a copy of that information in a cache on the local machine. When a user requests that information in future, the browser will check to see if the original source has updated; if not, the browser will simply use the cached version rather than fetch the data again.

By doing this, there is less information that needs to be downloaded, which makes the connection seem responsive to users and reduces bandwidth costs.

But if there are many browsers accessing the Internet through the same connection, it makes better sense to have a single, centralised cache so that once a single machine has requested some information, the next machine to try and download that information can also access it more quickly. This is the theory behind the proxy cache. Squid is by the far the most popular cache used on the Web, and can also be used to accelerate Web serving.

Although Squid is obviously useful for an ISP, large businesses or even a small office can use Squid to speed up transfers and save money, and it can easily be used to the same effect in a home with a few flatmates sharing a cable or ADSL connection.

THE SQUID LOGS

Note: The Squid log files monitor each object's access request to Squid. By default, the log files can grow very large very quickly. Make sure your disk space is sufficient, and if the files grow too large, set up cron (man cron **) to rotate Squid's logs more frequently.**

INSTALLING AND USING SQUID

As well as being a Web and FTP cache, Squid has other features such as multicast support, content acceleration and even basic content filtering. Once you've installed Squid, open the configuration file in your favorite editor. Common options are as follows:

Cachedir Make sure your cache is in a partition with enough free space for it to grow to its maximum size. Check the disk space on your partitions using df. The typical location is /var/spool/squid.

Cache size After using Squid for a while, you'll want to adjust the cache size. Choosing a correct value is a fine art, as it depends on the habits of your users. If your users use the Internet a lot and frequently visit a wide variety of sites, a large cache might work well. Of course, you don't want to fill your entire hard disk, so keep the size reasonable.

Cachemem Adjust this value to equal the amount of memory on your system.

cache_mgr The email address of the person running the proxy server — which should be you.

http_port The default Squid port is 3128, though many users prefer 8080. To keep it simple, comment out the icp_port and htcp_port values, which are used for cache server sharing.

Of course, Webmin can also be used to configure Squid.

ACCESS CONTROL

Squid uses Access Control Lists for fine control over who has permission to access the cache (after all, you don't want to be accelerating someone else's Internet connection and increasing your bandwidth charges). You'll need to define an ACL for your local network similar to the following:

```
acl local_network src
  192.168.0.0/255.255.255.0
  http_access allow local_network
```

The first line defines a group with the name local_network (the name is arbitrary) to handle any requests coming from the source (src) network 192.168.0.0, with a subnet mask of 255.255.255.0. The second line gives access to this group.

Squid can use Internet Caching Protocol (ICP) to participate in cache hierarchies. This allows proxy servers to request information from each other's caches, forward requests, and so on. Most business ISPs also provide caches to their clients, and the rate charged for data coming from these servers is less than that for non-cached data. A Squid cache can be the parent, sibling or child of another Squid cache, and by setting up your local Squid to be a child cache of your ISP you can save even more money. If you have a permanent connection with a business ISP, check if they offer peer caching. Most that do this already have instructions for setting up Squid, but they should appear similar to the following:

By setting up your local Squid to be a child cache of your ISP you can save even more money.

```
cache_peer [ISP cache hostname] parent 3128 0 no-query
```

Now you've got the file tweaked to your liking, you can start the service and begin configuring clients.

CONFIGURING CLIENTS

Configuring clients is performed within each Web browser under the heading of 'proxies'. In Netscape and Mozilla, this lives in Edit ⟶ Preferences ⟶ Advanced ⟶ Proxies. In Konqueror: Settings ⟶ Configure Konqueror ⟶ Proxies. In IE: Tools ⟶ Internet Options ⟶ Connections ⟶ LAN Settings. All you need to do is fill in the address and port for your proxy cache.

Email hosting

Function	**SMTP/IMAP4 and POP3**
Description	**Moves email between mail servers, stores it locally/Delivers incoming mail to clients when requested.**
Application used	**Postfix/IMAP Toolkit**
Packages	**postfix/imap**
Service name	postfix**/**imapd
Configuration file(s)	/etc/postfix**/None**
Log files	/var/log/maillog**/**/var/log/syslog
Webmin configuration	**Servers ⟶ Postfix/Servers ⟶ Extended Internet Services**
Service configuration	**Permanently running/Start on demand**
Access control	**Configuration file/User accounts**
Network port	**25/TCP (SMTP)/110/TCP (POP3), 143/TCP (IMAP4).**

ALL ABOUT MAIL

Alongside the Web, mail is the top reason for the popularity of the Internet. Email is an inexpensive and fast method of time-shifted messaging which, much like the Web, is actually based around sending and receiving plain text files. The protocol used is called the Simple Mail Transfer Protocol (SMTP). The server programs that implement SMTP to move mail from one server to another are called Mail Transfer Agents (MTAs).

In times gone by, users would Telnet into the SMTP server itself and use a command line mail program like elm or pine to check their mail. These days, users run email clients like Netscape, Evolution, KMail or Outlook on their desktop to check their email off a local SMTP server. Additional protocols like POP3 and IMAP4 are used between the SMTP server and desktop mail client to allow clients to manipulate files on, and download from, their local mail server. The programs that implement POP3 and IMAP4 are called Mail Delivery Agents (MDAs). They are generally separate from MTAs.

WHAT YOU'LL NEED

- An SMTP server to send email to other servers and receive mail from email clients.
- A POP3 or IMAP4 server to deliver email to email clients.
- A DNS MX record, which records a particular Internet host as being the mail server for a particular domain. You can arrange this with your DNS host — most likely your ISP.

CHOOSING AN MTA

SENDMAIL

Sendmail is one of the oldest mail servers used on the Internet and is extremely common, handling about 80% of the world's email. While it is popular, its age also means sendmail is slower and more difficult to configure than its modern counterparts and also has many legacy options available, which make it easy to misconfigure your server. Its configuration file is huge and complex. Additionally, all the various components of sendmail run as the root user, which means potential bugs in the program can be exploited to gain full control of the server.

POSTFIX

Postfix is a sendmail-compatible mail server with improved speed, ease of administration, and security. Originally written by IBM and also known as the

Email clients use the IMAP and POP3 protocols to download and manipulate email on their local mail servers.

HOW EMAIL WORKS

1. Joe Smith works for the Example company. Joe (joe@example.com**) uses Netscape Messenger to compose a message to his friend Mary (**mary@apcmag.com.au**).**

2. When Joe sends the mail, the mail client turns the message and all attachments into text format and sends the mail to the local mail server (called mail.example.com**) using the SMTP protocol.**

3. If Joe uses IMAP4 to interact with his mail server, he'll be keeping his messages on the server, so IMAP also sends a copy to of the message to his Sent Items folder on his server. If he's using POP3, he'll be storing his messages locally. So all that needs to happen is for his email client to copy the messages into his Sent Items on his desktop machine. POP3 won't be used at all for this step.

4. The mail server accepts that mail into the mail queue, where it waits to be sent to its destination.

5. When it is ready, mail.example.com **needs to find a mail server that will accept mail for** apcmag.com.au**. It does this by looking up the MX (Mail eXchange) record for** apcmag.com.au **from the** apcmag.com.au**'s DNS server (which stores this information). The MX for** apcmag.com.au **turns out to be** mail.apcmag.com.au**.**

6. mail.apcmag.com.au **receives the email from** example.com **using SMTP.**

7. The mail is received into the mail spool, the place where each user's messages are stored until their mail client receives them. This may be either a single large file called a mailbox **that lives in /var/spool/mail/mary or /home/mary, or individual files using the more modern** maildir **format, stored in /home/mary/Maildir. The format used depends on the choice of MTA and how it has been configured.**

8. Mary uses Outlook Express to download her mail, which retrieves the message (using POP3 or IMAP) from mail.apcmag.com.au **to her local machine.**

IBM Secure Mailer, it is used in many large-scale networks, has over 10 million users and is open source. We'll be using it for our tutorial.

QMAIL

Qmail is another MTA with most of the features of Postfix and some unique ones, such as excellent suitability for mailing lists (users can subscribe and unsubscribe by sending anything from their email address to a particular address, without worrying about the syntax of their messages). Qmail also has publicly available source code, but its unusual licence states that modifications can only be distributed as patches to the original version, making it more difficult to install and update.

Additionally, Qmail can only be made Sendmail compatible with various plugins, which duplicate Qmail's own functionality and can make it hard to understand. However, Qmail is fast, secure and recommended if Postfix isn't your cup of tea. Qmail running on FreeBSD powered Microsoft's Hotmail service until Microsoft decided it wasn't very good publicity for Hotmail not to be running Microsoft software.

OPENMAIL

HP's OpenMail is a groupware application for Linux which not only handles Internet standard transportation and delivery, but also collaborative functions. These enable it to be a drop-in replacement for Microsoft Exchange, including all Exchange/Outlook-specific functionality. It works very well and is free for installations of less than 60 users, with per user licensing for other situations.

CONFIGURING POSTFIX

There's a good chance an existing MTA may already be installed on your distribution. Check for Sendmail packages and uninstall them before continuing. Postfix itself installs as a single package.

Its various configuration files live in /etc/postfix, the main file being named (oddly enough), main.cf. The file is small and very well commented, and by default you won't need to change many options. Later, when you need to fine-tune or secure your server, you can investigate the rest of the file. The key elements are as follows:

myhostname specifies the Internet hostname of your new email server. This must be a fully qualified domain name — that is, your machine must be resolvable to this name from anywhere on the Internet. If you have a permanent connection, you should know this information already. This hostname, and your IP address, should also be listed in /etc/hosts.

mynetworks is the list of IP addresses and hostnames we allow to 'relay' — that is, use our mail server to send mail to users who aren't on that machine. This should most definitely be limited to your local network, as nasty folk like spammers will take advantage of your email servers. An example setup for mynetworks would be: mynetworks = 192.168.0.0/24, 127.0.0.0/8. **relay_domains** also limits who you can relay for, but based on domain information. For example:


```
relay_domains = *.example.com,
*.syd.example.com,
*.mel.example.com.
```

myorigin is the domain you specify your mail from — by default, the same as your actual domain.

home_mailbox specifies where mail should be delivered. This can be either the traditional mailbox format or Maildir, a more modern format. In our example, we'll use Maildir:

```
home_mailbox = Maildir/.
```

mail_spool_directory specifies where incoming mail will be kept. The default is /var/spool/mail.

smtpd_banner is displayed to other mail servers when they connect. By default Postfix displays the version number, but this information might be useful for potential crackers, so try this instead: smtpd_banner = $myhostname ESMTP $mail_name.

CHOOSING AN MDA

The first decision to make when choosing a mail delivery agent is which protocol to use.

POP3 is designed to allow users to download all their messages onto their desktops, where they can be read, sorted and eventually deleted. Mail only resides on the server until it is downloaded by the client, keeping down the amount of server disk space used.

IMAP is a later standard than POP3, and has a number of features which make it especially useful for both low bandwidth connections and situations where a user is checking email from a number of different machines. When an IMAP user checks their mail, their IMAP client first downloads the headers of each message and waits for the user to open each message before downloading it in full.

Additionally, IMAP stores a user's received mail folders on the server; so when a user downloads and organises their email, they modify information on the server, not their local machine. Thus IMAP works well for users who check their email from more than one location. However, as users keep their entire inbox and mail archives on the IMAP server, disk space can quickly reduce.

INSTALLING AN MDA

Mail delivery is a simple task, and there isn't much difference between the various MDA implementations. Install IMAP Toolkit, which provides start-on-demand IMAP4 and POP3 service. The package is typically called 'imap'. Once installed, simply make sure it's activated in, and then restart, xinet or inetd.

SETTING UP MAIL CLIENTS

Now you have an SMTP and a POP3/IMAP4 server, it's time to set up the clients in your office to deal with it. In Netscape, this can be performed using Edit ⟶ Preferences ⟶ Mail and Newsgroups ⟶ Mail Servers. KMail users should click Settings ⟶ Configuration ⟶ Network. Outlook users should use Tools ⟶ Services or Tools ⟶ Accounts, and be aware that Outlook's 'secure password authentication' is neither secure nor capable of working with Internet standards-based mail servers, so make sure it isn't turned on.

EMAIL EXTRAS

Of course, there's more to running a mail server than simply providing mail services. Day-to-day staff changes, such as moving to new locations, entering or leaving the company and going on holidays, will all require administrative tasks to be performed on the mail server.

MAKING SURE USERS CAN'T LOGIN TO THE SERVER

Once your email server is set up, you might wish to make sure that email accounts can use POP and IMAP to check their email, but can't log into the server itself (that is, via the shell or X) as a security precaution.

To take away this capability, first edit the /etc/shells file, which defines all the available shells on the system. Add /bin/false on its own line at the bottom of the file.

Then change each mail user's shell to /bin/false, either using Webmin, a program like KDE's User Manager (kuser) or by carefully editing the /etc/passwd file directly. Keep in mind that you'll have to restore the shell if you'd like to troubleshoot the mail account using a local email client.

ALIAS MAPS

Alias maps define the locations of the alias files. These allow you to receive mail for pretend users who don't actually have an account on your system by delivering the email to a real user on your system. The default location is /etc/postfix/aliases. Each line looks something like the following example:

```
joesmith:          jsmith
```

This would receive any incoming email addressed to joesmith, even though there isn't an account called joesmith on the system. The mail server would send the incoming mail to the jsmith account, which is a real system account. Aliases can send mail to accounts on the local server, or to another server entirely. For example:

```
joesmith:          jsmith@example.com
```

would receive any incoming mail for joesmith and send it to jsmith@example.com.

After you've made and saved any changes to the aliases file, run newaliases to make them take effect.

Aliases are useful for a number of reasons. They can make sure email addressed to common addresses like root or 'abuse' goes to the email account of the system administrator. They can prevent common mistakes in hard-to-spell email addresses. For example, if you have a user called stewart, it's fairly common for people to send to stuart instead.

Another common task is setting up a single alias for a group of users, so staff can mail an account called 'managers' rather than typing each manager's email address into their email client.

MAIL FORWARDING

When a user won't be able to check their office email account (they may be away on a business trip) but still intend to check their mail, forward files are often used. If a file called .forward exists in a user's home directory, it is used to set the account (or accounts) that future incoming mail will be forwarded to. For example, to send all of local user mike's email to an account at yahoo.com, create a file called /home/mike/.forward, with the following contents:

```
mike@yahoo.com, /mike
```

The first part (before the comma) specifies that mail should be forwarded to this external account. The next (/mike) specifies that email will also be forwarded to mike's local account, so when he returns to the office he still has all his messages in the one place (the slash character means 'a user on this machine'). You can forward to more accounts (remote or local) simply by using more commas.

VACATION MESSAGES

Of course, not everyone likes to read their email while they're on holiday. If a user is unable to answer their mail for an extended period of time, they may wish to inform the senders of incoming messages of these circumstances. In this case, the vacation program should be installed.

Packages are typically called 'vacation', and the program itself is activated through logging in as a user and typing vacation. This will bring up the vacation reply message in the user's default text editor. Edit the message as appropriate, then save the file — and you're done. Now when this user receives incoming mail, it will automatically be replied to with a message similar to:

```
Subject: away from my mail
Date: Tue, 27 Dec 2002 12:44:04
+1100 (EST)
From: mikem@pocketbooks.com.au
To: joe@example.com

I will not be reading my mail for a
while.
Your mail concerning "Howdy Mike!"
will be read when I'm back.
```

TROUBLESHOOTING

The most common problem with email is the bounce message — an email from a mail server somewhere telling your user that their message wasn't able to be transported to the appropriate mailserver. The cause for this is usually contained in the bounce message, so make sure users forward it to you if they don't understand it themselves.

The most common cause of bounce messages is users making spelling mistakes in email addresses. Other common messages are that the receiving user has filled their disk space quota (common if you're forwarding to Web-based email accounts), or that the user account no longer exists.

Since modern email setups consist of MTAs, MDAs and email clients, it's important to be able to isolate the particular area causing trouble. For example, certain email clients (all flavours of Microsoft Outlook) generate 'System Administrator' messages which aren't actually from the server, and merely report errors with the mail client itself (for example, a corrupted Outlook Personal Folder file). When you're unsure of where the problem lies, login and check/send email directly on the mail server using a basic mail client like Pine or Mutt.

If the problem is isolated to the mail server, checking the mail log file (/var/log/maillog) is always a good first step.

As with Web servers, it's also possible to Telnet directly to the SMTP port to test that the service is running and accessible.

FURTHER INFORMATION

Official Postfix Site
www.postfix.org

Tucows Linux guide to mail servers
howto.tucows.com/LDP/LDP/lasg/lasg-www/servers/email

Qmail and Qmail utilities
www.qmail.org

Official HP OpenMail site
www.openmail.com

DHCP serving

Function	**DHCP and BOOTP serving**
Description	**Automatically provides IP addressing and other information to systems that request it**
Application used	**Internet Software Consortium DHCPd**
Packages	**dhcp**
Service name	dhcpd
Configuration file(s)	/etc/dhcpd.conf
Log files	/var/log/dhcpd
Webmin configuration	**Servers ⟶ DHCP Server**
Service configuration	**Permanently running**
Access control	**Within configuration file**
Network port	**67/UDP, 68/UDP**

THE DHCP

Endeavouring to maintain static IP addressing information, such as IP addresses, subnet masks, DNS names and other information on client machines can be difficult. Documentation becomes lost or out-of-date, and network reconfigurations require details to be modified manually on every machine.

DHCP (Dynamic Host Configuration Protocol) solves this problem by providing arbitrary information (including IP addressing) to clients upon request. Almost all client OSes support it and it is the standard in most large networks.

There's not that much involved in getting your own DHCP server going. The first step is to decide on the information you wish to provide to clients. This will likely include standard Internet networking information, and may also contain other information such as WINS server location and NetBIOS node type for Windows-based PCs, or the location of kickstart files for Red Hat automated installs.

The following is a sample /etc/dhcpd.conf:

```
default-lease-time 86400;     #one day
max-lease-time 86400;         #one day
```

DHCP AND BROADBAND

We looked at DHCP clients when configuring Internet connections for cable modem providers in the previous Linux Pocketbook, and mentioned how Optus@Home users can use the -h **option on** dhcpcd **or** pump **to send their user numbers to Optus and establish a connection. Red Hat and Mandrake users can automate the process by adding:**

DHCP_HOSTNAME=[optus@home client number]

into

/etc/sysconfig/network-scripts/ifcfg-eth0**.**

```
subnet 192.168.0.0 netmask 255.255.255.
        {
        range 192.168.0.40 192.168.0.95;
option subnet-mask
option broadcast-address
option routers 192.168.0.1

#The IP address of the DNS server
option domain-name-servers 192.168.0.100
option domain name "example.com"

#For a Samba or Windows based WINS server
option netbios-name-servers
option netbios-dd-server
option netbios-node-type 8;
option netbios-scope "";
}
```

You can use DHCP to consistently assign the same information to a particular machine on your network based on its burnedin' (or MAC) address — a universally unique number on each network card. Thus you can maintain centralised configuration but sshave statically assigned IP addresses for machines which need them.

DHCP CLIENTS

The ISC DHCP server will work with any DHCP client on just about any OS, including Linux, Unix, Windows and MacOS.

Network filesystem

Function	**Network filesystem**
Description	**Unix-based file sharing**
Application used	**Linux 2.2 or 2.4 kernel**
Packages	**nfs-utils, nfs-utils-clients**
Service name	**Server:** nfs, portmap **Clients:** nfslock, netfs, portmap
Configuration file(s)	**Servers:** /etc/exports **Clients:** /etc/fstab
Log files	/var/log/syslog
Webmin configuration	**System ⟶ NFS Exports (to configure 'shares')**
Service configuration	**Permanently running**
Access control	/etc/exports **file**
Network port	**111/both, 369/both**

UNIX FILE SHARING

We looked at SMB, and its Linux implementation Samba, in *The Linux Pocketbook 2003 edition*. NFS is its Unix equivalent — a way to import and export local files to and from remote machines. Like SMB, NFS sends information including user passwords unencrypted, so it's best to limit it to within your local network.

As you know, all storage in Linux is visible within a single tree structure, and new hard disks, CD-ROMs, Zip drives and other storage spaces are mounted on a particular directory. NFS shares are also attached to the system in this manner. NFS is included in most Linux kernels, and the tools necessary to be an NFS server and client come in most distributions. However, users of Linux kernel 2.2 hoping to use NFS may wish to upgrade to kernel 2.4; while the earlier version of Linux NFS did work well, it was far slower than most other Unix implementations of the protocol.

NFS SERVING

Servers will need a package called 'nfs-utils'. NFS serving is controlled by the /etc/exports file on the server. The format of the file is simple: the local directory name, followed by the names of the machines to import it to (with optional permissions) as either hostnames or IP addresses. You can also use wildcards (asterisks) and specify permissions for the remote machine's use of the files and other options (see the man page). For example:

```
/home    *.office.example.com
/pub     fileserver.office.com (rw)
```

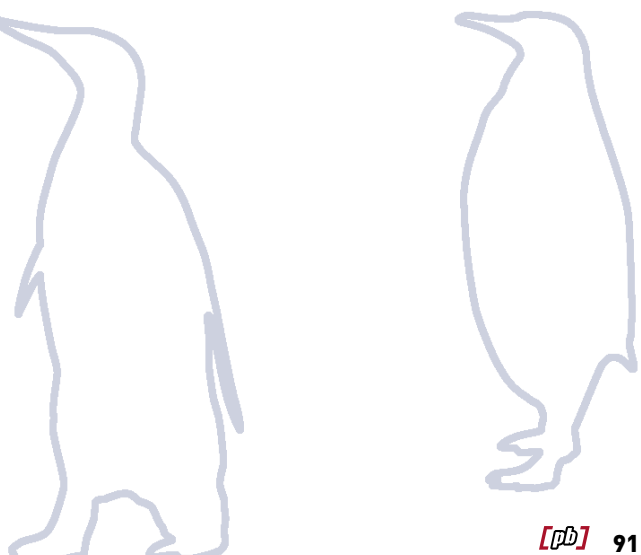
NFS CLIENTS

You can mount an NFS drive just like a disk or CD-ROM by using the mount command, or by editing the /etc/fstab file. This way, you can take advantage of a service run at boot time called netfs, which mounts any network-based directories (via NFS, SMB, or other means) listed in your fstab when it is started.

In addition, there are some extra options to specify when mounting NFS shares. Some can be rather nifty, such as the intr option which will allow client machines to recover if the NFS connection is interrupted. Once again, see the man page.

ROOT SQUASHING

Root squashing is a system where the root accounts on the NFS clients do not have the same permissions as root on the NFS server. You can turn the option off if you don't need it, but either way, don't assume that turning it on makes it safe to give out the root password to desktop machines. A desktop machine's root account can still use the su **command to get the permissions it needs.**



Telnet

Function	**Telnet serving**
Description	**Allows remote logins from Telnet clients over an unencrypted connection**
Application used	**Linux Telnetd**
Packages	**telnet-server, telnet**
Service name	telnetd
Configuration file(s)	**None**
Log files	/var/log/syslog
Webmin	**Servers ⟶ Extended Internet Services**
Service configuration	**Start on demand (default), permanently running available**
Access control	**Configuration file**
Network port	**23/TCP**

A WARNING

Like other protocols which emphasise access over security (such as SMB and NFS) it may well be acceptable to have a Telnet server in your local network. In fact, you may already have a few — nearly all network-aware printers and routers have Telnet as an available access method. But having Telnet installed on any machine with a direct connection to the Internet (such as your firewall and gateway machine) is, quite plainly, a bad idea.

Telnet sends all its information (including your passwords) in clear text streams that can be easily read by anyone between the machines. Performing any sort of administration via Telnet, whether it involves Telnetting in as 'root' (forbidden on most modern Linux distributions) or Telnetting in as a regular user and using the substitute user command (just as bad and unfortunately not banned), is not a clever idea.

Having Telnet installed on any machine with a direct connection to the Internet is, quite plainly, a bad idea.

Even if a cracker doesn't manage to get root access on your system, a regular user account and the latest vulnerability exploits can be used to acquire it. So don't use Telnet on Internet-connected machines — if you do, we'll send the Society of Eight Hundred Avenging fists over to your place to sort you out!

Telnet is rather ubiquitous, however. Clients and servers are available for all operating systems and it's also a handy way of running graphical applications remotely on X-based machines. What can replace it? See the next section, on SSH.

SETTING UP TELNETD

If you do need a Telnet server within your local network, then it is typically installed from a single package called 'telnet-server' or 'telnetd' and started on demand. Simply install the package and edit the inetd or xinetd configuration files appropriately.

Secure Shell

Function	**SSH serving**
Description	**Allows command line and graphical applications to be run safely over a remote connection**
Application used	**OpenSSH**
Packages	**openssh, openssh-server, openssh-clients, openssh-askpass**
Service name	sshd
Configuration file(s)	**Servers: /etc/ssh/sshd_config directory Clients: /etc/ssh/ssh_config directory**
Log files	/var/log/syslog
Webmin	**N/A**
Service configuration	**Permanently running, start on demand available**
Access control	**TCP wrappers (can also use** .rhosts **files)**
Network port	**22/both**

SAFE TELNETTING

The Secure Shell, or SSH, provides a way of running command line and graphical applications, and transferring files, over an encrypted connection. SSH uses up to 2,048-bit encryption with a variety of cryptographic schemes to make sure that if a cracker intercepts your connection, all they can see is useless gibberish. It is both a protocol and a suite of small command line applications which can be used for various functions.

SSH replaces the old Telnet application, and can be used for secure remote administration of machines across the Internet. However, it has more features.

SSH increases the ease of running applications remotely by setting up X permissions automatically. If you can log into a machine, it allows you to run a graphical application on it, unlike Telnet, which requires users to type lots of geeky xhost and xauth commands. SSH also has inbuilt compression, which allows your graphic applications to run much faster over the network.

SCP (Secure Copy) and SFTP (Secure FTP) allow transfer of files over the remote link, either via SSH's own command line utilities or graphical tools like Gnome's GFTP.

Like Telnet, SSH is cross-platform. You can find SSH servers and clients for Linux, Unix, all flavours of Windows, BeOS, PalmOS, Java and embedded OSes used in routers.

SETTING UP AN SSH SERVER

Setting up an SSH server is simple. The popular OpenSSH version of the service is usually distributed in three separate packages: 'openssh', 'openssh-clients' and 'openssh-server'. Also, all of the packages require OpenSSL to be installed first. It's best to install all three packages on your machine, and you should use a command line packaging tool to do so — the packages usually contain some interactive post-install scripts which ask for user input and won't show up when you're using a GUI installer. Once installed you can start SSH by running the service sshd.

USING SSH CLIENTS

Using SSH is also simple. To log into a remote machine with your currently logged in username, just type:

```
ssh [host]
```

Or, if the username of the other machine is different to the account you're using, you can type:

```
ssh [user]@[host]
```

To enable compression to speed up remote connections, particularly over a modem, add the '-C' switch.

KNOWN HOSTS

The first time the SSH client connects to a new server, it asks if the server should be added to a list of known hosts. Say Yes — this allows you to check whether the host key has changed during your future logons.

If it has, SSH will warn you with a message along the lines of 'SOMEONE MAY BE DOING SOMETHING NASTY!'. This means that the key returned by the remote SSH machine was different to one it had previously returned.

There are a number of reasons why this could happen: it may indeed mean someone has been fiddling with the remote SSH server (or is impersonating the remote machine), but it's also possible that the remote machine has had SSH reinstalled for some reason.

If you know the latter situation is the case, continue with the connection. Otherwise, you might wish to investigate . . .

SECURE COPY

SSH has a number of handy utilities, including SCP, a secure copying tool for moving files along the encrypted connection. You can use SCP independently of SSH (as long as there is an SSH server at one end of the connection). The syntax is easy — in order to copy a local file to a remote machine:

```
scp [local file to copy] [username]@
[remote machine]:[destination folder]
```

For example, to copy pocketbook.doc

from the current directory on your local machine to your home directory at example.com:

```
scp ./pocketbook.doc yourname@example.com:./
```

To copy a file /data/work/ on the remote machine to the current directory on the local machine:

```
scp yourname@example.com:/data/work/pocketbook.doc ./
```

There are a few nifty switches you can use as well, most notably -r for recursive copying, and -C to provide compression. Of course, there's also a graphical program you can use to do this. GFTP also handles SCP and provides a very simple drag and drop interface.

TROUBLESHOOTING

The most common reason for being unable to SSH into a particular host is that TCP wrappers and firewalls may be configured to limit SSH to a particular set of IP addresses or machine names.

Virtual Network Computing

Function	**Cross-platform remote desktop access**
Description	**Remote control of Linux and Windows desktops**
Application used	**VNC**
Packages	**vnc, vnc-server**
Service name	Xvnc **for server,** vncviewer **for clients**
Configuration file(s)	**None**
Log files	**/home/(username)/.vnc directory**
Webmin	**Others ⟶ VNC (for a VNC client)**
Service configuration	**N/A**
Access control	**Users can only connect to** Xvnc **sessions they started**
Network port	**59xx TCP/UDP for VNC itself. 58xx TCP/UDP for VNC's inbuilt Web server (used by the Java client). In both cases, 'xx' is the display number.**

THE POWER OF VNC

Virtual Network Computing (VNC) allows a user to operate a session running on another machine. Linux and all other Unix-like OSes already have this built in, but VNC provides further advantages because it's cross-platform, running on Linux, BSD, Unix, Win32, MacOS and PalmOS. Thus it is useful for a number of situations.

For example, you can use it to access applications running on another machine from a Linux desktop, or to provide support to all your Windows clients by taking control of their desktops from the comfort of your server room.

VNC is usually installed as separate packages for the client and server, typically named 'vnc' and 'vnc-server'.

VNC uses screen numbers to connect clients to servers. This is because Unix machines allow multiple graphical sessions to be started simultaneously (check this out yourself by logging in to a virtual terminal and typing startx -- :1).

For platforms (Windows, MacOS, Palm, etc) which don't have this capability, you'll connect to 'screen 0' and take over the session of the existing user. For Unix systems, you'll need to

Using VNC to control a remote Windows machine from a Linux desktop.

specify a higher number and receive a new desktop.

If you would prefer the Windows-style approach, where the VNC client takes over the currently running display, you can use x0rfbserver — see the sidebox below.

VNC SERVERS AND CLIENTS

On Linux, the VNC server (which allows the machine to be used remotely) is actually run as a replacement X server. To be able to start a VNC session to a machine, log into it and run vncserver. You'll be prompted for a password — in future you can change this password with the vncpasswd command. After you enter the password, you'll be told the display number of the newly created machine.

It is possible to control a remote machine by using the vncviewer command. If it is typed on its own it will prompt for a remote machine, or you can use: vncviewer [host]:[screen-number]

FURTHER INFORMATION

AT&T Research
www.uk.research.att.com/vnc

x0rfbserver
www.hexonet.de/software/x0rfbserver

NTP

Function	**Network Time Protocol clients**
Description	**Keeps the system time up to date from a remote NTP server**
Application used	**Clients Xntpd**
Packages	**ntp**
Service name	ntpd
Configuration file(s)	/etc/ntp.conf
Log files	/var/log/syslog
Webmin	**N/A**
Service configuration	**Permanently running**
Access control	**N/A (client)**
Network port	**123/both**

NTP is a standard method of synchronising time on a client from a remote server across the network. NTP clients are typically installed on servers.

There are a number of reasons why this is important: it makes sure your scheduled cron tasks on your various servers run well together, it allows better use of log files between machines to help troubleshoot problems, and synchronised, correct logs are also useful if your servers are ever attacked by crackers (either to report the attempt to organisations such as AusCERT or in court to use against the bad guys).

Users who have overclocked their machine might also use NTP to bring the time on their machines back to an accurate figure every 20 minutes or so.

All you need to do to configure NTP is comment out the existing server, fudge, and multicastclient lines in the /etc/ntp.conf file and add your own server line in the format: server [dns name or IP address]

Most business class ISPs provide NTP servers. Otherwise, there are a number of free NTP servers in Australia:

The University of Melbourne *ntp.cs.mu.oz.au*
University of Adelaide *ntp.saard.net*
CSIRO Marine Labs, Tasmania *ntp.ml.csiro.au*
CSIRO National Measurements Laboratory, Sydney *ntp.syd.dms.csiro.au*

Firewalling and masquerading

Function	**Packet filtering firewalling and Internet connection sharing**
Description	**Secure a system through packet filtering and flow control**
Application used	**Linux kernel 2.2 (**ipchains**)** **Linux kernel 2.4 (NetFilter/**iptables**)**
Packages	**ipchains/iptables**
Service Name	**Varies, depending on script used**
Configuration file(s)	**Varies, depending on script used**
Log files	/var/log/syslog
Webmin configuration	**N/A**
Service configuration	**Typically permanently running**
Access control	**It's a firewall, its purpose is control!**
Network port	**N/A**

FIREWALLING YOUR SYSTEM

As you already know, information over a network travels in packets. A firewall screens and limits these packets as they flow between network interfaces. In smaller networks, firewalls are used to protect desktop and server machines from those nasty people in the outside world called crackers, who break into machines and do whatever they feel like to the machine and its information. In larger installations they are used wherever two untrusting networks meet (for example, to protect an internal network from being attacked by crackers taking advantage of security holes in a publicly accessible Web server).

Firewalls have been essential to business users for some time now, and with the increase in permanently connected, high bandwidth home Internet connections (which are attractive targets for crackers), they are becoming necessary for home users too. The Linux 2.2 and 2.4 kernels contain inbuilt firewalling tools. Additionally, the same tools can be used for IP masquerading — a method of sharing an Internet connection to a network.

There's a big difference between Linux 2.2 and 2.4 when it comes to firewalling, masquerading and other aspects of security-focused networking. Linux 2.4's new Netfilter system (which provides a way for applications to control networking functions) contains iptables, a simpler and more powerful firewalling and masquerading system than Linux 2.2's ipchains. iptables provides stateful inspection (the ability to examine packets based on existing connections), integrated logging for rules, a wider range of ways for rules to specify which packets they affect, a wider range of tasks to do with packets that match those rules, and some tricky 'special effects'.

Packets also travel in a simpler way through the system, so iptables is easier to understand. If you're thinking about learning to firewall on Linux, install a 2.4 kernel if you're not already running one, and start with iptables.

For the rest of this section we'll show you how to set up your own firewall using some powerful and easy to configure firewalling scripts. Before we do, however, it helps to understand how firewalls work so you can better configure the firewall scripts for your system.

HOW CHAINS DETERMINE WHAT GOES INTO AND OUT OF YOUR MACHINE

For both ipchains and iptables packets must pass at least one set of *rules*. Rules define how packets travel through the machine (whether they are blocked, sent back, dropped, forwarded on, and so on). A set of rules is called a *chain*, and by default at least three chains will exist in every firewall configuration:

The input chain is used to manage packets coming into the machine.

In Linux 2.2, this is applied to all packets entering the firewall before they are passed onto applications or the forward chain (below). In Linux 2.4, it is applied only to packets entering the firewall and addressed to the firewall machine itself.

The forward chain is used in order to manipulate the destination of packets.

In Linux 2.2, this applies to all packets that have passed the input chain but are addressed to a different target — that is, packets being routed to other machines on a network. In Linux 2.4, the forward chain is applied immediately to any packets which enter the firewall but are not addressed to the firewall machine itself (thus bypassing the input chain).

The output chain is used to manage all outgoing packets.

Most of your interest will be in configuring these chains and the rules therein to match your preferences.

THE RULES THAT MAKE UP CHAINS

Each rule performs an action on any packets that match specified characteristics. These can be characteristics such as source and destination IP addresses, particular protocols, TCP and UDP ports, ICMP, particular network interfaces, or whether the packet is trying to start a TCP connection.

If the packet doesn't meet the conditions of a rule, it is sent to the

next rule in the chain. If the packet meets the conditions, it is sent to the *rules target*. Possible targets are:

ACCEPT: Pass the packet through untouched.

REJECT: Reject the packet and inform the host that created it that it was rejected.

DENY (2.2) or **DROP** (2.4): Drop the packet completely and do not inform the host that created it.

MASQ: Masquerade the packet. Typically, a network gateway will pretend (or masquerade) for other machines on a network so that those machines can communicate with the outside world without being directly connected to it. Masquerading is a type of Network Address Translation (NAT) called Port Address Translation.

REDIRECT: Used on the input chain, this will intercept a packet and send it to a port on the local machine.

RETURN: Use the default target for the chain, which is set by Accept, Reject, Deny or Masquerade.

Each chain also has a default target (or *policy*); if a packet misses all the rules in a chain, it is sent to this policy.

If this all sounds a little confusing just remember: packets that pass into a firewalled machine will be subject to a set of rules that determine how the packets should be handled. Through manipulation of these rules you can stop, pass on, redirect and manipulate those packets. This is how you can protect your machine from unwanted intruders and also set up handy services such as masquerading.

CREATING RULES

Once the appropriate modules are compiled in or loaded (use modprobe and see /lib/modules/[kernel]/ipv4 for 2.2 kernels and /lib/modules/[kernel]/kernel /net/ipv4/netfilter for 2.4) implementing a rule is performed through the ipchains and iptables commands.

In order to illustrate how chains are formed we will show you how to set up some simple and common rules. Although some commands differ between ipchains and iptables, wewill provide some example rules that use the same commands.

First, and to be paranoid (which is always good in security), we'll set the default policies (the -P switch) for the input, forward and output chains to DENY or DROP. This way, any packets that we do not set up rules to ACCEPT will automatically be denied.

ipchains: ipchains -P input DENY; ipchains -P forward DENY
iptables: iptables -P INPUT DROP; iptables -P FORWARD DROP

As you read earlier in this chapter, SSH is a good way of getting remote access to a machine over the Internet. So, let's create a rule (-A) to allow people to SSH into machines on our network. We want to check for packets using the TCP protocol (-p), of the type used by ssh. If we find packets that meet these criteria, we will ALLOW them. The rule is as follows:

ipchains: ipchains -A input -p tcp ssh -j ALLOW
iptables: iptables -A INPUT -p tcp ssh -j ALLOW

If we wanted to set up a firewall on a Web server that was publicly accessible by the IP 64.28.67.150, we'd want to ALLOW people to connect to port 80 on our machine, via this address, using the TCP protocol:

ipchains: ipchains -A input -p tcp -d 64.28.67.150 http -j ACCEPT
iptables: iptables -A INPUT -p tcp -d 64.28.67.150 http -j ACCEPT

To see these rules implemented, and other rules currently operating on a firewall, run ipchains or iptables with just the -L switch. To see a complete list of switches, run either command with — help.

THE FIREWALL SCRIPTS

Now you have some understanding of how Linux firewalls operate we'll introduce you to the two most popular scripts: Iridium Firewall for 2.2 kernels and MonMotha's Firewalling script for 2.4 kernels.

Using scripts such as these is far easier than building your own script of commands, as there are common rules that most firewalls will almost always implement.

To configure these scripts, just edit the script and enter the correct information at the start of the script — such as the network addresses it needs to operate with, the ports you'd like to keep open, wether you want to implement masquerading, and so on. Running them is as simple as making them executable with chmod and, if you want the firewall to be set up every time you boot up, add /bin/sh /[path to]/[script name] at the end of your /etc/rc.local file.

FURTHER INFORMATION

Official Netfilter and IPChains site
netfilter.kernelnotes.org

MonMotha's IPTables firewall (Linux 2.4)
t245.dyndns.org/~monmotha/firewall/index.php

Kurd's Iptables firewall (Linux 2.4)
my.netfilter.se

PMFirewall (Linux 2.2)
www.pointman.org/pmfirewall

Iridium Firewall (Linux 2.2)
www.karynova.com/iridium

USING CHAINS PRACTICALLY

Apart from these two very handy scripts there are, of course, a myriad other scripts, graphical interfaces for Gnome and KDE, and even some complex graphical firewalling programs that can help you tailor firewalling to your specific needs.

You don't need to know every intimate detail of ipchains and iptables to secure your box from crackers; scripts like those we've provided will configure everything for you. They'll guard your network, allow you to proxy for other machines, log intrusion attempts, and more.

TROUBLESHOOTING

If you want to set up your own custom rules, it's important to make sure you know what you're doing. Of particular importance is allowing the ICMP protocol (which is used by tools like ping and traceroute, as well as for negotiating details for connections). So is allowing use of the loopback interface (many Linux applications, like X, use loopback networking to perform their magic).

While most pre-made firewalling scripts will normally do this for you, be aware that if you construct your own chains manually, you need to add one more line to your script. This is to tell the kernel's networking code that you want to use the machine as a gateway. You can do so by changing a switch via the /proc filesystem:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

And you probably thought the /proc virtual filesystem was only useful as a static information system!

Security practices

Security is a state of mind

Your box is installed and your Internet connection is working. Now all you need is a firewall and you are safe and secure, right? Not quite.

Even if you have a firewall set up, what happens if the firewall code has a bug in it? Or if a bug is found in a program that can penetrate the firewall — your Web browser, for example — allowing an attacker into your system with the permissions of the user running the browser?

Security isn't an item on a checklist. You are not secure simply by virtue of the fact that you tick off 'firewall' on your list of things to do. System security is an ongoing process — it's a state of mind.

This chapter won't deal with the steps you can take to secure your system. We covered how to disable unnecessary services in *The Linux Pocketbook 2003 edition* and how to install a firewall in the networking section of this book. Rather, we'll deal with the most important facet of security — security habits.

THE SECURE SYSTEM

The only secure system is one that's turned off and stored in a locked safe in a locked cellar underneath a collapsed 20-storey building surrounded by barbed wire and a regiment of SAS with machine guns. Short of that, you must be prepared for the time when your box becomes 'owned' by an intruder who has root and is doing as they like.

Your efforts to secure your machine are wasted unless you can recover from a break-in, restoring your software and configuration files as well as your user data cleanly and safely.

First, you need to know what your system is doing and how that might make you vulnerable to internal and external attacks.

If you're a home modem user you are, by default, relatively secure. Modem-connected machines aren't permanently switched on and their IP addresses change every time they connect. There's some security in anonymity. However, this doesn't mean you are invulnerable to attacks. If someone is looking specifically for your machine, and they discover your IP address, whether or not they get in depends on the security precautions you have taken.

If you're a cable or ADSL user you should most certainly implement a firewall and full security precautions. A permanently connected machine is always a target. It's important to configure a secure machine, and then police it every now and then to ensure you have the latest security updates for your software.

If you're planning to secure a system for a business, small or large, you'll need to do all this and more. Daily checking and updating to the latest fixes are crucial for a high-profile, well-secured system.

If you're wondering if you need to be worrying about all this, just ask yourself a simple question: what value do you place on your data? If it's anything even remotely akin to 'I'd hate to lose it or have it stolen', then you need to be concerned with security.

Also remember that if you have multiple machines on a network, at home or in a business, sharing an Internet connection through a firewall-protected gateway, it's not just the gateway machine (the one that's visible to the Internet) that is a target. If the gateway machine is cracked, the whole network is then available to the intruder. The more machines you have on the network, the more information there is at risk if the gateway machine should be cracked and, therefore, the more imperative it is to properly secure the gateway machine.

SECURITY CHECK

Simple security questions you should consider:

- Are you running processes that might threaten your security from internal or external users?
- What exploits have been found this week and do you need to upgrade?
- When was your last backup?
- What are the security implications of the program you just installed or the configuration change you just made?
- Can you restore the software you just installed complete with the configuration changes you just made? How do you do that?

WHAT VULNERABLE SOFTWARE ARE YOU RUNNING?

The first place to start is the current state of your system. What software is running and why? Where are the configuration files — do you have a copy of them on backups? Look at each program started at bootup. What does it do and do you need it? Does it sit and listen on the Net?

Anything that listens is a door. If an attacker rattles your doorknob with their portscanner and sees that you have a door installed that they have a key for, then they're in. If there's no door, it makes their life a lot harder. Some doors are required, or you'd never leave the 'house', but those doors should be the latest model, solid and deadlocked, and treated with care.

Two good tools you can use to find what services are listening are netstat and lsof (LiSt Open Files). netstat is installed by default on most Unix installations, whereas you'll probably have to install lsof yourself. lsof is a little easier to read and has other uses, such as locating the files a suspect process is using.

As root, type lsof -i to see a list of all network ports open and which ones are in use. Look at each connection to determine what it is, and why you are running it. The equivalent netstat command is netstat -pa. For example, if you see:

```
in.identd 361 root  4u IPv4  319    TCP *:auth (LISTEN)
```

then that's the 'identd' daemon listening on the 'auth' port. Trying the man page for 'in.identd' tells you what it does and why. To see which ports are used for which services, see the /etc/services file. If there is no man page for a service, you can use lsof to find what files are open, and rpm or dpkg to tell you what package it is from.

For example, say you discovered you were running 'xisp' from the following output from lsof:

```
xisp   3228 root  4u IPv4 12179    TCP localhost:1724->localhost:6000
(ESTABLISHED)
```

To determine which files are associated with the process, just run lsof -c xisp and get a selection of lines, including:

```
xisp  3228 root txt  REG  3,1 296696 210496 /usr/X11R6/bin/xisp
```

From this we can use our package manager to determine which package that file belongs to using: rpm -qf /usr/X11R6/bin/xisp or dpkg -S/usr/X11R6/bin/ xisp for Debian.

This tells us the program is from the 'xisp' package (wasn't hard to guess that one!), and we can query the package's purpose and its files using: rpm -qi xisp or apt-cache--full search xisp for Debian.

As it turns out in our example, xisp is just a handy dialling program, so it's OK to have running. However, if you discover a program running that doesn't belong to a package you installed, or source code you compiled, it's possible a cracker has accessed your machine and installed their own software.

It is a good idea to run lsof -i regularly to ensure nothing is listening that shouldn't be. And, of course, if it's a service you don't need, then disable it (in inetd or einetd, or using ntsysv. A good rule of thumb is: 'If you don't know what it does, shut it down. If it doesn't make any difference, you don't need it.'

ARE YOU RUNNING THE LATEST SOFTWARE?

After you have installed only the doors that need to be there, you need to ensure that they are the latest models with resistant locks.

The first place to check is the home page for your distribution where updates can be downloaded, or you can use the automatic updating software for your distribution.

The next place to check is SecurityFocus (at *www.securityfocus.com*). This is the home of the Bugtraq mailing list and contains a list of known exploits. If you're securing a high-profile system, you should check this every time you sit down for some Net surfing so that you can update or reconfigure before every skript kiddie tries their latest cracking scripts on your system.

The more time that elapses between the announcement of an exploit and when you deal with it, the more time someone has to break into your box. Even if it's not an exploit that can be taken advantage of over the Internet, don't leave a known vulnerability on your system waiting for an *unknown* one to give a cracker a door into your root account.

OLD AND NEW SOFTWARE — IS IT SECURE?

Each software package you install, from CD or the Net, is a potential hole. So how do you go about reducing the size of that hole to minimise the risk?

First, know what you are installing, what it does and what files it uses to do it, where it puts these files and who has access to them.

Whether you are using rpm, dpkg or tarballs, you will have a list of files or at least a README file. Read the README, skim the doc files and the man page, and check the file list. You are looking for configuration files and passwords. If the software uses a configuration file, then run an eye over it, looking for any useful information.

If the package listens on a port, why? Does the whole world need access to that port or should you apply a firewall so that only people you know can access it?

If it uses passwords that are in cleartext, can normal users find and read them, or are they visible only to users of your Web site? If so, can you move them or change the permissions to stop that? Is there an option to use encrypted passwords? Are you happy with the risk those passwords present, or will you find a different package?

Do you have control over who can run the program? Is it OK for anyone who logs in to run it, or should you change permissions so only certain people can? What are the permissions on the data files and are those safe enough for you? Should you change the ownership and permissions so only certain people can read the data or run the program, and will the program run if you do? Is the setUID bit set and does it need to be?

How much effort you put into permissions will depend on who else uses your machine. But get

into the habit of at least looking. Not every bit of software you get from the Net has been put together by someone with security as their priority. And shrinkwrap is no protection; off-the-shelf programs are just as liable to have gaping holes as those downloaded from the Web.

If you install something that requires configuration, then configure it and back up the config files to floppy or CD, or some other removable medium. Label it and store it and update it when you change the config. That way, should your hard disk die or your system succumb to a cracker, you can recreate it the way you like it without the hard slog.

If you have a large-capacity backup device such as a Zip drive or a CD, then you can make regular copies of /etc which will catch almost all configuration files. Naturally, don't assume it will. Check each package you use to find out where it keeps its config files and add that to your backup.

If you install something that uses a password — MySQL, for example — then immediately change the default. Immediately. Do not let anything even *sit* on your system, let alone run, with the default password.

Choosing decent passwords can be a chore. There are many different methods for choosing passwords, but an easy one is to take a simple phrase — the line of a song or poem — and use the first letters from each word, including capitals and perhaps punctuation. So, 'Australians all let us rejoice for we are young and free!' becomes this 12-character password: AalurFwayaf!. The song verse means it's easy for you to remember but hard for anyone to guess, and it's unlikely that such a combination would be easily worked out by a cracking program.

Similarly, anyone who has access to your system should have a sensible password. It doesn't have to be complex — the easier it is to remember the less likely a user is to write it down where it can be found. But given a choice, many people will choose silly passwords: 'hello', 'football', 'please-crack-me'.

Remember, cracking software works from dictionary files. That's why when choosing a password for your ISP account you were asked to mix numbers and letters, using both capitals and lowercase. This is good advice to follow. Even 'He11o65' is infinitely harder to determine by software or mental prowess than 'hello'.

So check that your users have chosen a good password. All the firewalling configuration in the world will mean naught if a cracker gets in using a legitimate account.

All that said, if Linux is just a hobby for you on the home PC, then you can get by without being too stringent. However, if you're planning to configure systems for a business, or anywhere where information is important, the habits you learn now are the ones that will stick with you in the future.

Cracked . . . Say your home or small business gateway server has been cracked, or you're responsible for a server at a major corporation and a cracker gets in despite all your defences. What then?

You've found files that shouldn't be there. You've noted a process that seems to have snuck in and set up shop, and *you* didn't load any new software. Your log files are amnesic or missing.

You are owned.

Someone with a nickname full of numbers and misspellings has taken up residence and is using your system to attack other root accounts, store warez or run a distributed Denial of Service attack against someone they don't like, like the FBI. Now what?

Unfortunately the news is all bad. Time to reformat and reinstall.

You have no idea which critical files in /sbin or /usr/bin or anywhere else are not what they seem (especially if log files have been tampered with). You don't know what little time bombs the cracker has left hiding in home directories or lurking deep down in /var/lib/doom to bring your system to its knees at a later date.

You will need to reformat your drives and reinstall your system. You may not have to remove /home, if you use chmod to strip the execute bit from everything in it and replace any executables and dotfiles there with known good copies. But nothing that is normally run or that can alter an aspect of your system should be left untouched once a cracker has been in.

Overly paranoid? Maybe. After all, most crackers are skript kiddies with a root kit they downloaded from the Net. They don't know what it does, they can barely type 'cd /usr/sbin' and they aren't capable of much more than loading an IRC script and boasting about it to their mates at school.

But you may strike a cracker who **does** have a clue. For a home system, this isn't too much of an issue. Still, you wouldn't want the authorities coming after you should they discover attacks originating from your machine because the cracker has come back, knowing the back doors, and abused your Internet access. But if you run an important machine, don't take the risk — wipe it clean.

A sensible system administrator makes this whole messy business easier on themselves by keeping backups. Good backups consist of three kinds of data:

System software
This can be found on your installation CDs, so you already have that.

System updates
You don't need to back up RPMs or Debian packages you downloaded from the Net; you can get them again as long as you keep a record of which ones you need. But you do need to keep a record of your configuration changes, even if it's as simple as scribbling a few notes in a notebook that says 'X windows config — horizontal frequency 65Hz, vertical 60-90, check www.rage.com for latest driver'.

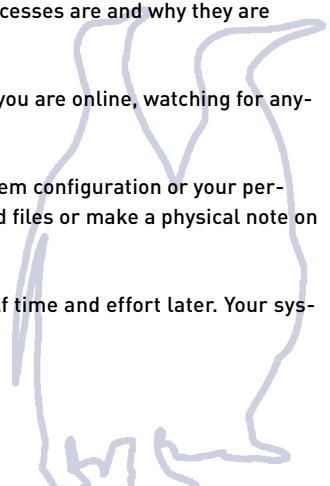
Private user data
Your email collection, your address book, your Doom cheat files, your personal dotfiles. The method you use for backing up will depend on personal preference. You may find floppies and a notebook are adequate; or you may decide to use this as the perfect excuse to buy a CD writer. Remember that an old backup is worse than no backup at all. Worse than this, though, is a backup that you can't read. Always check that you can retrieve your important data.

SECURITY IS A MATTER OF HABIT, NOT SOFTWARE.

Lastly, you should always:

- Keep all your software up to date with the latest security fixes.
- Regularly check what's running on your system and what is talking to the Net, making sure you know what those processes are and why they are there.
- Run an eye over your log files for the time you are online, watching for anything odd.
- Every time you make a change to your system configuration or your personal configuration, back up those changed files or make a physical note on paper.

Get into good habits now and save yourself time and effort later. Your system is only as secure as you make it.



Extending permissions

In the previous Linux Pocketbook we covered the use of permissions. Here we'll explore permissions in relation to security.

PROGRAMS THAT RUN AS ANOTHER USER

As we discussed in the previous Pocketbook, when a user runs a program, the program runs with the same permissions as that user account. For example, if user Joe doesn't have permission to modify the files in /etc, any program that Joe runs won't have permission either.

There's an exception to every rule. Files that have the *SetUID* bit set can run with the permissions of the user that owns them. For example, if the /usr/bin/xcdroast program was owned by the user root, and had the SetUID option turned on, and Joe ran X-CD-Roast, then X-CD-Roast would be able to do anything that the root user could. In this way, SetUID is handy to allow programs to run with permissions ordinary users don't have. In the case of X-CD-Roast, the burning software needs root access in order to use devices attached to the system.

Another example is SSH. Like X-CD-Roast, it needs to be able to perform tasks that are normally the province of root, so the SSH program file is owned by root, and it has the SetUID bit set. This way any user can run SSH and it will work fine. A listing for /usr/bin/ssh will show something like this:

```
-rws--x--x    1 root     root       183772 Nov 14 13:45 /usr/bin/ssh
```

Looks like a normal listing? Check again. Where you'd expect an 'x' for owner, there's an 's'. This is the SetUID bit. You can set the bit with:

```
chmod u+s [file]
chmod u-s [file]
```

to add or remove the bit respectively. But keep reading before you add this bit to a file, because SetUID programs (especially those owned by root) have some important security implications. If X-CD-Roast or SSH had a security hole, a cracker could perhaps use them to start other programs. Since these programs run as root, so do other programs started from them. Once a cracker can do this

on your system, they have free reign to view, modify, move and delete any file on your system.

SetUID programs (especially those owned by the root user) should be limited on vital servers that are exposed to the outside world. Even though you have a firewall (you do have a firewall, don't you? If not, see the Networking chapter!), being extra cautious about security is always a good thing. Hence users who own server machines, even with firewalls, should avoid installing applications which use SetUID unnecessarily. For example, on Linux systems, to make a server program listen on a particular port below 1024, root permission is necessary.

The older sendmail system does this and also uses SetUID root programs to handle other tasks such as queuing outgoing mail that don't need root permission at all. If a bug was found in the part of sendmail that queues mail, it could have disastrous effects. Postfix (which we set up in the Networking chapter), for example, is smarter. It only runs as root to bind Postfix to port 25 and then uses programs that run as other user accounts to handle mail queuing and all other tasks.

So how do you find files with SetUID activated? As described in the previous Linux Pocketbook, you can use the find command to find all sorts of information. Before you start chewing up your disk access searching your entire system (you're not going to find any SetUID programs in your MP3 archive directory), think about where SetUID programs might be placed, and search those directories . . .

It's absolutely imperative to close off all possible exploits on a system, no matter how remote the possibility.

Unless you have reason to believe otherwise, restrict your search recursively to /usr and /sbin. Run the following command as root:

```
find /usr -perm -u=s
```

Chances are you won't know the purposes of all the displayed programs, and you're going to need to read up on each (through its Web site, man page or accompanying documentation, or on Linux Web sites) to determine if it'll run smoothly with SetUID tuned off. This is why this step of security is only appropriate if it's absolutely imperative to close off all possible exploits on a system, no matter how remote the possibility (that is, if you're running a publicly accessible server). Security can be hard work!

DEFAULT PERMISSIONS

In *The Linux Pocketbook 2003 edition* we looked at how permissions work. We covered how each user has a primary group and that when that user creates a file, it will be owned by that user and the group will be set to their primary group. What we didn't cover was the use of the *umask* — the default permissions for a file specific to each user. When you create a new file it will have a set of default permissions defined by the umask. You can see the default permissions for files of a particular user with:

Users with server machines, even with firewalls, should not install applications which use Set UID unnecessarily.

```
umask -S
```

and change the settings of the umask (though you probably won't want to) with:

```
umask -S [permissions]
```

The permissions you can specify are in the same format as for chmod.

UMASKS AND DEFAULT GROUPS

In Red Hat and Mandrake, each user's default primary group is a private group. In the example below:

```
-rw- r-- ---    1 mike    mike    1000 Feb 13 00:24 pocketbook.txt
```

the file is owned by the user 'mike'. The file's group is mike's private group (that is, a group called 'mike' in which the user called mike is the only member). The default permissions give the user (mike) read and write access. The group gets read access, and all other users have no access to the file.

In Debian:

```
-rw- --- ---    1 mike    users    1000 Feb 13 00:24 pocketbook.txt
```

the file is owned by the user 'mike'. The file's group is mike's primary group, which is 'users'. The default permissions give the user read and write access to the file; the group and other users don't get access at all.

In both cases, by default, no other accounts have access to new files mike makes. So Red Hat, Mandrake and Debian use different methods in order to achieve the same result. But why is it that Red Hat and Mandrake seem complicated? Private groups make it easier to share files, as we'll now show you.

CREATING SHARED DIRECTORIES WITH SETGID

Restricting default permissions so that other people can't view your files is

great — especially when composing emails to your significant other. But if people had to change the permissions on every new file they wanted to share with others, life would be difficult. Linux (and Unix-like OSes in general) neatly overcomes this problem with the SetGID option.

SetGID is typically set on a directory where more than one user shares files. Think of it as your shared directory permission.

```
chgrp finance /shared
```

will change the /shared directory's group to the 'finance' group.

```
chmod g+xs /shared
```

will turn on SetGID for the /shared directory.

Now any new files created in this directory will inherit their group not from the user that made the file, but from the directory's own group. That is, all new files created in /shared will have their group set to 'finance', regardless of who creates them. Combined with a umask that gives permission to the group (default in Red Hat and Mandrake, and configurable on Debian) this means that any files a user saves to /shared will be instantly accessible to all the other accounts in the finance group.

Maximising Linux

Top 10 handy habits

Some things you just learn from experience. Following is a list of the top 10 handy habits experienced Linux users have come to live by.

10 HOST IT, BABY!

Tired of typing in long IP addresses when you Telnet or FTP to a host? Do you find it hard to remember the addresses themselves? Not a problem: throw the addresses and hostnames into your /etc/hosts file.

In fact, you should add the IP addresses and hostnames of all machines on the local network you expect to interact with. It's not just a matter of telnet Hal being easier to type than telnet 192.68.0.2 — many parts of your Linux system will look to the hosts file to find out the addresses of other hosts.

You'll notice an entry for 'localhost'. Now you know why you can do nifty things like type 'localhost' in your browser to access services such as Webmin and Swat.

9 KILL AND KILLALL

The kill command we introduced in *The Linux Pocketbook* can be used to terminate a process by passing kill a process ID (PID). By default, kill just tells the process to die, but if the process has made previous arrangements with the Linux kernel it is sometimes able to ignore the request. If you want to force a stubborn process to die, you can use the '-9' or '-KILL' switch to force it to die.

One way to tell whether a process actually did die is to simply repeat the kill command by pressing your up-arrow and then Enter. If the process has died, kill will produce an error message 'No such pid'; if the process hasn't died, kill won't output anything.

The kill command is more general than its name implies. It can be used to terminate a process, but this is just a case of using *signals*. A signal is a single number that gets sent to a process by the Linux kernel (either directly, or from another process). The kill command just happens to be a convenient way to send any signal to any process from the command line by specifying the signal number or name. In the previous example, '9' was the signal number, and 'KILL' was the signal name; to get a listing of all signals, type kill -l.

As an example of a nonviolent signal, number 10 is 'USR1', which is the first user-defined signal; this is by default ignored by processes, but it can trigger some application-specific behaviour.

For example, if you send the Apache Web server a USR1 signal, it will perform a reload of its configuration files — a quick and easy way to initialise any changes.

The killall command is a handy shortcut for sending a signal to a process if you know the name of the process, but not its PID. For example, you can kill your pppd daemon at the command line with killall pppd without ever issuing a ps to look up the process ID.

It's also useful if you want to kill multiple instances of a program without individually looking up and terminating each process with ps and kill. For example, if Netscape has bombed out on you and you have multiple instances running, you can terminate them all at once with killall netscape-communicator.

A word of warning though: if you ever happen to be using a non-Linux Unix computer, don't use killall — on many other Unixes, this will kill many more processes than you would expect it to!

8 COPYING AND PASTING

Even if they don't have Edit —→ Copy and Edit —→ Paste menus, most X applications support copying and pasting of text between windows using only the mouse. In its simplest form, copying is just a question of highlighting the relevant text by clicking and dragging.

To paste, just put the mouse pointer where you want to paste, and press your middle mouse button. If you only have a two-button mouse, X is most likely set up so that pressing both buttons together simulates the middle mouse button.

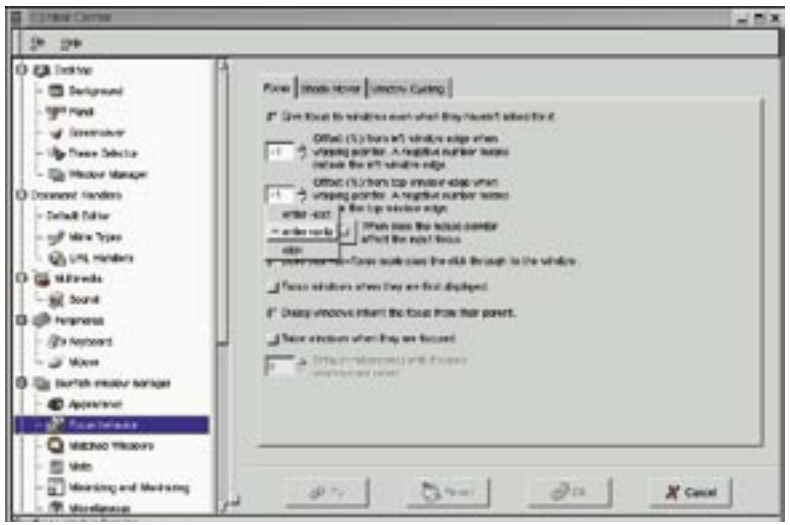
For applications that don't allow you to change the point at which you insert text, such as xterm or gnome-terminal, it doesn't matter where in the window you click the middle button — the text will just be pasted at the cursor.

For others where you can move the text cursor around, such as Netscape, you need to click where you want the text inserted.

Many applications also let you double-click to highlight a single word, or do a double-click-drag (press, release, press your mouse button quickly, then drag) to highlight a word at a time.

In a similar way, selecting a whole line or range of lines can be achieved by triple-clicking or performing a triple-click-drag. This might take a bit of practice, but it's rather useful.

If you have the gpm daemon loaded, this same functionality is available to you in full-screen console sessions. It can prove to be a real time saver when it comes down to editing configuration files at the console.



Experiment with the different focus methods for your desktop environment.

7 DIFFERENT FOCUS METHODS

The *focus method* refers to how the X window system determines which window you are typing in (the 'focused' window). Most window managers give you a choice of 'click-to-focus', 'point-to- focus' and 'sticky-focus'. The Sawfish window manager supplied with Red Hat Linux calls these 'click', 'enter-exit' and 'enter-only'. For KDE, see the Focus Policy option under Window Behaviour in the Control Center.

If you change the focus method to point-to-focus or sticky-focus then all you need to do to type in a window is to put the mouse cursor in it; no need to click. Some people find this less work if they switch between windows frequently. Experiment with the different options and see which methods you prefer.

6 KILLING X

Occasionally an X application may take control of your X server and prevent you from doing anything further; for example, you might still be able to move your mouse cursor, but find that clicking in any window has no effect at all.

In this sort of crash it is possible to kill your X server by pressing the magic sequence Alt-Ctrl-Backspace (similar to the 'three-finger salute', but using Backspace instead of Delete). This action will kill X immediately and violently (that is, it won't give X or its processes a chance to shut down and die gracefully).

Then you will either find yourself back at a text console (if you started X with startx), or your display manager will restart X (if you logged in at a graphical login prompt).

In some cases you might want the X server to ignore that key sequence. If so, add the line DontZap (for XFree86 3.x) or Option "DontZap" (for XFree86 4.x) to your /etc/X11/XF86Config or /etc/X11/XF86Config-4 file as appropriate.

This *explicit kill* feature also creates a core file that can be used for debugging. Generally, these can be found in your home directory and can be deleted to save space.

5 TAB COMPLETION

We covered the use of TAB in the very first Linux Pocketbook, but on the off-chance you missed this information, which *every* Linux user swears by, here it is again (and in a bit more depth).

We told you how handy the TAB key was to complete your commands on the command line. For example, to install the latest XMMS from an RPM file in your download directory you might type:

```
rpm -Uvh ~/downloads/xmms-1.2.4-somenumber.i586.rpm
```

Or, you could type:

```
rpm -Uvh ~/d[TAB]xmms[TAB]
```

The exit and logout commands can be used to exit from a shell or terminal, but pressing Crl-D has the same effect.

Or even, if it's the only file in the directory beginning with 'x', x[TAB]. If you want to be really lazy, you could even use TAB to complete the 'rpm' name, but chances are there are plenty of other programs on your system that start with 'rp'.

The wonders of TAB don't just extend to files, directories and commands in the current directory and in your path; TAB can also complete variable names, usernames and hostnames. Just precede the letters you want to search for completions with '$', '~' and '@' respectively. So for example to change to the /home/martigen directory we could use:

```
cd /home/martigen
```

or:

```
cd ~mart[TAB]
```

4 QUICKLY EXITING A SHELL

The exit and logout commands can be used to exit from a shell or terminal, but pressing Crl-D has exactly the same effect, and involves less typing. Ctrl-D will only work if the command line is blank, so if you've started typing a command, you can press Ctrl-C followed by Crl-D. Similarly, many command line programs support

exiting with Crl-D as well as their own exit commands.

And, if you're wondering, Ctrl-C is the universal 'quit' key sequence. All commands and programs run from the command line can be aborted with Ctrl-C. Very handy for aborting tasks you mis-typed or didn't realise would suck up so much CPU and disk time.

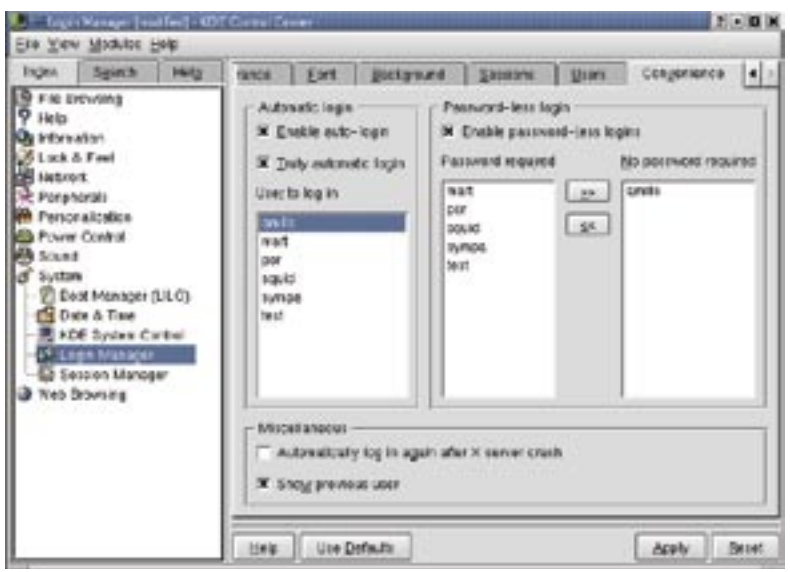
3 JOB CONTROL

We briefly covered the art of controlling running processes in *The Linux Pocketbook 2003 edition* with the bg command and the use of '&' following a program. Here's a little bit more information to guide you.

In addition to issuing a bg or fg command to background or foreground a process, you can also specify *which* process to operate on. In order to find out the number of tasks that you have running from your shell, use the jobs command.

For example, if jobs reports three processes, you can use fg %2 to bring the second process to the foreground.

What difference does this make? As a foreground process it'll respond to the shell; so, you can provide input if it's waiting for some, you can Ctrl-C to kill it or you can Ctrl-Z to suspend it.



Automatic logins make using Linux as a home operating system much smoother.

2 AUTO-LOGIN

This habit is *completely* insecure. It is, however, a very handy feature for a home system where you don't need to worry about other people getting access to your account — at least if you're using the latest KDE.

It's called auto-login. Under Control Center ⟶ System ⟶ Login Manager ⟶ Convenience you'll find two types of automatic login: passwordless logins and the full-blown boot-up-Linux-and-login-to-account feature. The latter is, effectively, like starting Windows — Linux will boot up and automatically log you in to the specified account.

Take note that some distributions also feature automatic logins for an account specified during installation.

1 REMOTE X

Since X is a network protocol, it is possible to send the output of applications running on one machine to the display of another. This is very cool and offers wonderful possibilities.

For example, say you have two machines on a LAN in your home, and one — a server Linux box — is acting as a file server, gateway machine et al and happens to include a CD burner. Because of the networking features of Linux and X, you can manage the server box remotely, and even burn CDs remotely. You can run a program like X-CD-Roast on the server box, but display it and use it over the network on your machine!

X makes it possible to send the output of applications running on one machine to the display of another.

From your end, it looks like the program is running locally. You can do this even if your machine happens to be a Windows client (we'll get onto this soon).

All the server box needs is a power cord and a network cable, with no other input or output peripherals attached. The functionality of Linux and X is such that a Linux machine can be *totally* administered remotely.

This is why administering Linux server farms is a simple affair — one monitor and one keyboard are needed on a single machine, then all the rest just need a network cable. This is something Windows machines can't provide, and Windows server farms often have a keyboard and monitor — or switch boxes — attached to each server.

How do you go about setting this up? For this example, we'll imagine two Linux boxes as described in the scenario above. We'll call the server 'Red' and the client machine 'Blue'.

By default, X is configured not to allow anyone else to display something from your machine, so the first step is to tell X on your machine that you want to accept connections from another host — in this case your server box, Red. We can do this as follows:


```
xhost +Red
```

Alternatively, if Red's IP address isn't defined in your /etc/hosts file you can specify the IP address. This command will allow X applications ('X clients') from Red to connect to the display which you are looking at on your machine (the 'X server').

Next, all you need to do is Telnet into Red and run a program — Netscape, for example — as usual. It should run on Red but display locally on Blue. If it doesn't, as can sometimes occur depending on the configuration of your system, one more command may be needed:

```
export DISPLAY=Blue:0
```

This tells any programs that use X to send their output to the first display ('0') on the machine called Blue. Again, Blue's IP address will need to be present in the /etc/hosts file on Red, or it can be specified directly in the above command.

You might also note from using the '--?' or '--help' switches with programs that many will support '--display'. This allows you to run a program and tell it where to display, regardless of any environment variables (such as the above). For example, you could Telnet into Red and then run:

```
xcdroast --display Blue:0
```

If you'd like to have this functionality always available, just add the xhost line to the end of your /etc/rc.local file.

Take note, however, that using xhost is not very secure because if, in our example, Red were actually a user's machine, xhost +Red would allow them to run applications on your desktop at will. For example, they could start a screensaver on your desktop, set a new background image, or run the xlock program to lock your display. (Not that this sort of thing occurred frequently at university or anything . . .)

But again, unless you grant access using xhost, X is quite secure. For a home LAN, this is fine. If security is an issue you need to consider, you can look at SSH (see 'Secure shell', in the Networking section), as this supports the X protocol.

Lastly, we mentioned earlier that you can also run Linux X programs over the network to display on Windows clients. How so? Easy, just grab a program that supports XFree86 clients for Windows. There are quite a few shareware programs available, and a quick Google search will be a great place to start looking.

Top 10 programs

Get acquainted with the must-have programs used by Linux gurus, and use them to advance further along the road to becoming a guru yourself!

10 SSH

We covered setting up an SSH server in the Networking section, but it's such a sexy program we have to include it in this list too. So here's a little more information about the SSH client.

As discussed earlier, SSH is a secure replacement for telnet, rsh and rlogin. Whereas many protocols (including Telnet) transfer everything over the network in clear-text, SSH encrypts all communications so that someone listening between two computers that are communicating cannot understand the information.

One of the great facets of SSH is that it is very easy to use, since all encryption is handled behind the scenes. SSH can even encrypt X Window sessions automatically. In fact it's easier to use SSH for this purpose than it is to use Telnet, because no xhost or export DISPLAY commands are needed.

SSH is often bundled with distributions, but you can grab it online at *www.openssh.com*.

Once it's installed, using SSH is simple. To Telnet to a host run:

```
ssh [username]@[remotehost]
```

If this is the first time you've connected to that host, you'll be told 'host key not found from the list of known hosts' and asked whether you want to add this host to your list. The reason for this is that SSH can ensure that you are connecting to the correct server, but it can't do so until it first gets the host key. Once logged in you can perform any task you normally would under Telnet, now safe in the knowledge that what you type and read can't be seen by anyone else.

For modem links it can help to enable SSH's inbuilt compression, and doing so is as simple as adding the '-C' switch. Similarly, you can enable the use of X over SSH with the '-X' switch.

9 TAIL

The tail command is a filter that processes data from files or other commands, and displays only the last 10 lines, or any value specified. It's an easy way to see what's new in a file rather than spending time and sys-

tem resources opening up a text editor, or cat-ing the program through more and browsing to the end. For example, to display the 10 most recent system log messages:

```
tail /var/log/messages
```

Or, along with ls, to display the 20 oldest files in a directory:

```
ls -lt |tail -20
```

This simple command also sports a feature which allows it to stay active and read files (and display information) as new information is added. This is a quick and easy way to monitor the output of log files in real time:

```
tail -f /var/log/messages
```

8 MC

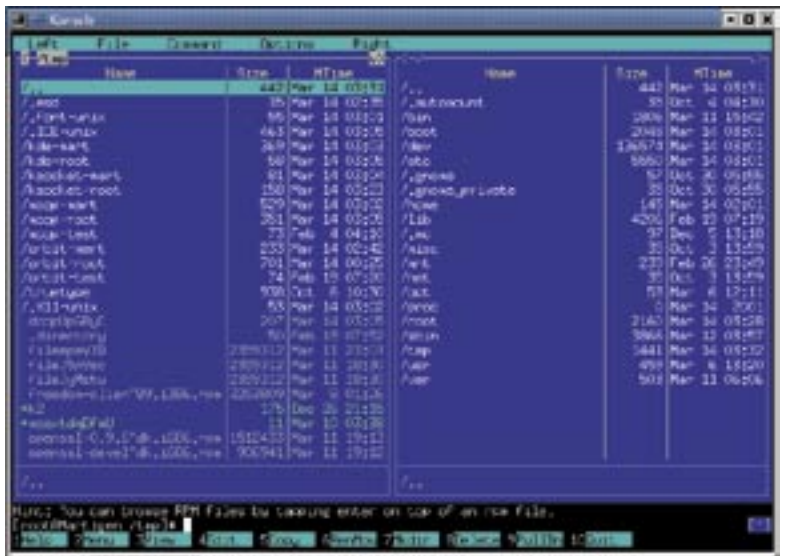
The Midnight Commander (MC) is a powerful, easy-to-use, feature-rich file manager that the editor of this Pocketbook thinks everyone should know and love. If you're a GUI fan who simply can't pull themselves away from a drag-and-drop file manipulation experience, well . . . give it a try anyway.

Midnight Commander is similar to the age-old Norton Commander and the downright fantastic X-Tree-Gold from the days of DOS. These console-based file managers use simple paned directory windows and shortcut keys to perform tasks such as copying, deleting, moving, viewing and editing. This speedy management is essentially the benefit of a program like MC over the Gnome or KDE default file managers (though, in actuality, Gnome's file manager is based on Midnight Commander, hence its name, gmc — Gnome Midnight Commander).

Midnight Commander sports many nifty features that even GUI file managers can't keep up with. MC can view inside and manipulate RPM and tar.gz files, and it can be used to directly connect to and view FTP and NFS servers as if they were local. It can perform system-wide searches for and in files, change file permissions and manage symbolic links. It even comes with its own file server system — mcserv — which MC clients can connect to over a network. If you're using Ext2, MC can even undelete your files.

7 RSYNC

The rsync program is an invaluable tool for copying files across a network. It can run much faster than traditional file copy tools due to its intelligent use of pipelining techniques and its ability to take advantage of files that are already present on the receiving end. Incidentally, rsync was written by Australian Andrew Tridgell of Samba fame as part of his PhD.



The popular, fast and fully featured file manager, Midnight Commander.

One of the most common uses of rsync is for performing backups from one computer to another. Another common use is helping to maintain Web sites; if you have a 'test' Web server and a 'production' Web server, then updating the production server from the test server once you are happy with the content can usually be done with a single rsync command.

The general format of the command is much like the cp command:

```
rsync [source] [destination]
```

You can, however, specify paths on remote hosts with a syntax such as 'host:/path'. To copy directories recursively and preserve file attributes you can use the '-a' switch.

Here's how you can copy the contents of /var/www to a remote host:

```
rsync -a /var/www/ [remotehost]:/var/www/
```

By default, rsync will login to the remote machine with rsh, but if you have SSH then you can tell rsync to use it with the '-e ssh' option. For more information about rsync, see the man page.

6 GNUPG

GnuPG is the GNU Privacy Guard, a free-software replacement for the popular PGP encryption software. GnuPG can be used to encrypt, decrypt and sign files for transmission over insecure media such as the Internet. Its most common use, apart from encrypting binary files, is to encrypt emails and other plain text communication.

To get started with GnuPG, the first step is to generate your private and public keys:

```
gpg --gen-key
```

The default kind and size of key are fine to use. Once the key is created, you can see your public key (to place it on your Web site, or send to other people) with the command:

```
gpg --export --armor
```

GnuPG can be used through the command line, but it is more often used indirectly through a mail client such as Mutt, Pine or the graphical mail client KMail.

For more information about GnuPG and encryption check out the GnuPG home page at *www.gnupg.org*.

5 VIM

VIM is an enhanced vi editor, which is supplied with most Linux distributions. vi has a reputation for being difficult to learn, and VIM has many features (such as extensive online help, multilevel undo, command line history and completion, and a GUI) that make it easier to use than the traditional vi.

Once you have reached a certain level of competence you will find that you are far more productive in VIM than in any other editor, with the possible exception of Emacs (the debating of which often generates more than a little heat).

For a tutorial on VIM, copy the VIM tutor file (/usr/share/vim/cdtutor) to your home directory, and run vim tutor.

Even if, after this tutorial, you find VIM not intuitive to use, don't worry — you're not the first to think this. Two facts remain, however: it really *is* an efficient text editor when you get used to it, and no matter how much geek knowledge you have you're not a real Linux guru until you've mastered — and regularly use — vi.

4 LYNX

Tired of waiting for chunky Web pages to load? Want to browse the Web when you're *not* using X? Welcome to Lynx!

Lynx is a text-mode-only, feature-rich Web browser. It comes in handy when you want to look for information quickly on Web sites; ignoring graphics allows download times to be extremely short. This is a great time saver for slow modem connections. Lynx is also ideal when all you have is a command line and you need to use the Internet, such as when you're performing maintenance locally or remotely over Telnet and you need to surf the Web to grab the latest updates or look up online information.

Using Lynx is a simple matter of passing it the URL (HTTP or FTP) you want to browse with lynx [URL], and using the cursor keys and Enter to navigate.

3 CRON AND AT

If you want to run a program unattended at a particular time, then you need either cron or at. The difference between the two is that cron lets you run programs at periodic intervals, whereas at lets you run a program once at a particular time. Either way, if any command that runs produces any output, it will be emailed to you.

To use cron, you need to create a 'crontab' entry (using crontab -e) which defines when you want your program to run. For example, if you have a backup script called /usr/local/sbin/run-backup which you want executed at 9pm every weekday, you would use this crontab entry:

```
0 21 * * 1-5 /usr/local/sbin/run-backup
```

The format of crontab entries is explained in the crontab(5) man page (man 5 crontab).

Using at is a little simpler. Type at now + 1 hour, and you'll see an at> prompt. You can then type the commands which you want to run in one hour's time followed by Ctrl-D to quit. Other time specifications are possible such as 'at noon tomorrow'; for details see the man page.

2 WGET

The wget program is a utility that can download files from HTTP or FTP servers non-interactively. This can be handy if you want to download a file from a script, but it can also be useful if you don't want to fire up Netscape just to download one file (simply run wget [url]).

In addition to being a method to quickly download files, wget supports recursive retrieval of files. When used on FTP servers it is able to read directory contents to download all the files contained within, and when used on HTTP servers, is able to follow links inside HTML documents. In other words, wget can mirror entire FTP and Web sites, and even be smart enough to change appropriate embed-

ded URLs to point to the local filesystem. So for example, if you mirror a Web site, any links that point to other files on the Web site will instead now correctly point to these files on the local filesystem. This behaviour is also very configurable. You can tell wget how many levels of links to follow ('-l' option), whether to follow links to other Web sites ('-H'), and much more.

For FTP and HTTP servers that support it, wget is also able to resume downloads from where they left off.

You can, of course, check the man page for more information.

1 TOP TIP! GREP IT, BABY!

One of the most useful and frequently used programs on *any* Linux system is grep. Its title comes from a classic Unix text editor (send us an email if you know which one) whose key sequence to perform a search was 'g/re/p' for 'global/regular expression/print'. grep performs a regular expression search *for* anything you specify *in* anything you specify. It can also search and replace information matching a pattern.

And grep is infinitely useful. You can use grep to search the contents of files or the output of other programs. For example, if you can't remember where you recorded the phone number of that guy from the party the other night, but you remember his name, try:

```
grep John *
```

grep will search all files in the current directory ('*') for 'John', displaying any lines that match.

More often than not, however, grep is party to other commands such as ps, find, locate, cat, and so on through the use of the handy pipe symbol. For example, the dmesg command displays a kernel log which can often run to hundreds of lines. If you want to see only the lines from dmesg which contain information about your IDE drives you can run:

```
dmesg |grep hda
```

and grep will pull out and display only those lines with 'hda' in them. This is why grep is so useful. Want to check the PID of a program in order to kill it, but running ps ax creates a long list that scrolls off screen? Just type ps ax |grep [process name]. Just installed the new official Nvidia driver and want to check it's being loaded by XFree86? Use cat /var/log/XFree86.0.log |grep nvidia. Can't remember that long make command you typed in a few days ago, but think it might still be in your shell's history? history |grep make. You name it, grep can do it!

Customising the shell

You can customise the shell with coloured prompts and have your favourite commands at your fingertips. And we'll show you how . . .

If you've ever piped the output of one command into the input of the other, or joined simple commands together to do complex tasks, then you have a taste for the power of the shell. As you use the shell more and more you'll discover how powerful it is and that, more often than not, the shell is quicker than the GUI.

In this section we'll show you how to customise the Bourne Again Shell (Bash). Bash is the default shell with most distributions and by far the most popular.

To customise Bash, we are going to edit two files, .bash_profile and .bashrc, both of which are found in your home directory. If you can't see them when you do a listing, try ls -la, and pipe it through more if the list scrolls offscreen.

When you login, .bash_profile is read and its commands executed. When you start an interactive shell that is not a login shell (say, by typing bash at the command prompt), another file .bashrc is read.

Typically, .bashrc is also read into a login shell, but this is not automatic. Rather it is done explicitly by a command in the .bash_profile file.

Don't be too worried about the distinction between these two files just yet. You'll learn how to use them as we go along.

SHELL VARIABLES

Shell variables (also called *environment variables*) are place holders for information and Bash maintains a suite of these to keep track of various settings. Have a look at what shell variables Bash knows about by typing the following:

```
env
```

One of those listed variables is PATH, which tells Bash in which directories to look for commands that you type. As you can see, the format of the variable is a list of directories separated by colons. When you run a command, Bash will search these directories in order to find your command. Why is this handy to know?

Say you installed some new programs to /usr/local/bin — which is a common place to store user space programs that won't get mixed up in the Linux distribution hierarchy. You might want to add this directory

to your PATH so you don't have to type /usr/local/bin/fancyprogram. You can do this as follows:

```
PATH=$PATH:/usr/local/bin
```

In fact, many distributions will add this to your user account PATHs automatically, so if you're logged in as your standard user account you might already see this directory in your PATH. Login as root, and then run env. Chances are you won't see /usr/local/bin in your PATH. This might explain all those times you've been root and found that running a program doesn't work — when you know it works under your normal account. The default root PATH is very different from the standard user account PATH.

What's the deal with the '$PATH'?, we hear you say. Glad you asked. Variables are referenced in the shell by their name, but preceded by '$' to clarify for Bash that you want to reference the variable rather than taking the word sliterally.

So, from our example above, $PATH represents (some say 'expands to') the current PATH as displayed by env, to which we are adding '/usr/local/bin'. Yes, it's pretty nifty that we're giving a new value to PATH (the 'PATH=' part) and referencing the current value of that variable at the same time. If you find this logic exciting, you should be a programmer!

So, now you know how to set a new environment variable from the command line. But how about making the change permanent? Simple, just add the line to your .bash_profile file.

By the way, if you want to add a new directory to your path that is searched *before* other directories you just add it, of course, in front like so:

```
PATH=/usr/local/bin:$PATH
```

To initialise any changes you can exit and login again or, alternatively, you can reload it with:

```
. .bash_profile
```

SETTING YOUR PROMPT

An environment variable of much greater interest is the one that sets your prompt, PS1. Let's start by setting the prompt to a *string* (programming lingo for 'collection of ASCII characters'). Here's a scary example:

```
PS1="C:\>"
```

Try it and see what your prompt looks like.

Configured like this, PS1 is a local shell variable. If you start another shell by typing bash then the prompt in the new shell will be as it normally is. The new shell does not get a copy of this shell variable, because you haven't told Bash to export it. If you want 'child' shells (scripts and the like) to inherit a variable you have to use the export command:

```
PS1="C:\>"
export PS1
```

Now if you run a new shell, the prompt will be the same as in the parent shell.

Hard coding the prompt to a string of your choice may pander to your delusions — "I await your command, oh Leader >" — but it's not very useful. Fortunately, Bash provides you with an assortment of special characters that can appear in the prompt and take on a special meaning (see Table 1). For example:

```
PS1="\W>"
```

The '\W' value in a prompt prints the basename of the current directory. If you are currently in the directory /home/work then the prompt looks like this:

```
work>
```

If you want to see the values that are used to make up the prompt you are familiar with on your distribution, open up a new terminal (and thus a new shell with the original variables) and type:

```
env |grep PS1
```

Yes, grep allows us to pick out just what we want to see. We said you'd learn to love this command!

Another neat trick you can do with prompts is to add colour using ANSI escape sequences. An escape sequence is a little signifier that tells the interpreter (in this case, ANSI) that what follows should be interpreted and not ignored.

Table 2 shows the colours available. These sequences look like gibberish, but the terminal — if it supports ANSI escape sequences — understands them. Table 1 shows that Bash allows you to embed non-printing characters into the prompt. These begin with '\[' and end with '\]'. Table 1 also shows that '\xxx' represents the character whose octal value is 'xxx'. The octal value 033 corresponds with the ANSI escape character and what follows are the magic incantations of Table 2 to turn on a colour. So, if you want your standard prompt to be purple, use the following:

```
export PS1="\[\033[0;35m\]\u:\W>\
[\033[0;0m\]"
```

Remember to turn off colour at the end of the prompt ('[\033[0;0m\]') so 'normal services are resumed'; otherwise the prompt colour will spill out on to what you type.

Once you've experimented and settled on a prompt that displays the information you want (see Table 1) in the colours you want (Table 2), make it permanent by adding the command to your .bash_profile file.

ALIASES

If you're a fast typer you might find yourself quickly typing cd.. in order to change to a parent directory instead of cd .. — if you're getting tired of seeing the bash: cd..: command not found error, try defining a new alias to take care of it:

```
alias cd..="cd .."
```

How does this work exactly? Whenever Bash sees the string cd.. starting a line, it recognises it as an alias and substitutes the string cd .. for it. Et voila! No more error messages. Aliases should be placed in your .bashrc file, because this is loaded whenever you start an interactive shell. If you cat your .bashrc file now, you'll probably see some aliases already set up by your distribution.

Alternatively, issuing the alias command alone will display aliases configured by .bashrc in addition to any system-wide aliases defined by your system administrator. If that's you, and you'd like to find out where these other aliases are being defined, take a look in the /etc/profile.d directory. Other global Bash settings are configured in /etc/bashrc and /etc/profile.

A couple of aliases intended to protect you may be predefined. To prevent you from crunching your files, you'll often see rm aliased to rm -i and mv to mv -i. The '-i' option to each of these commands causes the commands to prompt you for confirmation whenever you are going to overwrite or otherwise cause the demise of a file. If you don't want to be prompted for confirmation when deleting files, comment out these aliases or remove them at the command line by typing:

```
unalias rm
```

SHELL OPTIONS

You can toggle the behaviour of the shell by setting and unsetting various shell options using the shopt command. To see what options are set for you, type shopt.

One option which is handy to set and not enabled by default is the 'cdspell' option. Set it using:

```
shopt -s cdspell
```

This allows the shell to correct minor typos in the name of a directory when using the cd command. So, for example, if you have the directory 'new' and you type in cd enw, Bash will work out that you want to change to the 'new' directory.

Put any shopt commands into your .bashrc file.

You can find out other options by wading through the man page for Bash, or by having a read of the Bash Reference Manual (online at *www.gnu.org/manual/bash*). The Manual will also explain how to set other options using the set command and how to configure the command-editing commands on the command line, as well as thoroughly describing the Bash shell.

If you want to see how other people have configured Bash, have a look at a great collection of dot files at *www.dotfiles.com*.

CONTROL CHARACTERS AND COLOUR SEQUENCES

TABLE 1

\a	A bell character
\d	The date: 'Weekday Month Date'
\e	An escape character
\h	The hostname, up to the first '.'
\H	The hostname
\n	A newline
\r	A carriage return
\s	The name of the shell
\t	The time in 24-hour HH:MM:SS format
\T	The time in 12-hour HH:MM:SS format
\@	The time in 12-hour am/pm format
\u	Username
\v	Bash version
\V	Bash release number
\w	Current work directory
\W	Basename of the current work directory
\!	History number of the current command
\#	Command number of the current command
\$	# for uid=0, else $
\xxx	The character whose octal code is xxx
\\	A backslash
\[	Begin an escape sequence
\]	End an escape sequence

TABLE 2

Black	\[\033[0;30m\]
Red	\[\033[0;31m\]
Green	\[\033[0;32m\]
Brown	\[\033[0;33m\]
Blue	\[\033[0;34m\]
Purple	\[\033[0;35m\]
Cyan	\[\033[0;36m\]
Light grey	\[\033[0;37m\]
Grey	\[\033[1;30m\]
Light red	\[\033[1;31m\]
Light green	\[\033[1;32m\]
Light yellow	\[\033[1;33m\]
Light blue	\[\033[1;34m\]
Light purple	\[\033[1;35m\]
Light cyan	\[\033[1;36m\]
White	\[\033[1;37m\]
Terminal default	\[\033[0;0m\]

Filesystems and drives

In the pursuit of speed the first aspects of your system that you should take a look at are the hard drive and the filesystem.

The hard drive is currently, and always has been, the bottleneck of your system. Regardless of how fast it spins and the speed and size of its bus, a mechanical device such as the hard drive can't possibly match the efficiency of RAM. If non-volatile RAM were cheap, we'd all have 30G NVRAM systems, and booting and loading programs would take but a second.

This not being the case, we can at least work with what we've got and maximise our hardware by looking at two facets of the storage subsystem — the hard drive and the filesystem.

RAID

Once the domain of expensive server systems, the use of RAID (Redundant Array of Independent Disks) configurations is becoming more accessible to the average user, and is thus more widespread. Originally designed as a method to make use of aging disks, RAID has developed into a reliable and quick means for data storage, not only for server systems but now also for the desktop. Hardware IDE RAID solutions can be found built into many modern motherboards, and plug-in SCSI RAID cards have been available for years. With today's super-fast CPUs, software-based RAID is now a viable and significantly cheaper method of obtaining the speed and information protection that RAID offers.

For those new to the technology here's a quick summary: Two (hard drive) heads are better than one. A RAID 'array' is a collection of hard drives that act as one drive. Depending on the type of RAID implementation, this provides a significant speed boost, protection from data loss, or both.

RAID arrays that focus on data 'redundancy' are able to lose a hard drive — or even multiple hard drives — due to hardware failure, without losing *any* data on the combined volume. RAID arrays built for speed — those that literally *stripe* the data across all the drives as if they were one single drive — can significantly improve data transfer rates for each drive added to the array.

If you're any sort of performance geek you're either already using RAID, or you want to be. The good news is that Linux not only supports hardware RAID adapters, it also offers software-based RAID that is so efficient you can even use it on low-end systems. And setting it up is easy.

FILESYSTEMS

Before we get onto setting up RAID lets look at the other player in the storage subsystem game.

The filesystem determines how information is written to your drive. Filesystems are responsible for providing features such as security, data recovery and, of course, performance. If you've been a geek since the days of DOS, you've no doubt noticed that DOS and Windows 95 and its successors seem to have considerably faster disk access than Windows NT and Windows 2000. The FAT filesystem used by DOS and Windows 95 et al is a very simple filesystem which sports the following: it's terribly featureless, and it's fast. By contrast, the Windows NT/2000/XP filesystem NTFS provides all manner of features such as security and data recovery, but is clunky and slow. This is the standard payoff between features and speed.

A journaling filesystem keeps a journal of access requests, written before the requests themselves are carried out.

The Linux Ext2 filesystem has many features similar to NTFS, and is slightly faster. FAT remains, however, the fastest filesystem in use today — at least for small to medium-sized drives. After a certain limit (supposedly around 8G, but effectively closer to 4G) FAT starts to slow down rapidly — it's not designed for large filesystems.

Ext2 has been the standard Linux filesystem for some time, but it's not the only one available. One feature that many high-end Unix systems have is a *journaling* filesystem. Without going into the computer science theories behind filesystems too much, a journaling filesystem is one which keeps a journal of access requests that's written before the requests themselves are carried out. Why is this handy? Have you ever turned off Linux without shutting down and been forced to sit through an Ext2 filesystem check that's taken minutes to complete (or considerably longer for large drives and slow systems)? Wait no more; with a journaling filesystem a sudden crash or reboot can be recovered from in seconds.

During reboot, if the filesystem detects that it wasn't shut down properly, it will 'replay' the journal and carry out any requests it didn't have a chance to complete. This takes less than a few seconds, after which the integrity of the filesystem is restored. Moreover, this journaling feature means that it's very rare to lose any information from a sudden shutdown. And, if that weren't enough to convince you to install a new filesystem, journaling filesystems are also quite fast (faster, at least, than Ext2).

As it happens, Mandrake 7.2 and later versions come with ReiserFS, a popular journaling filesystem for Linux which is still under development. There are a few other Linux-based journaling filesystems on the horizon, including Ext3, but ReiserFS is the most complete at this point in time. And, as with the 2.4.1 kernel, ReiserFS support is now included in the kernel.

RECONFIGURING YOUR SYSTEM FOR SPEED

Given that setting up both software RAID and ReiserFS requires repartitioning and reformatting (yes, this means reinstalling Linux), we'll show you how to set up RAID and ReiserFS at the same time, and reap the performance benefits of both. If it's all too much hassle at this point, keep it in mind the next time you install Linux. If you're a performance freak and you're eager to do this now, backup your data and get out your distribution CDs!

You can, of course, set up RAID and ReiserFS for data drives separately from your main Linux installation. If you want to do this it's just a matter of following these instructions for whatever partitions you want to use without actually reinstalling Linux.

We'll be using Mandrake for our example setup since it not only comes with ReiserFS but also DiskDrake, which makes setting up a RAID partition easy.

SOFTWARE RAID

Hardware RAID adapters are configured through the adapter itself, and if you happen to have one of these then the chances are you already know how to use it. So here we'll cover the installation of software RAID for an average home user or server system. We'll also focus on RAID-0, the fastest implementation, as this is the most common setup. If you want to go all out for data redundancy, RAID-5 is just as easy to configure; follow these steps and make the changes where appropriate.

First, in case it isn't already apparent, you're going to need more than one hard drive. If you already have two or more drives you're ready for the next step. If you don't, well, you can read on but you'll only be teasing yourself until you get an extra hard drive.

PARTITIONING

For our example we'll assume two 4G drives with the following configuration:

Drive 1:	First 2G	Windows	FAT32
Drive 1:	Second 2G	Linux RAID 0 disk 1	ReiserFS
Drive 2:	First 2G	Shared data drive	FAT32
Drive 2:	Second 2G	Linux RAID 0 disk 2	ReiserFS

Yep, this means Windows has 2G of storage and Linux 4G, with a 2G shared data drive. We're keeping the partitions set aside for RAID the same size as this makes for an efficient array. Hardware and some software options demand same-size drives, but for Linux RAID you can in fact mix different-size partitions — but any space on one drive that isn't matched on another simply won't be RAIDed.

What about swap? Well, you can keep that on a single separate partition, or leave aside 128M on each of the Linux RAID partitions above and create a second 256M array for a RAID swap.

It also gets a little more complicated than this: though it *is* possible to boot a software RAID array, it's a little tricky unless you are running a 2.4 kernel. Given that Mandrake 7.2 comes with a 2.2 kernel, we will keep it simple here and will create a small Ext2 /boot partition for Linux to load kernels from, to keep the instructions backwards-compatible.

Don't worry, speed freaks, this won't detract from performance — /boot is used for looking up the system map file and loading your kernel into memory. Since this is normally between 500K and 900K it won't gain a visible speed boost for being on a RAID array.

If you're with us so far, backup any essential information, throw in your Mandrake Linux installation disk, and reboot.

SETTING UP RAID AND REISERFS

1 STEP ONE

For installation type choose Expert. This gives you access to the fun stuff.

2 STEP TWO

When it comes to partitioning, choose DiskDrake.

3 STEP THREE

If you haven't done so already, resize and delete partitions until you have enough equal-sized free space on each drive for your RAID-0 root (/) array. Create enough space for the two RAIDed swap partitions as well. Note that in our example layout above we specified to use the last 2G of each drive for Linux; as well as having the same size for each of the RAID-0 partitions, it also helps to have the same location on the drive.

The best RAID setup that you could have would be two identical model hard drives with your Linux RAID partitions set at the same boundaries on each drive.

4 STEP FOUR

Create an Ext2 partition of around 20M on the first drive and set the mount point to '/boot'.

Given some machines have problems seeing extended partitions on large hard drives, it's actually a good idea to have a small, separate boot partition below the 1024 cylinder boundary anyway. Hopefully the drive size will be such that you have this 20M lying around, but if not, take it out of the space you allocate to your Windows partition; we need to keep the Linux partitions of equal size for the RAID array.

5 STEP FIVE

Click on the empty space you want to allocate to the first RAID-0 array and select 'Add to RAID'. It will be called 'md0' by default (meaning 'multiple disk'). Leave it on the default 'RAID 0' for 'Level', and leave the chunksize on '64k'. The chunksize is the size of the *stripe* across the two drives. A smaller value increases performance for smaller files and a larger value increases performance for larger files — 64K is a good default. If you want to select a higher RAID level for data redundancy you can select RAID-1 to create mirrored drives. This gives maximum redundancy and no speed increase (actually, a decrease for writing). RAID levels above 1 mix speed and redundancy in different amounts and all require at least three hard drives. Hence, we'll leave them alone for our example.

Select the equal-sized empty space on the second drive and select 'Add to RAID', choosing md0 for the target.

Do the same for your RAID swap partitions, creating 'md1'.

6 STEP SIX

Click on the new RAID tab in DiskDrake, click on the RAID array's free space and click on Type. Scroll down the list and choose ReiserFS. For md1, do the same but select Linux Swap.

7 STEP SEVEN

Set the mount point for md0 to '/'. After all is said and done, you might end up with a layout like this (given our example 4G drives):

Drive 1:	First 2G	Windows		FAT32
Drive 1:	20M	Linux	/boot	Ext2
Drive 1:	Second 2G	Linux RAID 0 array 1 partition 1	/	ReiserFS
Drive 1:	Last 128M	Linux RAID 0 array 2 partition 1	swap	N/A
Drive 2:	First 2G	Shared data drive		FAT32
Drive 2:	Second 2G	Linux RAID 0 array 1 partition 2	/	ReiserFS
Drive 2:	Last 128M	Linux RAID 0 array 2 partition 2	swap	N/A

8 STEP EIGHT

The next step is to install Linux as normal and Mandrake will take care of the rest. For future versions of Mandrake or any other distribution that allows you to create RAID and ReiserFS partitions during install, these instructions will apply.

What kind of performance difference can you expect? It depends, of course, on your hardware, but to give an indication of the difference RAID makes, here's the output of hdparm's speed test commands on the Pocketbook Editor's machine. This is a PIII 560 (overclocked 450) with two equal-sized, 7,200rpm, Wide SCSI drives. The first test shows the speed of a single drive (/dev/sda), while the second shows the speed of the RAID-0 array (/dev/md0).

```
[root@Martigen /root]# hdparm -Tt /dev/sda

/dev/sda:
 Timing buffer-cache reads:   128 MB in  0.94 seconds =136.17 MB/sec
 Timing buffered disk reads:  64 MB in  3.40 seconds = 18.82 MB/sec

[root@Martigen /root]# hdparm -Tt /dev/md0

/dev/md0:
 Timing buffer-cache reads:   128 MB in  0.85 seconds =150.59 MB/sec
 Timing buffered disk reads:  64 MB in  2.34 seconds = 27.35 MB/sec
```

Not a bad speed increase, given it didn't require the purchase of extra hardware!

FURTHER INFORMATION

To check if the md driver is being loaded smoothly, run dmesg |grep md**. To check the status of your arrays, run** cat /proc/mdstat**. Also, to squeeze even more performance out of your storage subsystem, be sure to try the** hdparm **speed tweaks in the next section.**

Optimisation tips

If you're like most Linux users you want the latest and the best, and this includes optimising the performance of your hardware and Linux system. Here are a few tips to get you started.

MEASURING

To ensure that any changes you make actually have a positive effect, it helps to be able to measure performance differences. The following list of programs will help.

- **The two best-known process and memory measurement tools are** ps **and** top**.**
- **The** vmstat **utility is able to produce a scrolling log of various system parameters. Run it as** vmstat 5 **and the list will be updated every five seconds.**
- hdparm**, covered on the next page, lets you measure the speed of your disk subsystem.**
- **Two other popular hard drive benchmarking programs are Bonnie and XHDBench, both of which can be found via *www.freshmeat.net*.**
- **To benchmark the speed of X, try** xbench **and** x11perf**, which are probably on your system.**
- **Apache includes a program called** ab **— the Apache HTTP server benchmarking tool — that lets you test the speed of your Web server. For example, you can use it to measure speed differences if you make changes to your Web server configuration.**
- **Lastly, the classic test of CPU, RAM and disk subsystem is the kernel compilation test. Time how long it takes to compile a kernel before and after any changes you make.**

HARDWARE TIPS

- **In general, the single biggest overall system speedup is obtained with more memory. Linux uses as much memory as possible for a dynamic disk cache, and since memory is so much faster than disk, extra memory can make a dramatic difference to performance.**
- **If you have two IDE drives in use under Linux, place them on separate controllers if possible. It is a limitation of the IDE/ATAPI protocol that simultaneous read/writes cannot be done to two different drives on the same controller.**
- **If you have multiple physical hard drives, configure them in RAID formation (see previous section).**
- **Disable Advanced Power Management in your computer's BIOS.**
- **Try out Powertweak (*linux.powertweak.com* to optimise your CPU and chipset configuration settings.**

DISK PERFORMANCE

The methods that Linux uses to access IDE hard disks are by default quite conservative, so that Linux will run reliably on nearly any IDE hardware. Of course, the trade-off for this is speed. The hdparm **utility lets you adjust disk parameters, but you need to do so with caution, as setting parameters which are not supported by your hardware can result in filesystem damage. Try each of the following commands and test your drives after each one to make sure they are accepted by your hardware:**

hdparm -c3 [drive]	**to turn on 32-bit I/O support**
hdparm -u1 [drive]	**to unmask interrupts; can improve the performance of the rest of the system during heavy disk activity**
hdparm -m16 [drive]	**to allow the transfer of multiple sectors per interrupt**
hdparm -d1 -X34 [drive]	**to turn on multiword DMA mode 2 transfers**
hdparm -d1 -X66 [drive]	**to turn on UltraDMA mode 2 transfers**

To test the speed of your drives after each of these commands run:

```
hdparm -Tt [drive]
```

Once you've found the appropriate settings for your hardware, you can place all the switches into one command line and add the command to /etc/rc.d/rc.local**. None of these settings are preserved across a reboot, so they need to be set every time in a startup script.**

FILESYSTEM PERFORMANCE

- **As we've explained in the previous section, the new ReiserFS filesystem is a lot faster than the standard Linux Ext2 filesystem.**
- **Whenever you read a file, Linux needs to update a timestamp on that file called the 'atime' (last access time). As this information is rarely used, there is an option to turn it off called 'noatime'. You can enable this option by adding the word 'noatime' to the** /etc/fstab **mount options for all your Linux partitions. For example:**

```
/dev/hda / ext2 defaults,noatime 1 1
```

The changes will take effect automatically on your next boot, but to make them work immediately, run this command for each of your partitions:

```
mount -o remount [mount point]
```

FILESYSTEM PERFORMANCE (CONTINUED)

To test whether it worked, type cat /proc/mounts**; you should see 'noatime' mentioned in the list of options.**

- **During installation, put your swap partitions near the beginning of your drive. This area is physically located on the outer portion of the disk, so the read/write heads can cover much more ground per revolution.**
- **What about defragging you say? After all, you do this all the time under Windows and it helps a little with filesystem performance. The Ext2 filesystem is pretty good at avoiding fragment ation, but despite this a defragmenter is available: Just do a search on *www.freshmeat.net*. Be aware, however, that it is quite old and only works on small to medium-sized filesystems. If you're using a more advanced filesystem like ReiserFS, you certainly don't need to worry about fragmentation!**

GENERAL TIPS

- **If you haven't yet ventured into compiling your own kernel, take the plunge! Tailoring the kernel to your system allows you to create a sleek, compact kernel that's faster and uses less memory.**
- **As discussed in *The Linux Pocketbook 2003 edition*, to speed up booting and general system performance, turn off any services you don't require, such as network daemons. If you require some services only occasionally and you are short of memory, you might want to start these only when required.**
- **As cool as Gnome and KDE are, these desktop environments use more memory than their simpler predecessors. If you can live without the fancy user interface, you can try using X with just a lightweight window manager instead. There is information about a number of different window managers at *www.plig.org/xwinman*.**
- **To speed up X performance, try using a different colour depth. If your applications don't require many colours, using a 256-colour (8-bit) mode should make display updates faster. If you like more colours, stick with 16-bit; unless you're an artist you won't notice a difference over 32-bit, but your 2D desktop will render faster.**

Developing Linux

Shell scripting

The shell is the single most powerful tool in Linux and scripting is the incantation that unleashes its power. Following is a tutorial on the basic elements of scripting the Bash shell.

To the uninitiated, the shell is an archaic throwback to the days before the GUI. The initiated know better. If you need to find files exactly three days old containing the text 'All your base are belong to us!' and sort the resulting files from biggest to smallest, then you need the shell (try doing *that* with a mouse).

But that's just the beginning. How about finding files three days old containing the text 'Linux rules!', moving them to a directory called 'Propaganda' and then being mailed to say the task is complete and to report the number of files moved? For that, you'll need a shell script.

Scripting is programming. It's about using the commands you already know — cd and such like — and stringing them together as a series of commands in a file. But unlike C++ and other programming languages, scripting is easy to learn. If you're interested in basic scripting and all the possibilities it can open up, this section is for you. This introduction assumes you have zilch programming experience. Despite this we will move quickly, so hold on to your hat!

We are going to be scripting with the GNU Bash shell, which is probably the interactive shell that you use as it's the default on most systems. But it doesn't matter which shell you normally use, you can still run Bash scripts, since scripts run in their own shell. We'll say that again: scripts run in their own shell. It's a subtle point but keeping it in mind will save you trouble later. For now, let's start scripting.

YOUR FIRST SCRIPT

Let's skip the preliminaries and jump straight into a script. If you're always running out of disk space at work or at home and find yourself hunting around for the big files to delete, then we have the script for you. A very simple script to find these disk hogs is created by typing the following command into a file — we'll call it 'findhogs':

```
du -ak | sort -nr | head -10
```

The command du -ak reports file space usage of files and directories in kilobytes, sort -nr sorts the result in reverse numerical order and head lists the top 10 lines from the resulting output. To run this script, type the following:

```
bash findhogs
```

HASH-BANG-BIN-BASH

This is a useful little command, but it's going to be tiresome to precede the command with the word 'bash' every time. If you add the following line:

```
#!/bin/bash
```

to the top of the file then this specifies which shell the script is to be run in. So, the complete script looks like:

```
#!/bin/bash
du -ak | sort -nr | head -10
```

Before the new script can run, you need to make the file executable. Do this using the chmod command:

```
chmod u+x findhogs
```

and the script can be run by typing:

```
./findhogs
```

Now that you've written your script it can be executed from the command line like any other command. If you place it in a directory in your path you can run it from anywhere (but until you do, you'll need to run it with './').

But what should you call your script? Findhogs? FindBiggest? The name you choose is your business, but you don't want it to conflict with a name already used by your system, or you may get interesting results.

In order to see find out if a program called 'findhogs' is already known to your system, try the following commands:

```
whereis findhogs
which findhogs
alias findhogs
man findhogs
```

None of these commands found a 'findhogs' (an enigmatic silence from whereis indicates that no command of that name was found), so it's safe to use 'findhogs' as a script name.

SHELL VARIABLES

There is an annoying restriction with the findhogs script — it only tells you the largest files and directories in the *current* directory. If you want to find the biggest files in another directory, you have to change to that directory and run the script there. What you need is a means of telling findhogs which directory to report on. You can do this using shell variables.

What you want to do is to be able to type something like:

```
findhogs ~/Data/Images
```

That is, tell findhogs to report on the directory ~/Data/Images. Here is a script which does this:


```
#!/bin/bash
du -ak $1 | sort -nr | head -10
```

The new addition of '$1' is a shell variable (which you learned in the previous chapter). Here '$1' represents the first argument on the command line. In our example the shell substitutes '$1' with '~/Data/Images' before executing any of the commands in the script. Now the script reports on the directory you tell it to. If you don't specify any directory, it will happily run in the current directory as before, because there'll be no information to substitute for '$1'.

What if you want to create a script that accepts more than one argument? The first argument is represented by $1, so the second will be represented by $2, the third by $3, and so on. You could modify the script, for the purposes of example, to make it possible to specify the number of files and directories reported:

```
#!/bin/bash
du -ak $1 | sort -nr | head -$2
```

So, you can check the number of big files in the /tmp directory, and ask to see only the top five, by running:

```
findhogs /tmp 5
```

If you don't specify a value, head will default to 10 lines regardless. This might be getting fun, but those of you running on caffeine will probably notice that the script isn't as easy to use as before. Because the $1 represents the first argument following a command, you can't specify the number of files to report unless you also specify a target directory. If you only supply one argument to the script, it will always be taken as the directory to search. Hence, if you want to search the current directory but display the top 20 files, you would need to run the new script as follows:

```
findhogs . 20
```

Which is why, at least for this script, it's enough to use a single argument that allows you to specify a target directory.

In addition to the sequential program arguments represented by variables, other built-in shell variables are listed in Table 1 at the end of this section.

USER-DEFINED SHELL VARIABLES

In addition to the built-in shell variables you can also create your own. The name of a shell variable can be any sequence of letters and numbers as long as the first character is a letter. For example, 'TIME', 'name2', 'Program_Dir' and 'a255' are all valid variable names. You can then assign values to these variables as follows:

```
name=Jimmy
echo $name
```

Here you are creating the variable 'name' and giving it the value 'Jimmy'. Note that when creating or assigning a variable, you do not use the '$' prefix. The '$' prefix tells the shell to substitute the *value* of the variable. Also, be careful when defining shell variables to ensure that there is no whitespace (spaces and tabs) around the = sign. The bash command echo here simply prints a variable to the default output device (usually your screen).

If you've already been exploring shell scripts, you may have seen braces, {}, used to surround shell variables. These have the purpose of preserving the variable despite the surrounding characters:

```
base=cat
echo ${base}egory
echo ${base}amaran
```

will give you the output:

```
category
catamaran
```

The shell variables that we have considered so far are all strings — they are just a collection of characters. The script below may not produce the result you expect:

```
#!/bin/bash
X=1
X=X+1
echo $X
```

Expecting to see '2'? To treat strings as numbers, you have to use the inbuilt let command:

```
#!/bin/bash
X=1
let X=X+1
echo $X
```

CONDITIONAL COMMANDS

So far, the scripts have just been a sequence of commands one after the other. It's time to make a decision — or at least it's time your scripts made a decision. The following script, countargs, reports the number of arguments passed to it:

```
#!/bin/bash
if test $# = 0
then
    echo "ERROR: You must supply at least one argument"
else
    echo "You supplied $# arguments"
fi
```

The end of an if-then-else decision block is marked by the word 'fi' (yes, that's 'if' backwards). If you execute:

```
./countargs
```

you can see what happens by walking through the script. The built-in parameter '$#' is a count of all the arguments passed to the script (see Table 2). In our case, this will be zero, so the condition being tested, '$# = 0', is true. Line four is executed, and line six (the 'else' part) is skipped. If, instead, you execute the command:

```
./countargs linux is the best
```

then '$#' is four, the condition (line two) tests false, and the else part of

the conditional statement is executed. It is, however, not necessary to have an 'else' branch to your statement.

A synonym for the keyword test is the [] notation. You can say if [$# = 0] to mean the same as if test $# = 0.

As with shell variable assignment, be attentive to the use of whitespace. There must be whitespace around the brackets, and around the test condition. All of [$# =0], [$#=0] and [$# = 0] won't work. This is the exact opposite of what you learned before about using whitespace when a variable is being assigned a value, so be careful.

The final variation on the theme is to use an elif statement (meaning 'else if') as in the following snippet:

```
if [ $# -eq 0 ]
then
    echo "No arguments"
    echo "Usage : somescript arg1
[arg2]"
elif [ $# -gt 2 ]
then
    echo "Too many arguments"
    echo "Usage: somescript arg1
[arg2]"
else
    echo "Running program"
    echo "Whirr chunk chunk"
fi
```

The '-eq' setting means 'equal to' while '-gt' means 'greater than'. There are similar options for less than, less than and equal to, greater than and equal to, and so on. In this way you can be more precise about the information you are testing for.

If-then-else type statements only work for binary decisions; is it black or is it white? But what if it's off-white? Enter the case statement, which has the form:

```
case test_pattern in
    pattern1)
        statement
        ;;
    pattern2)
        command
        ;;
esac
```

Note that the case statement ends with an 'esac' ('case' backwards). The 'case' attempts to match 'test_pattern' with 'pattern1'. If it doesn't match, it moves on to 'pattern2', and so on, for as many patterns as are listed. If no pattern matches, then it does nothing. The example below uses the '$SHELL' predefined shell variable to report which shell you logged into:

```
#!/bin/bash
case $SHELL in
    /bin/tcsh)
        echo "Your login shell is the
C-shell"
        ;;
    /bin/bash)
        echo "Your login shell is
Bash"
        ;;
    *)
        echo "I have no idea what
shell this is"
        ;;
esac
```

The '*' matches anything and so acts as a catchall once the other conditions have been tested.

ITERATION

Iteration, or looping, commands allow you to repeat the execution of a group of commands.

We'll begin our exploration with a 'while loop' that counts any number of seconds specified on the command line:

```
#!/bin/bash
COUNT=0
while [ $COUNT -lt $1 ]
do
    let COUNT=COUNT+1
    sleep 1
    echo $COUNT
done
```

The while loop repeats everything between the 'do' and 'done' *while* the test condition tests true, and then exits the loop. In this case, the loop repeats while 'COUNT' is less than ('-lt') the number passed on the command line, '$1'.

The 'until loop' complements the while loop, performing its body *until* the condition is true. The script above, therefore, would be written as follows:

```
#!/bin/bash
COUNT=0
until [ $COUNT -ge $1 ]
do
    let COUNT=COUNT+1
    sleep 1
    echo $COUNT
done
```

Here, you're looping until the COUNT is greater than or equal to ('-ge') $1, at which stage the loop will exit. To show that looping is not limited to numbers, have a look at this little script which lets you know that your coworker Kali has just logged on:

```
#!/bin/bash
until who | grep kali
do
    sleep 60
done
echo Kali has logged on
```

This uses the exit value of 'who | grep kali'. If grep doesn't find a match, the command exits with a 1 (false), and the body of the loop is performed (which is to wait a minute).

If it finds a match then it returns a 0 (true) and the loop ends, which causes the script to run the next statement after the loop, which in the case of our script is the echo command.

Another loop structure is the 'for loop'. The for loop works its way through a list, executing its body once for each item in the list and terminating when it reaches the end of that list.

```
#!/bin/bash
for PARAM in apple banana orange
do
    echo $PARAM
```


```
done
```

The output from this script is:

```
apple
banana
orange
```

Let's look at an another script which is a little more practical. The for loop is particularly useful for working its way through wildcard matching from the command line. The script backup uses mv to rename files to .bak:

```
#!/bin/bash
for file in $*
do
    echo $file renamed as $file.bak
    mv $file $file.bak
done
```

The use of a for loop to run through command line arguments is so common that it is the default behaviour when you leave off the 'in $*' statement. Here is a script that finds and displays all the files in the current directory on the command line that exist but are not regular files.

```
#!/bin/bash
for name
do
    if [ -e $name ]
    then
        if [ ! -f $name ]
        then
            echo $name
        fi
    fi
done
```

The first 'if' statement tests if the filename or pattern passed as an argument to the script exists ('[-e $name]') and the second tests if they are not regular files ('[! -f $name]'). The '-f' is the option that checks if a file is regular (that is, not a directory, etc) while the operator '!' represents 'not'. Hence, the condition 'if [! -f $name]' says 'If the file is not a regular file'. Without the '!' the statement it would mean 'If the file is a regular file'.

TABLE 1: BUILT-IN SHELL VARIABLES

$*	**All of the script arguments starting from the first**
$#	**The number of positional parameters**
$?	**Return value of the last command**
$-	**Current option flags**
$$	**Process ID of the shell**
$!	**Process ID of the most recently executed background process**
$0	**Name of the shell**
$_	**The absolute filename of the shell script**
$1, $2..$9	**First, second through ninth arguments passed to the script**

The for loop works on lists, not numbers. To count through a sequence using a for loop, you use Bash's arithmetic for statement:

```
for (( expression1 ; expression2 ;
expression3 ))
```

When the loop is started, 'expression1' is evaluated, using Bash's arithmetic rules. 'expression2' is then evaluated. If it is zero, the loop stops. If it is non-zero, then the commands between 'do' and 'done' are executed, and finally 'expression3' is evaluated. Clear? Yeah, right. To give you an example, the following loop counts down from 10:

```
#!/bin/bash
for (( i=10 ; i ; i=i-1 ))
do
    echo $i
done
```

When the for loop is entered, the variable 'i' is assigned (expression1) the value 10. It then checks to see if 'i' is zero (expression2), echoes the value, and then decrements 'i' (expression3) by one. It repeats this until 'i' is zero.

COMMAND SUBSTITUTION

We'll end this tutorial with a brief mention of command substitution, a powerful shell feature. This allows you to take the output of one command and treat it as if it were written on the command line. To 'capture' the command's output, surround it in backquotes: ``.

WHEN THINGS DON'T WORK

Script not working? Try this checklist:

- **Have you made the script executable?**
  If not, then chmod **it:** chmod u+x [scriptname]**.**
- **Bang-hash-bin-bash — does the first line contain** #!/bin/bash **?**
- **No whitespace around assignment statements:** var=value**, not** var = value**.**
- **Whitespace around a test:** [var = value]
  not [var=value]**.**
- **You can see your script run line-by-line by executing it with the trace option:**
  bash -x [scriptname]**.**

Here is a script called search that looks through a directory and its subdirectories and reports the names of files that contain a string passed on the command line.

It's a very simple yet incredibly handy script; if you're ever looking for a file and you can remember some words contained in it, this is for you:

```
#!/bin/bash
if [ $# -eq 0 ]
then
    echo "Usage: search string"
else
    grep -l $1 `find . -name "*" -type
f`
fi
```

This script uses two powerful commands, grep (which we covered earlier) and find (covered in the previous Linux Pocketbook). The former looks through a file trying to match a pattern while the latter lists files

TABLE 2: BUILT-IN TESTS

STRING TESTS	
[-n] STRING	**the length of STRING is nonzero**
[-z] STRING	**the length of STRING is zero**
STRING1 = STRING2	**the strings are identical**
STRING1 != STRING2	**the strings are not identical**
NUMERICAL TESTS	
NUM1 -eq NUM2	**NUM1 is equal to NUM2**
NUM1 -ge NUM2	**NUM1 is greater than or equal to NUM2**
NUM1 -gt NUM2	**NUM1 is greater than NUM2**
NUM1 -le NUM2	**NUM1 is less than or equal to NUM2**
NUM1 -lt NUM2	**NUM1 is less than NUM2**
NUM1 -ne NUM2	**NUM1 is not equal to NUM2**
File Tests	
FILE1 -nt FILE2	**FILE1 is newer (modification date) than FILE2**
FILE1 -ot FILE2	**FILE1 is older than FILE2**
-e FILE	**FILE exists**
-d FILE	**FILE exists and is a directory**
-f FILE	**FILE exists and is a regular file**

Try man test **for more tests.**

meeting specified criteria.

We've covered the basics of scripting with Bash, and you should be able to write some pretty useful scripts. If you've searched around your system, you'll have noticed that shell scripts are used all over your system for all sorts of tasks. But there is still a lot more to learn, so maybe you're asking 'where to next?'

There are, of course, heaps of references on the Web. Before you consider these you should take a look at other people's scripts. A great repository of useful scripts is the ShellDorado archive site at *www.oase-shareware.org/shell*. There, you'll find not only useful scripts, but reference material and many links to other information, including shell scripting tutorials.

Introduction to Python

Although scripting covers many programming concepts, the shell scripting language isn't as capable as more fully featured programming languages, such as Python.

In this section we'll introduce you to Python as a starting point if you're interested in learning to program a more fully featured language. If you've been around Linux any longer than a month or two, you've probably heard of Python. So what is it?

Like Perl and Bash scripts, Python is a scripting language, and programs are usually distributed and run as text files full of commands. Unlike Bash, Python is not a pure interpreted language. With a Bash script, the Bash interpreter takes lines of the program one at a time and executes them.

With Python however, the Python interpreter reads in the source code file and translates it into an intermediate form, which is run on a virtual machine much like the Java virtual machine.

The advantage of this is that if your program contains looping constructs where the same statement is executed a number of times, the loops are only translated into the intermediate form once. This is similar to how C programs work, except that C programs are compiled to a form which runs on a specific machine type.

As well as being a great language for an experienced programmer to get things done quickly, Python is an ideal language for the first-time programmer. It has built-in support for strings, arrays (multidimensional variables) and dictionaries (which are much like miniature databases) and has a very simple and uncluttered syntax. It encourages good programming practices but at the same time is extremely flexible and powerful.

Python makes it easy to separate parts of a program into blocks and tasks which can be kept in separate files. This helps make programs more modular and in the end more maintainable. It also encourages the writing of generic libraries of functions which can be reused over and over again.

All these features allow Python to scale from small 50-line scripts to large projects with teams of programmers and tens of thousands of lines of code. Python is also an object-oriented language with a clean and elegant class system that hasn't just been bolted onto an existing procedural language.

Now that you're sold on the Python feature set, you're probably wondering what you can use it for. In addition to standard scripting tasks, Python is well suited to GUI programming, CGI scripting, networking servers and clients, text file manipulation and a host of other tasks. Python code is highly portable and it is relatively easy to write code which runs on Unix as well as other operating systems. The only places where compiled languages such as C or C++ have an advantage are in areas like kernel and library programming or in applications where speed is one of the primary objectives.

For a final piece of Python advocacy we suggest you read the experiences of an advanced hacker (Eric S Raymond) with Python: *www2.linuxjournal.com/lj-issues/issue73/3882.html*.

So, lets look at some code! As learning the intricacies of Python would take more pages than can fit into this Pocketbook, we'll introduce you to Python through the use of an example program. If you find yourself craving more, we have provided a number of places at the end of this section where you can start looking.

A SAMPLE PROGRAM

This is a program which prints out the names and the sizes of the five biggest files in the current directory — a Python version of our 'findhogs' bash script:

```
#!/usr/bin/python

import os, stat, string

def get_file_list (my_dir):
    file_list = os.listdir (my_dir)
    new_list = []
    for name in file_list:
        if not os.path.isfile (name):
            continue
        stat_data = os.stat (name)
        s = '%010d %s' % (stat_data [stat.ST_SIZE], name)
        new_list.append (s)
    return new_list

def print_file_list (sorted_list):
    for item in sorted_list:
        size, name = string.split (item, ' ')
        size = int (size)
        print "%20s %10d bytes" % (name, size)

# Main program starts here.

file_list = get_file_list ('.')

file_list.sort ()
file_list.reverse ()

if len (file_list) > 5:
    file_list = file_list [0:5]

print_file_list (file_list)
```

Compared to a Bash script, this is quite long. However this reveals the difference between a fully featured programming language like Python and a scripting language. For a start, in our Python program we're writing our own *functions* — that is, our own commands. The core of the program itself is actually where it states 'Main program starts here', in which case the Python program is no longer than our script. You can see the two new functions we're defining with the two 'def' lines.

As with a Bash script you can run by making the file executable with chmod.

Let's pull apart our program and see how it works.

Python has little of the 'punctuation' that so many other languages require and this makes it much easier to read. The indentation of the blocks of code under the function definitions (and elsewhere) is significant. Unlike C, Perl and a whole host of other languages, Python does not use braces or BEGIN and END statements to show which statements are logically grouped together. Instead, Python uses indentation. There is no requirement for a specific amount of spaces or tabs (remembering that a tab character is equivalent to eight spaces); you just have to be consistent. All the lines of code which are supposed to be grouped together should be at the same indentation level.

Many programmers, when they see this for the first time find it rather objectionable. However, after a short period actually working with Python, they become used to it and even begin to like it. Python is simply enforcing good programming practice.

In the first function 'get_file_list', the first three lines are indented by four spaces. The third of these lines is the start of a for loop. The for loop executes all the lines between it and the 'return' statement. Inside the for loop there is also an if statement with a single line inside its indentation level. Only if the condition after the if statement is true will this single line be executed.

Similarly, the second function 'print_file_list' has a for statement with three lines of code within its indentation level. One last thing to note is that any text on a line after a '#' character is a comment. Comments are not executed like the other lines in the program but are notes to remind the programmer or inform someone else who has to read the program later what their intentions were. The one exception to this is the first line which starts with '#!'. This line tells the operating system that

If you're interested in learning to program under Linux, Python is an excellent place to start.

this file is an executable script, and that the interpreter to run this script is the Python interpreter.

So much for the structure; what does all this code do? The third line is an 'import' statement which tells the Python interpreter to import three of the standard Python modules. The Python distribution comes with about 150 of these standard modules for doing everything from interacting with the filesystem to connecting to and communicating with FTP and HTTP servers. Here we're importing modules for performing OS-related tasks and a module for manipulating text strings. You might notice that in the first line of the first function we make a call to the function 'os.listdir'. Given this function name, Python will look in the 'os' module for a function named 'listdir'.

When this program is run, the first statement that is executed is the statement 'file_list = get_file_list ('.')'. What we're doing here is passing a string representing the name of the current directory ('.') to the function 'get_file_list'. This function, when it has finished, returns a list of filenames which gets assigned to the variable 'file_list'. The next two lines sort the list and then reverse its order. Then, if the length of the list is greater than five elements, the list is truncated so that it only contains the required number of elements. If the list has less than five elements, the statement 'file_list = file_list [0:5]' is not executed. Finally, the list is passed to the function 'print_file_list' before the program terminates.

Now that you understand the mechanics of the main part of the program we can have a look at the functions beginning with 'get_file_list'. In the first line we call 'os.listdir' which returns a list containing the name of every item in the given directory. The following line declares a new empty list represented by open and close square brackets with nothing between them. The new empty list is assigned to the variable 'new_list'. The next line contains a for statement which takes each element of 'file_list' one at a time and assigns it to the variable 'name' without modifying 'file_list'. The code within the loop is then executed with the given value of 'name'.

In the body of the for loop, we first of all check whether the directory item 'name' is a file by calling the module function 'os.path.isfile'. If it isn't a file (it may be another directory for instance), the rest of the loop is bypassed by using a 'continue' statement. If on the other hand 'name' is a file, we call 'os.stat' to retrieve information about the file.

On the following line, one particular part of this information, the size of the file, is used to construct a string. This string will contain the file size as a 10-digit number (with leading zeros), followed by a space and then the name of the file. If, for instance, there was a file called 'foo' which was 1,234 bytes long, the string created would be '0000001234 foo'. This string is then appended to the end of 'new_list' which was created earlier. When every element of 'file_list' has been processed, the function returns 'new_list' to the main part of the program where 'get_file_list' was called.

The function 'print_file_list' is even simpler. For each string item of the sorted list that has been passed to it, it splits the string in two at the space character. It then converts the first part a string representing the size of the file to an integer. Finally, it prints out the names and the sizes of the files in a neatly formatted manner. Simple isn't it? (We hope so!)

Most Linux distributions come with Python as a standard component and it should already be installed on your system. If not, you'll find it on your distribution CDs or online at *www.python.org* along with extensive documentation, tutorials and programming examples. There is also a newsgroup, *comp.lang.python*, devoted to Python.

This short introduction has hopefully given you an idea of Python as a programming language.

PYTHON RESOURCES

Python Hompage
www.python.org

PythonWare
www.pythonware.com

Vaults of Parnassus
www.vex.net/parnassus

Starship Python
starship.python.net

How do programs run?

Ever wondered what happens when you run a program? Ever downloaded software from the Net that didn't work, even though it may have compiled and installed correctly?

If the answer to either of these questions is yes, you probably want to know more about how Linux runs programs. Even if you can't say yes to either of these questions, you'll undoubtedly be curious.

In this section we'll look at how a program starts up when it is run from the command line. We won't deal with running a program from a menu pop-up in X, as this depends on which window manager you're using.

You're at the console and you type foo and then press Enter. One of two things will happen: the shell will print out a message saying that the command foo' was not found, or it will run a command, foo.

When asked to run a command like this, Bash takes a number of steps to figure out what to do. First it checks whether the command is a built-in Bash command like while, for or if (to see a list of all the inbuilt commands type man while at the prompt). If the given command was not a built-in command it then checks if it is a Bash alias or function. Aliases and functions can be defined at the command prompt or in one of the Bash initialisation files; either the per-user ones (~/.bash_profile or ~/.bash_login) or the global one (/etc/profile).

If the command foo was not found in any of the above steps, Bash then looks for the command in the directories listed in the PATH environment variable. Finally, if it could not find the given command in any of these directories Bash will print out the dreaded 'bash: foo: command not found' message.

If foo does exist and the person running Bash has execute permissions for the file, Bash will execute it. Inside the Linux kernel, there is some code to figure out how to run the file. For instance, the file may be an ELF executable or a script for one of the many scripting languages common on Linux. If foo is a script, the kernel will find the script interpreter's name in the first line of the file (such as '#!/bin/bash') and start the interpreter, passing it the name of the script it has to execute.

If, on the other hand, foo is an ELF executable, then it's much more complex. Firstly, there are dynamically and statically linked ELF executables. Statically linked executables are completely self-contained and do not need any external libraries to run. When a statically linked binary is running, the only time it executes code outside of itself is when it makes a system call. A system call is a call into the Linux kernel to perform tasks like opening, reading or writing a file, starting another program, getting process information or any of a number of other tasks. Library calls are similar in many ways but pure library calls do not need to enter the Linux kernel to complete their required operation.

Dynamically linked executables, on the other hand, do use code from external libraries, which are shared with other programs, to perform some of their work. There are a number of advantages to this. Firstly, it means the program will take up less disk space because part of its code is stored elsewhere. In addition, the application programmer's job is easier as they can use code written and debugged by someone else instead of writing it themselves from scratch. More importantly, it reduces the usage of system memory.

Consider three programs all running at the same time and all linked to 'libxxxx.so'. This library will contain code, static data such as strings and constants and data that is modified during the running of the library code. The Linux kernel will only load the code and static data sections of this library into memory once, but it will give all three programs access to it. The only part of the library which will have to be replicated is the dynamic data section; the part of the library that changes during its running lifetime.

You might already have seen packages such as 'gnome-libs' and 'kdelibs'. When you install a new Gnome or KDE program, it can't run without the Gnome or KDE libraries installed as it calls on these to perform certain functions — such as loading a file save dialog or drawing buttons a certain way. In fact, even KDE itself was built with the QT toolkit — a set of libraries that provide GUI functions to help make producing an interface a breeze. Hence, to run a KDE program you need the KDE and QT libraries installed.

When the Linux kernel reads the headers of a program and detects that the program is a statically linked executable it simply runs the program. However, for a dynamically linked program it needs to

WHAT'S THIS ABOUT ELVES?

ELF is an acronym for Embedded Linking Format and is really just a specification for how a binary program is packaged. An ELF binary contains information about what part of the binary is code, what part is static data (such as test strings which are printed out) and so on. A binary can also contain things such as debugging information which the GNU debugger gdb **requires to display the source code of the program while it is being debugged.**

figure out which dynamic linker is needed by the program. Historically there have been three different dynamic linkers on Linux and most systems have all three: '/lib/ld.so', '/lib/ld-linux.so.1' and '/lib/ld-linux.so.2'. The last one is the most recent; it is used by programs linked to glibc-2.1 and is the only one we will look at here. All dynamically linked programs will have one of these three dynamic linker names in its ELF header.

Once the kernel has validated the ELF executable and figured out which of the three dynamic linkers to use, it runs the linker and passes it the name of the ELF executable. The dynamic linker then reads the headers of the file and finds a list of dynamic libraries that the program requires to run. The list of library names does not contain the full paths of where the program expects the libraries to be, just the names. This allows the libraries to be placed anywhere on the system and as long as the dynamic linker knows where they are, they will be linked successfully.

So, how does the dynamic linker know where to find the libraries? This is Linux, so it'll be in a configuration file. This file is /etc/ld.so.conf, which contains a list of library directories with one directory per line. If you have a look at this file on your system you'll notice that '/lib' is not in the list. That's because that is the one directory the dynamic linker searches automatically. Once the linker has found all the libraries the program needs, it loads the program itself and then runs it.

Now, back to where we started: why won't a program sometimes run? If you have a situation like this, you should now be able to figure out why. If, when you type foo at the command line, you get a 'command not found' message, this means that Bash can't find it anywhere on the current path, which you can fix. If, instead, the Bash shell says something like 'Permission denied', you have a file permissions problem, which you might be able to fix.

If the program foo is available and has execute permissions, it still may not run if a dynamic library it requires is missing. You can use the command file /full/path/to/foo to tell you whether it is dynamically or statically linked along with some other useful information. To find out which libraries a program needs you can use the ldd command: run ldd /full/path/to/foo to get a full list of the libraries used and their locations.

Now you know what goes on under the hood, you're better equipped to solve these problems when they arise.

Keeping up to date

Much of the software that is available for Linux is developed by groups of programmers who are widely dispersed and have never met. How do they do it?

How do Linux programmers manage to work remotely yet efficiently on such large and complex projects? The tool that allows these projects to proceed without descending into chaos is CVS, the Concurrent Versions System.

CVS works a little like a database which can store many different versions of a piece of software. The concurrent part in the name alludes to its purpose of allowing many developers to work on the same piece of code concurrently. The repository keeps the differences between versions, which allows changes in the code to be rolled back to track bugs or explore new avenues. In other words, CVS is like a programmer's groupware.

The CVS package consists of two parts: a server and a client program. The server maintains the source code and the changes while the client connects to the server to download the code or upload changes (check-out and check-in respectively in CVS parlance). The CVS server machine can either be the local machine or a machine somewhere on the far flung reaches of the Internet. Developers use the CVS client to connect to a central CVS server for a project to download all the latest updates and upload any of their own changes. Once the client finishes working, the developer's local copy of the source code tree will be identical to that on the server. It's an efficient method for keeping everybody's source code up to date.

But CVS isn't just useful for developers. Many open source projects allow anonymous access to the CVS archive to download the most recent version of the source code tree.

SETTING UP CVS

Want the latest cutting edge version of KDE and don't want to wait two months for the next release? Welcome to CVS.

Setting up CVS is simple. You may well already have the CVS package installed on your system. Type cvs -v and see if it's present. If not, it will surely be included on your distribution CDs, or you can download the latest version from *www.cvshome.org*.

An alternative to CVS is CVSup. Unlike CVS, which is a console-based application, CVSup uses a basic

graphical front-end to report activity. It also uses a different protocol, so you'll find many projects supporting CVS, CVSup, or both. CVSup can be downloaded from *www.polstra.com/projects/freeware/CVSup*.

To begin with, we'll go for a smaller project rather than something as big as KDE, Gnome or XFree86 — all of which are available as CVS. Let's take a look at getting the source tree for LAME, the popular open source MP3 encoder.

The LAME sources, like so many other projects, are kept in CVS form at Sourceforge. The Web page at *sourceforge.net/cvs/?group_id=290* gives a very brief description of how to download the LAME source code using CVS. Here we'll go through it a little more thoroughly and explain what's happening.

The following command assumes your machine is connected to the Net:

```
cvs -d:pserver:anonymous@cvs.lame.sourceforge.net:/cvsroot/lame login
```

This command logs you onto the CVS server at *cvs.lame.sourceforge.net* as an anonymous user; the '-d' option tells CVS the name of the CVS server. After the server comes the path to the CVS repository on the server followed by the CVS command 'login'.

When executing this command you will be asked for a password. In this particular case there is no password and you should just press Enter. For other CVS servers the anonymous CVS password may be 'anoncvs' or something similar, but it is always detailed on the project's CVS page. Once you have logged in successfully, you can download the latest LAME source code with:

```
cvs -z3 -d:pserver:anonymous@cvs.lame.sourceforge.net:/cvsroot/lame co lame
```

The last part of the command, 'co lame' checks-out the code for LAME while the '-z3' switch specifies that the code should be compressed using gzip before being sent over the Internet. As the command runs you'll see a number of filenames being printed out as the files are downloaded. When the files have finished downloading you can change into the 'lame' directory which has just been created to configure, make and install the software just as you would any other source code package.

Later, when you want to update your version of LAME, you just need to go back to the directory on your system where you issued the commands and run them again, substituting 'co' with 'up' (for *update*). CVS will then download only the *changes* since your last update, making updating — even with large projects like KDE — very quick.

In fact, the only difference between downloading the source code for a large project like KDE and doing so for a smaller one like LAME is the amount of disk space you need to hold the code. Downloading each using CVS is essentially the same.

The only other point you should be aware of is that, due to the way CVS operates, and especially the development methods different projects use, the steps taken to compile the source may differ. Always read up on the project's home page for compiling instructions.

Enjoy exploring the world of cutting edge source!

CVS PAGES FOR POPULAR PROJECTS

KDE
www.kde.org/anoncvs.html
www.kde.org/cvsup.html

Gnome
www.gnome.org/start/source.html

XFree86
www.xfree86.org/cvs

GIMP
www.gimp.org/devel_cvs.html

XMMS
www.xmms.org/cvs.html

USE THE SOURCE

CVS
www.cvshome.org

CVSup
www.polstra.com/projects/freeware/CVSup

Why open source?

Ever wondered why it is that programmers labour long and hard on software, only to give it away? To round off this chapter we'll look at why programmers write free software.

THE COMMERCIAL REALITY

Software isn't the easiest thing to create. Building a new application can be fun for a while, but adding features, fixing bugs, designing user interfaces and writing documentation can be very time consuming — yet still open source projects continue. Open source software, including nearly all major Internet server software, competes with similar closed applications that sell for thousands of dollars in licensing and access fees. Why would someone want to give away software when they could charge a considerable amount of money for its use? There are many possible motives for writing open source software, and the personal motives of developers vary considerably.

Like their closed source counterparts, many larger open source software projects are created by authors who are paid wages for their work. You don't have to sell applications to make money. Red Hat provides a Linux distribution free of charge, but sells goods such as manuals and training, professional services for managing computer networks. Digital Creations, the maker of the popular Zope Web application toolkit, was a closed source developer. It decided that by giving Zope away, it could win considerable market share over its entrenched competitors and then use the newfound popularity of its software to profit from consulting and custom development.

But most of these larger projects didn't start off with commercial aspirations. They started out small, and like most smaller projects today, were originally produced for altruistic reasons. So the question remains: if you're not getting money out of doing it, why create software and let people use it as they please?

There are other rewards involved besides money. Open source guru Eric S Raymond states that "Every good open source project starts by scratching a developer's personal itch". Haven't you ever wanted to change or create a piece of software to produce something that works the way you think it should? If you've got the knowledge to do so, why not do it? Guenter Bartsch found the only existing DVD player on Linux was difficult to install and use. So he used the best bits of it, combined it with his own work, and produced the brilliant Xine DVD player. This gained the attention of other contributors, and because Xine was open source, they were able to consolidate their efforts into a single project.

There's also a large sense of community between open source contributors and users. Since many projects are created using existing OSS text editors, compilers, documentation and graphics tools, it often makes sense to give back to this community by using an open source licence. Within the community, those who contribute are also rewarded with respect from their peers. This can prove useful for impressing potential employers, or even just for flying round the world discussing latest projects at Linux conferences.

In addition, most open source contributors are the type of power users who enjoy being able to tailor and customise their environments the way they please. Open source makes this possible and is the reason why one person's embedded firewall can be another's secure GUI desktop. By creating software with source code that's free to be modified, the author creates software that's infinitely customisable.

CLOSED OR COMMERCIAL SOFTWARE?

The opposite of open source or Free software is closed source or proprietary software. Many open source projects are produced for commercial reasons — so using the term 'commercial software' to describe closed source applications isn't quite accurate.

THE PHILOSOPHY

There are other reasons why programmers create software and allow other people the ability to use, view, copy, modify, give away and charge for the source code. The answer lies in a little bit of history that starts, as always, with humble beginnings.

In the early '70s, software was always released as source code so that anyone could fix bugs, modify it, or improve it. This was the norm until the early '80s when, as a young developer by the name of Richard Stallman noticed, things began to change and more and more software was released as binaries only.

This came to a head for Stallman when he was unable to fix a bug in a printer driver that was causing him hassle and he discovered that the maker of the software wouldn't reveal its source code for him to fix. Stallman was incensed at something that seemingly crippled the software, and believed that it was wrong to impose a licensing model on something intangible like software (where ownership can be given to others without the original owners losing it). It is this distinction that separates software from traditional material commodities.

In response to this, Stallman created the Free Software Foundation (*www.gnu.org/fsf/fsf.html*) and later the GNU Project (*www.gnu.org*). For Stallman 'free' is meant in the sense of liberty — users should be free to fix bugs, modify or improve the software. The aim of the FSF is to bring about a computing utopia where all software is free, and having the source open is for the common good of the computing public.

For Stallman 'free' is meant in the sense of liberty — users should be free to fix bugs, modify or improve the software.

This ability to share source code not only allows you to have control over the programs you have purchased, it also allows you to build on the work of others, and others to build on yours. This results in software created by many people, maximising the talents of a wide variety of programmers, and producing results that wouldn't be possible if the code wasn't available.

Stallman didn't just create this philosophy for others to follow; he started out right away creating software whose source code was freely available. The famous Emacs editor, the GNU debugger gdb and the GNU C compiler, gcc were all written by Stallman. Many great projects, from GNU Zip to Gimp, have been produced by the FSF.

However, many free software developers didn't entirely agree with the FSF's stance. Some believed that it shouldn't be 'wrong', as it was in Stallman's eyes, to produce closed source software at all, but that providing source code is preferable simply because with more people contributing and fixing bugs, better software is produced. As a result, the Open Source Initiative (*www.opensource.org*, a non-profit organisation) was created in 1998.

The term 'open source' was invented because it distinguished free software from freeware in the minds of the general public. Open source advocates don't mind coexisting and interoperating with closed source software and see themselves as being business friendly. Since the formation of the Open Source Initiative, Linux has grown dramatically. This is largely because the OSI has promoted a healthy relationship between open source and business, and is seen by many to have more moderate views than the FSF.

For end users, there is little difference between open source software and free software. Most open source and free software development isn't overly political, and neither organisation speaks for all the developers of software with modifiable source code. The conditions a piece of software must meet to be recognised as being open source or free software, and licensed as such, can be found at *opensource.org/docs/definition.html* and *www.gnu.org/philosophy/free-sw.html* respectively. Both of these documents spell it out in a clear and concise manner, and give users the freedom to use, view, copy, modify, give away and charge for the source code.

Helping the cause

The Linux community

Linux has no doubt impressed you by now. Powerful, efficient, reliable software obtainable for free. Wonderful. But it didn't just appear all by itself.

WHY YOU SHOULD CONTRIBUTE

Linux has evolved, and continues to evolve, under the open source philosophy of the free sharing of ideas. If not for this massive, shared, open community environment, Linux — and much of the open source software that runs the Internet — wouldn't be here. It's a testament to this community and its philosophy that the creation of a powerful operating system such as Linux could come about through the interaction and cooperation of thousands of people worldwide.

Linux is a symbol of what's possible when we work together as a whole, sharing each other's ideas and building upon each other's work. It's built by everybody, for everybody, and it's free.

If you're a home user, Linux has hopefully provided you with a stack of useful applications which have helped you get your work done and have fun. It may have also provided you with an excellent insight into how your computer and modern Internet-based networks operate. If you're in a business environment, it may have saved you thousands of dollars in licence fees, and saved you a great deal more by simply being a reliable server OS. You might be writing closed source applications that run on this powerful and stable platform. You might be providing consulting services to help other businesses take advantage of the operating system. If you're like some of the authors of this book, it may even have provided you with a career. If you've benefited from Linux and open source, perhaps you'd like to contribute? As a community, every contribution helps build a better product. The more the merrier.

If you want to show your appreciation for what Linux has enabled you to do, achieve or experience, you can offer your own skills and services to improve the operating system, and through this others will benefit just as you have benefited from the work of others. This is the beauty of community.

HOW TO HELP

Perhaps the first thing you should know is that you don't have to know how to program to be of help.

You can help out by writing documentation, managing open source projects, hosting file space, providing information over the Web, and much more. Whatever you choose, the easiest method is to join an open source project. Unless, of course, you're a programmer with an ambition to have your code accepted into the kernel for the fame of having your code used by millions worldwide!

HELPING OPEN SOURCE PROJECTS

Open source projects are well coordinated and run under a meritocratic system (those who have contributed the most to the project are higher in the hierarchy) while still preserving the fun atmosphere OSS development is renowned for. They use various open source management tools, including CVS (a revision control system), Bugzilla (a bug tracking system), GPG (a digital message signing tool), and various mailing list management applications. You'll probably need to find out what systems your chosen project is using and read the various online guides to familiarise yourself.

The quintessential guide to working within an open source project (for programmers, documenters or otherwise) is Eric S Raymond's *'The Cathedral and the Bazaar'*. It is available both as a hardcover book and free online from *www.tuxedo.org/~esr/writings/cathedral-bazaar*. Every project contributor should read it at least once, as it is often used as a reference for guiding the development of projects.

PROGRAMMING

Keep in mind that not everyone can be a kernel hacker overnight, and furthermore, there are bigger, bolder efforts underway elsewhere. So join an existing project before beginning your own open source masterpiece. Joining an existing project will also allow a smooth transition into open source methods, and working with other people's code. This is important, as even if you start your own project you should avoid reinventing the wheel by basing it on libraries and code from other projects wherever possible.

The best way initially to get involved in a project is to identify and fix bugs, clean up code, and so on. By reading all the information on the project's site and chatting with the other developers, you'll soon get an idea of where the project is headed and can nominate yourself for a specific task to help achieve that goal.

It's important to realise that not every submission or idea is a brilliant one, and that your efforts may well be rejected.

If this happens, pay attention to the feedback. Many major parts of Linux (like the powerful Reiser FS) were initially rejected, then worked on in line with feedback before their eventual acceptance into the Linux kernel.

DOCUMENTATION

If you're a technical person but don't desire to code full time, or don't code at all, and you can write well, then documentation might be just your thing. In fact, open source projects usually need documenters more than new programmers, to write FAQs, HOWTOs, UI guidelines, and much more. If you've ever found documentation to be lacking, then don't complain — search out the appropriate parties and write it yourself! If you find your work beneficial, others will too.

There are a few things you'll need to learn, however; most open source projects are documented using SGML, a markup language specifically for structured documents. SGML lets authors create documents without having to worry about how they look. It simply requires you to pick a Document Type Descriptor (or DTD) that defines the structure of the document (how the heading, subheadings, images, and tables will relate to each other) and then define items in your documents as one or more of these structures. When it's time to publish, an SGML document converts easily to HTML, PDF and even Microsoft Word in a single, easy step.

HELPING PROJECTS IN OTHER WAYS

The most obvious methods are by programming and documentation, but many projects also require artists (KDE has a whole team who simply draw and regulate icons), Webmasters, skin designers, packagers, and more. Enquire at the Web site for a project about how you can assist.

OTHER CONTRIBUTIONS

If you don't have the time or skills to devote to a particular open source project, then there's still a myriad of ways you can help:

BUG HUNTING

All software needs good testers, and reporting helps you and other users. Take care to follow the guidelines for submitting bugs for a particular project, usually found on the project Web site or in the README file. Many open source projects use specialised bug tracking systems, such as Bugzilla or Sourceforge, which must be adhered to if the bug is to be accepted. The more detail you can give when submitting a bug report the better. For example, is there anything you think might be causing the problem? If something stopped working, when did it last work, and what's changed since then? Take care to follow up your bug report too — you might need to answer some extra questions to help programmers fix the bug.

Some programs, and certainly environments like KDE, come with built-in bug reporting facilities that make it trivial to type in a bug report and send it to the developers.

ADVOCACY

If you love Linux, you'll probably want to tell others about it. There are right and wrong ways of doing so; using terms like 'Windoze' or 'Micro$oft' definitely falls into the latter category. Nobody likes being told they selected their OS poorly, and it's unlikely to endear them towards whatever you're advocating.

Refusing to acknowledge Linux's deficiencies (yes, they do exist) is likely to invoke similar feelings. Be positive and polite: emphasise Linux's good points rather than another operating system's bad points.

Above all, make sure that learning Linux is as easy as possible; point people to the best books and Web sites; show them where they can get help; and answer their questions if you know the answers. If you don't, point them to somewhere which can. Don't overwhelm them with technical jargon, and keep in mind that not every user can understand man page information. Read the advocacy HOWTO in /usr/share/doc/HOWTO for hints on successful advocacy.

HELPING OTHER USERS

You might remember way back in the first Linux Pocketook we mentioned that going to other Linux users — the community at large — was the single best method of seeking help. If you've been with us all this time you're probably more experienced with Linux than you realise. You're now one of those users who can help new users ease the learning curve. So if you hang around the news groups, mailing lists, Web site discussion forums and IRC channels and you see someone who needs help, lend them your knowledge and experience. You might not know enough about Linux to be able to build a clustered supercomputer, but if you encounter another user online who's having the same sound-card problem you once needed help with, give them a hand.

HELPING YOUR LOCAL LUG

There are like-minded Linux users all over the world, and every state in Australia has at least one Linux User Group, ranging from 100 or so members to over 1,000 in the larger states, in registered, non-profit organisations. Their work typically involves many functions:

Mailing lists Besides the common use of free support between LUG members, the lists are also used for Linux discussion, event coordination, special interest groups (such as programmers or gamers) and more. Mailing lists are hectic things, and always require administrators to help people subscribe, unsubscribe and change their details; make sure posts are directed to the mailing lists; moderate posts; make sure lists stay on topic; and create new mailing lists. If you're a System Administrator type capable of handling a list, why not volunteer?

Meetings Many LUGs hold monthly

meetings, where users discover the latest developments in Linux, listen to educational talks on a variety of topics, enjoy the odd giveaway and, most importantly, go out for pizza afterwards. Speakers are always welcome, and so are talks aimed at Linux newcomers. If you have public speaking aspirations and some knowledge to share, now's your chance to make a difference to other users.

Web sites LUG Web sites need to be good-looking and navigable across a variety of platforms and browsers, including the bare bones text browsers many old-school Unix users prefer. If you're capable of designing attractive, clearly presented Web pages, your LUG would no doubt appreciate your help.

Event coordination LUGs create and participate in all manner of events, ranging from Installfests and gaming LANs to Linux conferences and IT industry trade shows. These events require all sorts of organisation such as marketing, equipment donorship, sponsor liaisons and caterers.

Finance All these events involve funding, and most larger LUGs need treasurers to take care of the financial side. If you're good with numbers, volunteer!

HIRING OPEN SOURCE DEVELOPERS

If you're a business and would like the ability to customise an open source piece of software to your needs, or fix your own bugs, hiring an open source developer is an excellent option. It allows you to have the features you need, either in the main application or your own branch, as well as have direct knowledge of how the software works, on hand.

SUPPORT COMPANIES THAT CREATE OPEN SOURCE

If you can't hire an open source developer yourself, why not support a company that does? Companies like Mandrake, Red Hat, Progeny (a Debian-based distribution), Caldera, theKompany, Eazel, Ximian, VA Linux, and Linuxcare all pay the wages of various open source developers. Their business models include professional services, education, hardware sales, closed software sales, hosting, custom development and certified training courses. Many of these companies now have presences within the Asia Pacific region, so when you're looking for products and services they can provide, you might consider giving them your business.

SUPPORT COMPANIES THAT WORK WITH OPEN SOURCE AND LINUX

Not only can you support the companies that create open source, you can support the companies that support them. For example, ATI was good enough to pay VA Linux to create a brilliant set of drivers for its Radeon cards.

Other companies write closed source applications on the Linux platform. Sometimes they're better than open source equivalents, sometimes not. Buying quality closed source products that are available for Linux shows these companies that you appreciate the choice of platform.

DONATE TO OPEN SOURCE PROJECTS

Open source projects need hardware, bandwidth and legal services. If you're in a position to provide these services, why not make a donation?

Contributing to the development of Linux is not only a good way to appreciate the work of others, it's also exciting. Being part of a team, working on new projects and knowing that others will get to see and appreciate your work is a fantastic experience. If you've been looking for ways to give back to the community that's given you so much, you now have plenty of options!

AUSTRALIAN LINUX USER GROUPS

CLUG	Canberra Linux Users Group	*samba.anu.edu.au/clug*
HUMBUG	Home Unix Machine, Brisbane Users Group	*www.humbug.org.au*
LinuxSA	Linux Users Group of South Australia	*www.linuxsa.org.au*
LUV	Linux Users of Victoria	*www.luv.asn.au*
MLUG	Melbourne Linux Usergroup	*www.mlug.org.au*
PLUG	Perth Linux Users Group	*plug.linux.org.au*
SLUG	Sydney Linux Users Group	*www.slug.org.au*
USQLUG	University of Southern Queensland Linux Users Group	*www.sci.usq.edu.au/lug*