Linux Sea

Sven Vermeulen

Linux Sea

Sven Vermeulen

Abstract

The book "Linux Sea" offers a gentle yet technical (from end-user perspective) introduction to the Linux operating system, using Gentoo Linux as the example Linux distribution. It does not nor will it ever talk about the history of the Linux kernel or Linux distributions or dive into details that are less interesting for Linux users.

For various topics, the online Gentoo Handbook offers a very detailed approach and as such is mandatory reading for any Gentoo Linux user who wants to know the full power of this Operating System. Although there is definitely overlap between "Linux Sea" and the online Gentoo Handbook, "Linux Sea" is by no means meant to replace the online Gentoo Handbook.

"Linux Sea" will attempt to focus on topics that everyday users would probably need to know to continue working with Gentoo Linux.

The version you are reading currently is v1.16 and has been generated on 2013/12/16. PDF [http://swift.siphos.be/linux_sea/linux_sea.pdf] and ePUB [http://swift.siphos.be/linux_sea/linux_sea.epub] versions are available as well.

Table of Contents

List of Figures

List of Tables

Chapter 1. What is Linux?

Introduction

Within the realms of desktop environments, Linux is supposedly no large player (market investigations estimate a market share of about 3%). However, you'll most likely know two or more people who use Linux, some of them even exclusively. If you take that into consideration, either you know the personal OS preference of over a hundred people (you're popular) or the estimation is to be taken with a grain of salt.

Nevertheless, 3% is still a lot (ever thought about how many desktop systems there are? I did, never found the answer though). And if we take other markets into account (embedded systems, servers, network appliances and more) the Linux share increases.

But still, many people have no idea what Linux is or how to work with it. In this book, I offer a technical, quick introduction to the Linux operating system from a users perspective. I'm not going to dive into the concept, advantages or disadvantages of Free Software (although a few paragraphs don't hurt) and am not going to talk about the history and evolution of Linux Operating Systems. For more resources on these subjects I refer you to the Further Resources section at the end of this chapter.

For a book to be complete about Linux as an operating system, it is important to inform the user about operating systems in general. Linux is very modular and open, meaning that each component in the Linux Operating System is visible to the user. Without understanding of the structure of an operating system it would be hard for a user to comprehend the reasoning behind each module in the Linux OS. For this reason alone I devote an entire section on operating systems in general.

Once I have crossed the tasks of an operating system, I continue with an explanation on the real Linux operating systems: Linux distributions.

Finally, each chapter in this book will offer a set of exercises that you could attempt to solve. You will not be able to find the answers of each question in this book. Rather see the exercises as a means to push you further and help you seek (and find) more topics related to Linux. At the end of the book, a list of tips and/or answers is given for each question.

The Anatomy of an Operating System

An operating system is actually a stack of software, each item designed for a specific purpose.

- The *kernel* is the core of an operating system: it manages communication between devices and software, manages the system resources (like CPU time, memory, network, ...) and shields off the complexity of device programming from the developer as it provides an interface for the programmer to manipulate hardware.

- The *system libraries* contain program methods for developers to write software for the operating system. The libraries contain methods for process creation and manipulation, file handling, network programming, etc. It is a vital part of an operating system because you can't (or shouldn't) communicate with the kernel directly: the library shields off the complexity of kernel programming for the system programmer.

- The *system tools* are built using the system libraries and enable administrators to administer the system: manage processes, navigate on the file system, execute other applications, configure the network, ...

- The *development tools* provide the means to build new software on (or for) the system. Although not a required part of an operating system I do like to mention it because with Gentoo, this is a requirement (we'll see later on why this is the case). These tools include compilers (translate code

to machine code), linkers (which collect machine code and bring it together into a working binary) and tools that ease the build process considerably.

Other libraries on the system enhance the developers' coding experience by providing access to methods other developers have already written. Examples of such libraries include graphical libraries (for manipulating windows) or scientific libraries. They aren't required on every operating system, but if you want to run a specific tool, it will require certain libraries to be installed on your system. On top of those additional libraries you will find the end-user tools (office suites, multimedia tools, graphical environments, ...) that you want to install on your operating system.

Kernel

A kernel has generally four basic responsibilities:

- device management
- memory management
- process management
- handling system calls

The first responsibility is called *device management*. A computer system has several devices connected to it: not only the CPU and memory are available, but also disks (and disk controllers), network cards, graphical cards, sound cards, ... Because every device operates differently, the kernel is required to know what the device can do and how to address and manipulate each device so that it plays nice in the entire system. This information is stored in the device driver: without such driver, the kernel doesn't know the device and will therefore not be able to control it.

Next to the device drivers, the kernel also manages the communication between the devices: it governs access to the shared components so that all drivers can happily operate next to each other. All communication must adhere to strict rules and the kernel makes sure that these rules are followed.

The *memory management* component manages the memory usage of the system: it keeps track of used and unused memory, assigns memory to processes who require it and ensures that processes can't manipulate the data of one another. To do this, it uses the concept of virtual memory addresses: addresses for one process are not the real addresses, and the kernel keeps track of the correct mappings. It is also possible for data not to be really present in memory although it is present for a process: such data is stored on a swap space. Because swap space is much, much slower than real memory, use of this space should be limited to data that isn't read often.

To ensure that each process gets enough CPU time, the kernel gives priorities to processes and gives each of them a certain amount of CPU time before it stops the process and hands over the CPU to the next one. *Process management* not only deals with CPU time delegation (called scheduling), but also with security privileges, process ownership information, communication between processes and more.

Finally, for a kernel to actually work on a system, it must provide the means to the system and the programmer to control itself and give or receive information from which new decisions can be made. Using *system calls* a programmer can write applications that query the kernel for information or ask the kernel to perform a certain task (for instance, manipulate some hardware and return the result). Of course, such calls must be safe to use so that malicious programmers can't bring the system down with a well-crafted system call.

A Linux operating system, like Gentoo Linux, uses Linux as the kernel.

System Libraries

Because a kernel can't do much out of itself, it must be triggered to perform tasks. Such triggers are made by applications, but these applications must of course know how to place system calls for the

kernel. Because each kernel has a different set of system calls available (it is very system specific), programmers have created standards with which they can work. Each operating system supports these standards, and these are then translated to the system specific calls for that operating system.

One example standard is the C library, probably the most important system library available. This library makes pretty vital operations available to the programmer, such as basic input/output support, string handling routines, mathematical methods, memory management and file operations. With these functions a programmer can create software that builds on every operating system that supports the C library. These methods are then translated by the C library to the kernel specific system calls (if system calls are necessary). This way the programmer doesn't need to know the kernel internals and can even write software (once) that can be build for many platforms.

There is no single specification on what a system library is. The author of this book believes that system libraries are whatever library is part of the default, minimal install of an operating system. As such, system libraries for one operating system (and even Linux distribution) can and will differ from the system libraries of another. Most Linux distributions have the same system libraries, which is to be expected because all Linux distributions can run the same software and this software is of course built for these system libraries. Some distributions just don't mark one library part of the default, minimal install while others do.

The most well-known system library for Linux systems is the GNU C Library, also known as glibc.

System Tools

Just like with system libraries there is no single specification for *system tools*. But, unlike system libraries, system tools are quite visible to the end user. Because of this, almost all Linux distributions use the same system tools, or similar tools with the same features but different implementations.

But what are system tools? Well, with a kernel and some programming libraries you can't manipulate your system yet. You need access to commands, input you give to the system that gets interpreted and executed. These commands do primitive stuff like file navigation (change directory, create/remove files, obtain file listings, ...), information manipulation (text searching, compression, listing differences between files, ...), process manipulation (launching new processes, getting process listings, exiting running processes, ...), privilege related tasks (changing ownership of files, changing user ids, updating file permissions, ...) and more.

If you don't know how to deal with all this stuff, you don't know how to work with your operating system. Some operating systems hide these tasks behind complex tools, others have simple tools for each task and bundle the power of all these tools. Unix (and Linux) is one of the latter. Linux systems usually have the GNU Core Utilities for most of these tasks.

Development Tools

With the above three components you have a running, working operating system. You might not be able to do everything you want, but you can update your system until it does what you want. How? By installing additional tools and libraries until you have your functional system.

These additional tools and libraries are of course written by programmers and they must be able to build their code so that it works on your system. Some systems, like Gentoo Linux, even build this software for you instead of relying on the prebuilt software by others. To be able to build these tools, you need the source code of each tool and the necessary tools to convert the source code to executable files.

These tools are called a *tool chain*: a set of tools that are used as in a chain in order to produce a working application. A general tool chain consists out of a text editor (to write the code in), compiler (to convert code to machine-specific language), linker (to combine machine-code of several sources - including prebuilt "shared" libraries - into a single, executable file) and libraries (those I just mentioned as being "shared" libraries).

A tool chain is of the utmost importance for a developer; it is a vital *development tool*, but not the only development tool. For instance, developers of graphical applications generally need tools to create graphics as well, or even multimedia-related tools to add sound effects to their program. A development tool is a general noun for a tool that a developer would need in order to create something, but isn't vital for an operating system of an average non-developer user.

The most well-known development tools are also delivered by the GNU foundation, namely the GNU Compiler Collection, also known as gcc.

End User Tools

Once a developer has finished creating its product, you have an end-user tool with accompanying libraries (which might be required by other tools that are build on top of this product). These end tools are what makes a system unique for a user: they represent what a user wants to do with his system. Although not required by an operating system they are required by the end user and are therefore very important for his system.

Most operating systems don't install all or most of the end-user tools because there are just too many to choose from. Some operating systems don't even provide the means to install end-user tools to their users but rely on the ingenuity of each programmer to create an installer that sets up the tool on the system. Other operating systems bring a small but considerable subset of end-user tools with them so that their users can quickly update their system to whatever shape they want without requiring a long and difficult search across the Internet (or even worse, computer/software shop) to find the software they need.

Examples of end-user tools are well known, such as office suites, graphic design tools, multimedia players, communication software, Internet browsers, ...

Okay, I bite, What is this GNU?

The GNU Project is an effort of several programmers and developers to create a free, Unix-like operating system. GNU is a recursive acronym that stands for *GNU is Not Unix*, because it is Unix-like but contains no Unix code and is (and remains) free. The GNU foundation, the legal entity behind the GNU project, sees free as more than just the financial meaning of free: the software should be free to use for any purpose whatsoever, free to study and modify the source code and behaviour, free to copy and free to distribute the changes you made.

This idea of free software is a noble thought that is active in many programmers' minds: hence many software titles are freely available. Software is generally accompanied by a license that explains what you can and cannot do with it (also known as the "End User License Agreement"). Free Software also has such a license - unlike the EULAs they actually allow most things instead of denying it. An example of such license is the GPL - GNU General Public License.

Linux as the Kernel of the Linux Operating System

When we look at a Linux Operating System, its core component is its kernel. The kernel all Linux Operating System use is the Linux kernel, or just Linux. Yes, that's right, the Linux Operating System is called after the kernel, Linux.

Now although all Linux Operating Systems use Linux as their kernel, many of them use a different flavour. This is because the kernel development has several branches. The most important one I call the *vanilla* kernel. This kernel is the main development kernel where most kernel developers work on; every other kernel is based on this kernel. Other kernels introduce features that the vanilla kernel doesn't want yet (or has tossed away in favour of another feature); still, these kernels are fully compatible with the vanilla kernel.

The Linux kernel saw its first light in 1991 and is created (and still maintained) by Linus Torvalds. It grew rapidly (in 1994, version 1.0.0 saw the light) both in size (1.0.0 had more than 175000 lines of code) and in popularity. Over the years, its development model stayed the same: there are few major players in the development who decide what goes in and what stays out of the kernel code, but the majority of contributions happen from several hundreds volunteers (kernel 2.6.21 had contributions from more than 800 individuals).

The latest kernel version at the time of writing is 3.10.7. The first two numbers play the role of the major version, the third number is the minor version (mostly bugfix releases). Sometimes a fourth number is added when a one-off bug fix was needed. The Linux kernel development generally increments the major numbers (most of the time the second number) for functional improvement releases: for every increment, users (and developers) know that the kernel has new features.

Once a new version of the Linux kernel is released, it isn't distributed to all of its users. No, this is where distributions come into play...

Linux Operating Systems: Distributions

If an end user would want to install a Linux Operating System without additional help, he would need to build a Linux kernel himself, build the components of the operating system (like the libraries, end tools ...) and keep track of changes in the free software world (like new versions or security fixes). And although all this is perfectly possible (look for the *Linux From Scratch* project), most users would want something that is a bit more... user friendly.

Enter distributions. A distribution project (like the Gentoo Project) is responsible for a Linux Operating System (the distribution) to such an extend that for the end user, the distribution project is *the* point of contact for his Linux installation.

Distribution projects make choices regarding the software:

- How should the users install the operating system?

  Perhaps users are encouraged to perform as many steps as possible during the installation process (the "distribution" Linux from Scratch [http://www.linuxfromscratch.org/] probably has the most intensive installation process). The very inverse is an installation CD or USB image that doesn't even require any configuration or installation: it just boots the environment and you're ready to start using the Linux distribution.

- What installation options are there (CD, DVD, network, Internet, ... ?)

  Most Linux distributions offer an installation CD/DVD as it is the most popular method for acquiring software. But many other installation options exist. You can install a distribution from a network using net booting (a popular approach in enterprise environments as it makes unattended installations possible) or from within another operating system.

- What software should be available to the user?

  Popular desktop Linux distributions offer a wide range of software to the end users. This allows the distribution to become widely accepted as it fits the needs of many users. However, more advanced distributions exist that focus on a particular market (like set-top boxes for multimedia presentations, firewalls and network management, home automation appliances, ...) and of course, these distributions offer different software titles to the users.

- How is the available software built (specific system, features ...)?

  If a distribution wants the software to run on as many processor types as possible (Pentium, i7, Athlon, Xeon, Itanium, ...) it needs to build the software for a generic platform (say i686) rather than for a specific one (Itanium). Of course, this means that the software doesn't use all features that new processors provide, but the software does run on many more systems.

The same is true for features supported by certain software titles. Some software titles offer optional support for ipv6, ssl, truetype fonts, ... but if you want it, you need to compile this support in the application. Distributions that offer software in a binary format (most distributions do) need to make this choice for their users. More than often, they attempt to offer support for as many features as possible, but not all end-users would need or even want this.

- Is internationalization of the software important?

  Some distributions are targeting specific user groups tied to their language and geography. There are distributions that are fully localized to a specific group (say "Belgian Dutch-speaking users" or "Canadian French-speaking users"), but also distributions that try to offer localization for as many groups as possible.

- How should users update and maintain their system?

  Many distributions offer an automated software update process, but not all distributions offer a live upgrade process (where, once installed, your installation gradually builds up and becomes the latest version of that distribution without any specific actions). Some distributions even require you to boot from the latest installation CD and perform an upgrade step.

- How would a user configure his system?

  If you are a graphical Linux user, you definitely don't want to hear about configuration file editing or command-line actions to be taken. So, you will most likely look for distributions that offer a full graphical interface to configure your system. But some users do like the idea of writing the configuration files directly as it offers the largest flexibility (but also the highest learning curve) and distributions often work on these sentiments. Some distributions don't even allow you to update the configuration files directly as they (re)generate those files anyway (overwriting your changes).

- What is the target user group of the distribution?

  Most desktop distributions target home/office users, but there are distributions that target children or scientists. Some distributions are made for developers and others for elder people. There are distributions for visually impaired people and distributions for people without Internet access.

- What policies does the distribution put on its software?

  Organizations like FSF have a vision on how the (software) world should look like. Many distributions offer a way of implementing these visions. For instance, some distributions only allow software that is licensed under an FSF-approved license. Other distributions allow users to use non-free software. There are distributions that implement a higher security vision in the distribution, offering a more hardened approach to operating systems.

- Should the distribution be freely available?

  Of course, money is often a major decision point as well. Not all distributions are freely downloadable / available on the Internet, although the majority is. But even when the distribution is freely available, it might still be necessary to obtain commercial support, even just for the security updates of the distribution.

- ...

You'll find several distributions in the world; each of those distribution projects answers the questions a bit different from the others. Hence, choosing the right distribution is often a quest where you have to answer many questions before you find the correct distribution.

Of course, when you're starting with Linux, you probably don't have a strong opinion about these questions yet. That's okay because, if you want to start using Linux, you should start with the distribution of which you'll have the best support. Ask around, perhaps you have friends who might help you with Linux. And be honest, what better support is there than personal support?

What is a Distribution?

A distribution is a collection of software (called the packages) bundled together in a coherent set that creates a fully functional environment. The packages contain software titles (build by other projects) and possibly patches (updates) specific for the distribution so that the package integrates better with other packages or blends in better with the overall environment. These packages are usually not just copies of the releases made by the other software projects but contain a lot of logic to fit the software in the global vision of the distribution.

Take KDE for example. KDE is a (graphical) desktop environment which bundles several dozens of smaller tools together. Some distributions provide a pristine KDE installation to their users, others change KDE a bit so that it has a different default look and such.

Another example would be MPlayer, a multimedia player especially known for its broad support of various video formats. However, if you want to view Windows Media Video files (WMV), you need to build in support for the (non-free) win32 codecs. Some distributions provide MPlayer with support for these codecs, others without. Gentoo Linux lets you choose if you want this support or not.

What does a Distribution Provide?

When you want to use a distribution, you *can* (but you don't have to) use tools built by the distribution project to ease several tasks:

- to install the distribution you can use one or more installation tools provided by the distribution project
- to install additional packages on your system you can use one or more software management tools provided by the distribution project
- to configure your system you can use one or more configuration tools provided by the distribution project

I cannot stress enough the importance of the term *can*. You don't have to use the distributions' installation tools (you can always install a distribution differently), you don't have to install software using the software management tools (you can always build and install your software manually) and you don't have to configure your system with the configuration tools (you can always edit the configuration files of the various applications by hand).

Why then does a distribution put all this effort in these tools? Because they make it a lot easier for the user to use his system. Take software installation as an example. If you don't use a software management tool, you need to build the software yourself (which can be different depending on the software you want to build), keep track of updates (both bug fixes and security fixes), make sure you have installed all the dependent software (software this depends on software that which depends on library a, b and c ...) and keep track of the installed files so that your system doesn't clutter up.

Another major addition distributions provide are the software packages themselves. A software package contains a software title (think of the Mozilla Firefox browser) with additional information (such as a description of the software title, category information, depending software and libraries ...) and logic (how to install the software, how to activate certain modules it provides, how to create a menu entry in the graphical environments, how to build the software if it isn't built already, ...). This can result in a complex package, making it one of the reasons why distributions usually cannot provide a new package at the same day the software project itself releases a new version.

For security fixes however, most information and logic stays the same so security fix releases by the software project usually result in a quick security fix release of the software package by the distribution project.

Next to the software that is the distribution, a distribution project provides supporting items:

- documentation about the distribution

- infrastructure where you can download the distribution and its documentation from
- daily package updates for new software
- daily security updates
- support for the distribution (which can be in the form of forums, e-mail support, telephone support or even more commercial contractual support)
- ...

Now, a distribution project is more than all that. By bundling all packaging into a single project, developers can work together to build an operating system that extends the "commercial-grade" operating systems. To do this, most distribution projects have divisions for public relations, user relations, developer relations, release management, documentation and translations, etc.

What is an Architecture?

I haven't talked about architectures yet, but they are important nevertheless. Let me first define the concept of *instruction set*s.

An instruction set of a CPU is the set of commands that that particular CPU understands. These commands perform a plethora on functions such as arithmetic functions, memory operations and flow control. Programs can be written using these functions but usually programmers use a higher level programming language because a program written in this specific language (called the *assembly* language of that CPU) can only be run on that CPU. That, and assembly is so low-level that it is far from easy to write a tool with it. The tools that still use assembly language are compilers (which translate higher-level programming language to assembly), boot loaders (which load an operating system into memory) and some core components of operating systems (the Linux kernel has some assembly code).

Now, every CPU type has a different instruction set. The Intel Pentium IV has a different instruction set than the Intel Pentium III; the Sun UltraSPARC III has a different instruction set than the Sun UltraSPARC IIIi. Still, their instruction sets are very similar. This is because they are in the same family. CPUs of the same family understand a particular instruction set. Software tools built for that instruction set run on all CPUs of that family, but cannot take advantage of the entire instruction set of the CPU they run on.

Families of CPUs are grouped in *architectures*. Architectures are global and represent the concept of an entire system; they describe how disks are accessed, how memory is handled, how the boot process is defined. These define the large, conceptual differences between system. For instance, the Intel compatible range of systems is grouped in the x86 architecture; if you boot such a system, its boot process starts with the BIOS (Basic Input-Output System). Sun Sparc compatible systems are grouped in the sparc architecture; if you boot such a system, its boot process starts with the Boot PROM.

Architectures are important because Linux distributions often support multiple architectures and you should definitely know what architecture your system uses. It is most probably the x86 or amd64 architecture (both are quite equivalent) but you should understand that other architecture exist as well. You will even find tools that are not supported for your architecture even though they are available for your distribution, or some packages will have the latest version available on one architecture and not yet on the others.

Myths Surrounding Linux

Linux is frequently hyped in the media - sometimes with reason, most of the time without. Although I discussed what Linux is previously, a quick recap:

> Linux is a generic term referring to the Linux Operating System, a collection of tools running under the Linux kernel and most of the time offered through a Linux distribution project.

Of course, this is often not clear for users unknown to the world beyond Microsoft Windows. Although the best way of discovering what Linux is is by using Linux, I feel it is important to debunk some myths before I continue with the rest of the book.

Myths

A myth is a story which is popular, but not true. Myths surrounding Linux will always exist. The next few sections try to offer my ideas behind many of these myths...

Linux is hard to install

It is always possible for someone to point to a Linux distribution that is difficult to install. The Linux From Scratch "distribution" is actually a document explaining the entire process for setting up a Linux distribution by building compilers, building software, placing files, etc. Yes, this is hard and might even be difficult if the documentation wasn't up to date.

However, many distributions (most of them even) are simple to install. They offer the same installation approach as other operating systems (including Microsoft Windows) together with online help (on-screen help) and offline help (installation guides). Some distributions can even be installed with as little as two or three questions, and you can even use Linux without having to install it at all.

There is no support for Linux

There were days that Linux had no commercial support, but that was in the previous century. You can now obtain the Linux operating system from major software vendors such as Novell or RedHat (with support), or use a freely downloadable Linux distribution and get a contract with a company that offers support for that distribution.

All distributions offer excellent free support as well (something I'll talk about in the next few chapters) and many have an active security follow-up, resulting in quick security fixes as soon as a vulnerability is found or reported. There is often no need for a desktop user to obtain commercial support as the freely available support channels offer a major advantage compared to some other, proprietary operating systems.

Linux is free software, so security holes are easily found

Actually, because it is free software, security holes are far more difficult to remain in the source code. There are too many eyes watching the source code and many free software projects have a very active developer community that checks and rechecks source code changes over and over again before they are pushed to the end user.

Linux is non-graphical

The Linux kernel is not a graphical kernel, but the tools that run beneath the Linux kernel can be graphical. Even more, most distributions offer a full graphical interface for every possible aspect of the operating system: it boots graphically, you work graphically, you install software graphically, you even troubleshoot issues graphically. Although you can work with a command line exclusively, most distributions focus on the graphical environment.

This book is not a good example regarding this myth as it focuses on the command-line. However, that is because of the personal preference of the author.

I cannot run my software under Linux

For many Microsoft Windows titles, this is true. But there is almost certainly software available in Linux that offers the same features as the software you are referring to. Some software even *is* available

for Linux: the popular browsers Firefox and Chrome are two examples, the freely available office suite OpenOffice.org is another.

There are also Windows emulators and libraries that offer an interface allowing Microsoft Windows applications to run within Linux. I don't recommend using this software for every possible software title though. It is more of a last resort in case you definitely require a certain software title but already perform the majority of your work within Linux.

Linux is secure

This is also a myth. Linux is no more secure than Microsoft Windows or Apple's Mac OS X. Security is more than the sum of all vulnerabilities in software. It is based upon the competence of the user, the administrator, the configuration of the system and more.

Linux can be made very secure: there are distributions that focus on security intensively through additional settings, kernel configurations, software choices and more. But you don't need such a distribution if you want to have a secure Linux system. Better is to read the security documentation of your distribution, make sure that you regularly update your system, don't start software you don't need or visit sites you know aren't legit.

Linux is too fragmented to ever become a wider player

Many groups refer to Linux as being fragmented because there are so many Linux distributions. However, a user of one distribution can easily work with users of other distributions (no issue here). A user of one distribution can also help users of other distributions, because their software is still the same (no issue here either). Even more, software created on one distribution runs perfectly on another distribution (no issue here). The widespread availability of distributions is a strength, not a weakness, as it offers more choice (and more expertise) to the end user.

Perhaps people are referring to the various Linux kernel trees that exist. Yet, all these trees are based upon the same mainline kernel (often called the "vanilla kernel") and every time the mainline kernel brings out a new version, these trees update their own code so branches are never lagging behind. The additional trees that exist are there because of development purposes (additional patches for unsupported hardware before it is merged with the mainline kernel, additional patches for specific virtualisation solutions that are otherwise incompatible or cannot be merged due to license issues, additional patches that are too intrusive and will take a while before they are stabilized, etc.)

Or perhaps people are referring to the various graphical environments (like KDE and GNOME). Yet, they do not speak about the interoperability between those graphical environments (you can run KDE applications in GNOME and vice versa), the standards that this diversity creates (standards on dealing with file formats, menu entries, object linking and more), and more.

Controlled fragmentation is what Linux (and free software in general) offers. Controlled, because it is matched with open standards and free specifications that are well documented and that all software adheres to. Fragmented because the community wants to offer more choices to the end users.

Linux is an alternative for Microsoft Windows

Linux isn't an alternative, but a different operating system. There's a difference between the terms. Alternatives try to offer the same functionality and interface, but using different means. Linux is a different operating system, because it doesn't strive to offer the same functionality or interface of Microsoft Windows.

Linux is anti-Microsoft

It isn't because people that have certain feelings about Microsoft are often using Linux, that Linux is anti-Microsoft. The Linux operating system wants nothing more than be fully interoperable with any other operating system. Software projects most definitely want their software to run on any operating system, not only Microsoft Windows or Linux.

Weaknesses

Yet not all information spread around are myths. Some are real weaknesses that Linux still needs to work on.

Linux has little support for games

True. Although there are many free software games around, most games are developed for Microsoft Windows exclusively, and not all games can be run using emulators or libraries like WINE within Linux (luckily, many are). It is hard to ask game developers to develop for Linux as most developers focus their endeavours on libraries (like DirectX) that are only available for Microsoft Windows.

However, another trend is also emerging: more and more games are only being released on consoles, dropping the PC environment altogether. I personally don't know how games will evolve in the future, but I think that real action games will focus on game consoles more.

Still, gaming is a sore weak spot of the Linux operating system.

Recent hardware isn't quickly adopted within Linux

If the vendor of the hardware doesn't offer Linux drivers, then it does take a while before the hardware support is brought within the Linux kernel. However, this is not a process spanning multiple years, but rather months. Chances are that a brand-new graphic card / sound card is supported within 3 to 6 months after being released.

The same is true for wireless network cards. Whereas this was a weakness previously, support for wireless network cards is now well integrated within the community. A major reason here is that most vendors are now officially supporting their wireless chip set for Linux, offering drivers and documentation.

Exercises

1. Create a list of Linux distributions you have heard of and check, for every one of them, how they perform in the fields you find important (for instance, availability of documentation, translations, support for specific hardware, multimedia, ...).

2. List 7 CPU architectures.

3. Why are new kernel releases not distributed to the end user immediately? What role do distributions play in this process?

Further Resources

- Why Open Source / Free Software [http://www.dwheeler.com/oss_fs_why.html], by David A. Wheeler - a paper on using Open Source Software / Free Software (OSS/FS).

- Distrowatch [http://www.distrowatch.com], a popular site that attempts to track all available Linux distributions and has weekly news coverage.

Chapter 2. How does Free Software affect Linux?

Introduction

The Linux OS has become increasingly popular mainly due to the freedom it allows (and of course also the low or zero-fee price of the entire operating system). In this chapter we see how these freedoms come to life and how they are protected and sustained.

We also take a look at the development model used by free software projects in general because it is a major result of said freedoms, one that makes free software projects often more interesting than closed-source commercial software projects. The development model is also one of the major strengths of free software.

Free Software

If we take a step back from all technical aspects, Linux differs from the closed-source commercial software in an important aspect: licensing. Licensing is what drives free software...

What are Software Licenses?

Software is someone's intellectual property. Intellectual property is a heavy word that shouldn't be interpreted to anything else than the result of some effort to create something that is not a plain copy. If you write some text, the resulting text is your intellectual property (unless you've copied it from somewhere).

Intellectual property is protected by law. Copyright protects your intellectual property by prohibiting others to copy, adapt, reproduce and/or redistribute your ``thing'' without your consent. Mind you though that not every intellectual property is copyright protected and copyright differs from country to country. An example of intellectual property that isn't copyright protected is a mathematical method: even though the inventor of the method had to ponder years and years on it, his method isn't copyright protected (but if he wrote a text about this method, the text itself is). Copyright is automatically assigned: it doesn't cost you anything and it is broadly accepted.

Another protection is a patent. Patents are (or should be) granted to new inventions who are not known to the public at the time of the patent request. Patents are often used to protect intellectual property that isn't protected by the copyright: methods for doing stuff (including medical compositions). Sadly, the industry is often abusing patents for much more when they have a patent with a broad action field: the patent covers too much, allowing the company to force others not to use a method they actually do have the right to use. Also, both the request and the patent grant are very costly and only larger companies have the abilities to obtain (and protect) several patents. Smaller companies or individuals don't have the means to obtain a patent, let alone protect themselves in a court because they might have used a method that is described in one or more patents.

I use the word *abuse* because companies often get patents for methods that are broadly used or are so silly that you'd wonder what patent office (patent requests are - or should be - checked for their validity before they are granted) has granted those patents.

I'll abstain from elaborating on this (politically sensitive) topic more and move on to *software licenses*. A software license is a contract between you, the software user, and the software copyright owner. It tells you what you can and cannot do with the software. Any software that is not licensed is fully copyright protected, meaning you shouldn't even have it, let alone run it.

Most commercial-grade licenses are often called the EULAs, or End User License Agreements. They usually say what you are allowed to do with the software (often including what you are allowed to use

the software for). The EULAs more often stress what is denied rather than allow anything. One of the many topics is redistribution of the software. Most EULAs explicitly disallow redistribution.

Linux (and free software in general) is different. The accompanying license grants you the right to copy the software, obtain the source code, modify it and redistribute it (with or without modifications) and even sell it. Because there are many variations possible there are many popular licenses.

What Licenses Exist?

I'll list a few of the more popular licenses here, but be advised, there are more than 800 licenses around. Many of those licenses are quite similar (or are exactly the same) and the free software community should start to consolidate all those licenses in a much smaller set. Sadly, they haven't done so yet. Luckily, the 90-10 rule here applies: 90% of all free software uses 10% of the free software (or other) licenses. The other licenses are only marginally used, sometimes just for a single application.

Public Domain

When software is placed under the public domain, you're free to do whatever you want with it: the author waves any right he can to allow for full freedom of his software.

MIT License and some BSD-like Licenses

The MIT license and some BSD-like licenses are almost like the public domain, but ask you to keep the copyright notice intact. This is a very popular license because the author allows you to do whatever you want as long as you keep his name on the product copyright notice as well.

GPL

The GNU Public License is the most widely used free software license, but for some people also the most restrictive free software license. The GPL tells you that you can do whatever you want with the software, as long as you provide the source code of your modifications to whoever you distributed the modified version to and as long as this modification is under the GPL as well.

The Linux kernel is GPL licensed.

OSI Approved Licenses

An OSI approved license is a license that adheres to the *Open Source Definition* written down by the *Open Source Initiative* of which the following points are a free interpretation:

- free redistribution
- source code available
- modifications are allowed (including redistribution)
- no discrimination (people, fields ...)

FSF Approved Licenses

An FSF approved license adheres to the *Free Software* definition written down by the *Free Software Foundation* of which the following points are the core of the definition:

You should be free to ...

- run the program for any purpose
- study how the program works and adapt it to your needs
- redistribute copies

- improve the program and release your changes to the public

Free Software isn't Non-Commercial

Free software is often perceived to be a pure hobbyist project: it would not be commercially viable to bring free software to the enterprise world. After all, if software is freely available, what kind of profit could a company make from it. Nothing could be further from the truth...

It is true that free software requires a different look on software in a commercial environment (including companies). Companies who *use* software want to be assured that they have support for the software when things go wrong. They often close (costly) support contracts with the software company where service level agreements (abbreviated to SLAs) are defined. Based on these contracts, the company has the assurance that if certain services become unavailable, the supporting company will do whatever it can to bring the service back or, in some occasions, compensate the financial damage that the downfall has caused.

Most of the time, these support contracts are closed with the software company itself because it has the most knowledge of the software (as it is probably the only company with access to the software code). Sadly, as good as this reason is, companies don't look at free software ``because there is no support''. This isn't true; support for free software is still (commercially) available, but most of the time not from the creators themselves. And although this scares the companies, the reason why this support is still as good as with off-the-shelf software remains the same: the supporting company has access to the source code of the tool and has professional knowledge about the tool. It probably has developers in the software project itself.

Companies that *sell* software are of course often against free software. When these companies major income depends on the sales of their software, it would not be viable to make the software free. If they would, competing companies would have full access to the source code and improve their own product with it.

I don't think this is a disadvantage though. Software companies should use their main strength: knowledge about the tool. As mentioned before, other companies often want to close support contracts to ensure the service that the software delivers; if the software company creates free software, this wouldn't change. For many software companies, support contracts are the main source of income.

It is still possible to sell free software; some pioneering companies are payed to made modifications to free software because companies don't have the resources to do so themselves. These companies can keep the modifications private if the free software license allows this) but can also bring these modifications to the public by contributing it to the software project itself.

A major proof of this is the acceptance of free software by major software players such as Sun Microsystems and IBM, and the emergence of new software players that build their business upon free software, such as RedHat or MySQL (recently acquired by Sun Microsystems). The latter company uses a dual-licensed software approach: the MySQL source code is available in two licenses, a free software one for the public and a more closed one for companies who want support from MySQL itself. Using a dual-licensed approach allows the company to support a fixed state of their product while keeping the software free. Supporting a fixed state of the product is of course much easier than to support the software in general.

However, don't think that every free software project is enterprise-ready or that you will be able to find (paid) support for every software project. You should carefully check out every software title you want to use if you want to use software, free or not. For end users, distributions help you to pick software. If a distribution packages a certain software title, it feels that the software title is stable and well supported.

So Linux is Free?

Yes, Linux is free. It is certainly free in the sense of ``free speech'' and although most software titles are also free in the sense of ``free beer'', you shouldn't be surprised to see distributions you can or have

to pay for. In that case, you can be paying for the software medium (the burned DVD), accompanying printed documentation, 30-day installation and usage support or for the resources that the distribution has to acquire itself (like infrastructure).

Most distributions have free downloads with online documentation and wonderful community support (active mailing lists or Internet fora), which is why Linux is that popular: you can download, install and use several distributions to decide which one is best for you. You can try the software (without loosing any functionality) and you don't even have to pay for it to continue using it (as is the case with shareware). Gentoo is one of those distribution projects. Such distributions get their financial backing (for infrastructure and organisational needs, including juridical support and bureaucratic paperwork) from user donations or sales of pressed DVDs. Companies also tend to support distributions financially or with hardware / bandwidth donations.

Some distributions are only available when you pay for it. In that case you often pay for the support or for additional software in the distribution which isn't freely available. A popular distribution is RedHat Enterprise Linux, a Linux distribution specifically targeting companies who want to set up Linux servers. You don't just pay for the support, but also for the resources that RedHat has put in the distribution to make it certified for other software (such as Oracle and SAP) so that you can run (with support from the software company) this software on your RHEL installations.

It is important however to understand that distribution projects only develop a very small part of the software that you install on your system. Most software comes from other free software projects and these projects often don't get gifts from the distribution projects. Nonetheless they do face the same problems as any other (larger) free software project: bureaucratic paperwork, juridical support, infrastructure needs, ... So it comes to no surprise that these projects also have the same income streams as the distribution projects: user gifts, commercial sponsorship and software / support sales.

Development Model

Due to the nature of free software projects, you'll find that it has quite some differences with closed-source commercial, off the shelf software...

Multi-Project Development

One distribution provides an aggregation of software. Each of those software titles is built by a software project which usually differs from the distribution project. Hence, when you install a distribution on your system, it contains software from hundreds of software projects around the world.

So to obtain support for a flaw you found, or an issue you come across, the first place to seek support would be the distribution, but chances are that the distribution will put the support question *upstream*, meaning that it forwards the request to the software project that develops the software you have an issue with.

Transparent Development

Free software is usually developed transparently: if you are interested in the development of your favourite software title, you can quickly find out how its development works and how to participate.

Usually, software projects use a *concurrent versions system* such as CVS or SVN to keep the source code in. Such systems allow for dozens (or even hundreds) of developers to work on the same source code simultaneously and keep track of all changes that have happened (so they can easily be reverted). This isn't just for free software projects - almost all software projects use such a system. However, free software projects usually allow non-developers to see the progress of the development by giving them read-only access to the system. This way, you can track every change to the software personally.

To discuss the future of the software, or to take software design decisions, most free software projects can't use real-life meetings: their developers are scattered around the world. A solution to this problem are communication systems such as mailing lists, IRC (chat) or forums (Internet or Usenet). Most

of these communication systems are also open for non-developers to participate in the discussions, meaning that end users have direct communication with developers.

The latter has a major advantage: changes requested by the users are directly communicated to the developers so that misinterpretation is less frequent, allowing for projects to update their software more accurate and frequent.

Fast Release Cycles

Larger free software projects have hundreds of contributors and several dozens of developers. Those developers are very motivated to work on the software by passion. If they weren't, they wouldn't be working on the software as there usually is no other incentive to work for (such as a nice pay check) although it must be said that there are software projects (and they aren't small in numbers) who have paid developers as well. As a result, the software is quickly progressing and new features are added quickly (some projects even have new features on an almost daily basis).

To make sure that new features and fixes are tested properly, software development snapshots are communicated to a broad community of testers and stable snapshots are often released to the general public as a new release of the software. Different release types are commonly used in free software environments:

- *nightly snapshots* are extracts of the source code at a certain period in time which are built and put online for everyone to use. These releases are automatically generated and are bleeding-edge as they represent the state of the software title only a few moments ago. They are highly experimental and only meant for developers or experienced contributors
- *development releases* are intermediate releases, similar to nightly snapshots, but somewhat more coordinated by the developers. They usually have a ChangeLog which lists the changes in it since the previous release. Such releases are meant for experienced contributors and testers who don't mind the software to be broken from time to time.
- *beta releases* contain a preliminary vision of how the final release will look like. It might not be fully stable or complete but individuals who don't participate in the frequent tests can try and see if the new release would still work for them and contain the fixes they requested. Beta releases are also important for distributions as they can now start developing packages for the software so that they are ready when the final release of the software is made.
- *release candidates* are proposals for final releases. They contain the software such as the developers would like to release it. They now wait for a certain period so that the testers and general public can run their tests to ensure no bugs are in it any more. New features aren't added to the software now, only bug fixes. When no new (or major) bugs are found, the release candidate is converted to a new release
- *stable release* are the final releases of the entire development process. These releases are now used by the users and distributions and the entire development process can start over.

Stable releases also tend to be released in specific gradations, reflected by their version number. A popular numbering scheme is x.y.z where:

- x is the major version; this version number is only updated when the software has been substantially changed. Often such releases also require all packages that depend on it to be updated as well because they might use features or libraries that are changed.
- y is the minor version; this version number is updated every time the software has been updated with lots of new features
- z is the bugfix version; this version number is updated whenever mainly bug fixes have been added to the software

As an example I'll list the release dates for the KDE 4.9 release. Since KDE is a complete graphical environment its release cycle is ``slower'' than others. Yet if you compare it with the release cycle

of for instance Microsoft Windows its still blazingly fast. Of course, that would be like comparing apples with glass...

- 2012-05-30: KDE 4.9 beta1 is released
- 2012-06-14: KDE 4.9 beta2 is released
- 2012-06-27: KDE 4.9 release candidate 1 is released
- 2012-07-11: KDE 4.9 release candidate 2 is released
- 2012-08-01: KDE 4.9 released
- 2012-09-04: KDE 4.9.1 released
- 2012-10-02: KDE 4.9.2 released
- 2012-11-06: KDE 4.9.3 released
- 2012-12-04: KDE 4.9.4 released
- 2013-01-02: KDE 4.9.5 released

Just for your information, KDE 4.10 beta 1 is released on November 21st, 2012, merely 6 months after KDE 4.9's beta release, and KDE 4.10 (now called "KDE Software Compilation" as the applications for KDE themselves are following their own release cycle) has been released on February 6th, 2013 - 6 months after KDE 4.9.

Large Documentation Base

Because the project often can't deliver human, paid support for the software, its success is largely based on the documentation the project delivers. If the accompanying documentation contains all information about the software, experienced or independent users can find all user related answers in the documentation.

Free software projects usually have high profile documentation, often better than the online available documentation of closed-source off the shelf software. Many larger projects even have all this documentation available in several languages. And if you don't find your answer in the project documentation, chances are that one or more users have written independent guides on the software elsewhere.

There are many sites on the internet that link to the various documentation resources and the same problem as with free software itself arises: often you have too many resources making it harder to find the correct document to guide you through your end user experience of the software. However, unlike the plethora on software titles around (making it difficult to find the right software for the right job) it is easier for a user to know if documentation is good or not so there is no need for a ``documentation distribution".

Software Life Cycle

If you buy software of an unknown, smaller company, you have the chance that after a number of years, the company doesn't exist any more or is taken over and doesn't support that software since. Something similar is true with free software: if the project decides that there aren't enough resources to continue the development of the software (usually due to a shortage on developers) it can stop the development of the software, usually resulting in a drop of support from users as well.

However, unlike the case of the software company, the free software source code remains available to the public. If you desperately need the software to work for you, you can just pick the source code and continue the development of it yourself (or pay others to do it for you). You're also confident that the software will remain free.

If at any time all the copyright owners of the free software decide that the software falls under a different license which you don't agree after, you can take the source code of the moment right before the copyright holders decided to switch the licenses and continue the development under that license (as that software is still under the original license and not the new one). This process (where a group of developers disagree with the development plans of the software and start a new project based on the same source code) is called *forking* the project.

A well known example of such a fork is the creation of the X.org project, a fork of the XFree86 project which at a certain point in time decided to change their license. The license change wasn't the only reason for that fork: some developers were also unhappy with the development policy on new features and the development pace. Both projects are currently still around although X.org is now the most popular one.

Open Standards

Because so many projects are involved, it is important that each project uses standards as much as possible. Only by complying to open standards can projects easily and efficiently work together. Next are a few important standards or well perceived specifications in the free software world.

File system Hierarchy Standard

The first standard I discuss is the *File system Hierarchy Standard*, abbreviated to FHS. This standard is used by almost all distributions and discusses the file locations on a Linux file system. One can read the FHS online at `http://www.pathname.com/fhs/` [http://www.pathname.com/fhs] but many other resources describe the FHS layout as well.

The file system layout for Unix/Linux is quite different from the file system layout as seen from within Microsoft Windows. Instead of marking partitions by a drive letter, Unix/Linux sees a file system as a tree-like structure, starting with a root and building up through directories and files. You could say that the branches in the structure are the directories and the leaves are the files. If you think you have not encountered a Unix/Linux file system before, think again: URLs that you use on the Internet are based upon this structure. For instance, the URL http://www.gentoo.org/doc/en/faq.xml denotes the file called `faq.xml` which can be found on the server of www.gentoo.org [http://www.gentoo.org], in the directory `/doc/en`. So, / is the root, "doc" is a branch of this root and "en" is a branch of "doc".

Distributions that adhere to the FHS allow their Linux users to easily switch between distributions: the file system structure remains the same so navigation between folders, device files ... doesn't change. It also enables independent packagers to create packages for several distributions at once (as long as the distributions use the same package format). But foremost, it allows Linux users of one distribution to help users of other distributions as there isn't actually any difference between their file system layouts.

The current version of this standard is 2.3, released on January 29th, 2004.

Linux Standard Base

The *Linux Standard Base*, or LSB sets the layout, binary compatibility, required libraries, required commands and more for a Linux operating system. If a distribution adheres to the LSB standard it can install, run and maintain LSB compliant (software) packages.

Distributions should adhere to the LSB if they want to ensure that they don't deviate from a good Linux standard. As a consequence, the LSB is an effort to ensure that distributions stay similar with regards to libraries, commands ... or in overall, user experience. It is a good effort to ensure that no fragmentation occurs in the Linux world.

Because the LSB is a broad standard, it comprises of other standards, including the aforementioned FHS but also the *Single Unix Specification* (SUS) which defines how a Unix system should be. However, one cannot say that his Linux operating system is Unix because he would need to certify

the OS (which requires serious financial support) and this certification wouldn't last long because the Linux OS changes often.

One of LSBs' largest advantages is that ISVs (Independent Software Vendors) such as Oracle, IBM, Sybase ... can package their software in an LSB-compatible software package which can then be installed on any LSB-compliant distribution.

Free Desktop Specifications

On `http://www.freedesktop.org` you'll find a set of desktop specifications that are well known in the free software community. Although they aren't standards (as freedesktop is no standards body and the specifications haven't been converted into OASIS or ISO standards) many distributions adhere to them.

These specifications define how menu entries are created and maintained, where icons should reside, but also how drag and drop between different libraries (most notably Qt and GTK+, the graphical libraries for KDE and GNOME) should be made possible.

Exercises

1. What is the difference between GPLv2 and GPLv3?

2. Part of LSBs standard is the ELF or Executable and Linking Format, the binary format for executable, compiled code used by various Linux/Unix distributions. Can you find other operating systems that support the ELF format beyond Linux/Unix?

3. Some people see fast releases as a weakness in the free software community: users are "forced" to upgrade their software more often and even though it is free, it still takes time (and sometimes headaches) to upgrade the software this often. Some distributions tend to help those users by offering stable (both in stability and in version releases) software only. How is this possible?

4. How is it possible that many distributions allow you to upgrade to the latest version without needing an installation CD or re-installation from scratch?

Further Resources

- The Cathedral and The Bazaar [http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/], by Eric Steven Raymond - an essay on two different development models used in the Free Software community.

- Foundation for a Free Information Infrastructure [http://www.ffii.org], a NPO dedicated to establishing a free market in information technology.

- Fighting Software Patents [http://www.gnu.org/philosophy/fighting-software-patents.html], by Richard Stallman - GNUs vision on software patents.

Chapter 3. The Role of the Community

Introduction

A very important asset of free software is the free software community. Just like with any technology or concept, free software has adepts that defend and promote free software to great extend. The free software community itself is very vivid and eager to help others in exploring the wonderful world of free software...

Communities

Free software communities are similar to real communities, but with the Internet as main communication channel. Hence, these communities aren't clustered in space like real life communities would, but are scattered throughout the world. Nevertheless, the Internet ensures that participants of a community, even when they are light years (figure of speech) apart, talk to each other the same way as neighbours do.

The Internet is a great asset for these communities: you are not judged based on the colour of your skin, your age or your looks. What matters is how you communicate with others, how you present yourself and how you react in discussions. Debates in a community can often become quite vivid, especially when the subject is one where facts aren't sufficient to provide good answers. And when these discussions change from debates into almost insulting fights, a flame war is born.

In flame wars, facts and reason are often far away. You should definitely try to avoid flame wars for discussions where decisions have to be made, but it is impossible to really prevent them as they are the result of people who have an active interest in a subject they are eager to defend, especially when there is no clear answer to the question that started the flame war.

Examples of such flame wars are ``What is the best Linux distribution?'' or ``What text editor should I choose?'' because these questions don't have clear answers: the best distribution for one person might be the worst for another, and there are many text editors around. In latin one would say ``de gustibus et coloribus non est disputandum'' (one shouldn't argue about tastes and colours) and this is very true for these kind of questions.

When you don't have a choice, flame wars don't exist: you cannot compare one product with itself. But in the free software world, choice is an important concept. You have the choice between many free operating systems (next to Linux you have many BSD flavors, Sun Solaris 10 and even less popular but promising operating systems like the GNU Hurd), distributions (there are over a hundred distributions around), graphical environments (not a single day goes by without battles about GNOME versus KDE), office suites, etc.

An often debated subject is ``the best distribution'' and although this book might seem a bit biased on the subject the best answer I can give you is that there is no best distribution, at least not generally speaking. The meaning of the term ``best'' is judged by people who have personal preferences about their operating system. And many of these people defend their best distribution very vividly.

Distribution communities are very active, mostly because they are quite large. The Gentoo community for instance is known for its responsiveness: the Gentoo chat channel is always alive (with more than 800 participants at any time) as is its forum (with more than a thousand posts per day) and mailing lists. Of course, general flame wars on distributions are often on more neutral grounds, but heated discussions on other topics are a daily routine.

For this reason, most communities have people who keep the discussions sane and prevent flame wars from growing too much. People who try to induce flame wars on the communication channels (called

trolls) are taken care of by these operators: channel operators can kick or even ban such people from the chat channel, mailing list operators remove these people from the list and forum operators remove the profiles of these users. You can safely say these people are the police of the community.

Local Communities

A specific type of community is one which is local in space. Such communities often organize meetings (conferences, talks, barbecues, ...) and offer help to people local to the location where the community is hosted.

LUGs (Linux User Groups) are successful examples of such communities: these groups aggregate together, debating on the evolution in the Linux world and help others with Linux installations (Linux Install Fests are local meetings that offer help in deploying your favourite Linux distribution on your system). You might find a LUG very close by.

Many LUGs offer various services to their users which is often unseen in communities for commercial software. Moreover, many LUGs offer these services free-of-charge:

- individual, on-site help with installation, configuration and maintenance of a Linux distribution or other free software
- courses, talks and presentations offering you more insight in available Free Software
- specific documentation tailored to the needs of its own users

If you have some time to spare, I really recommend to join a local LUG - even if you are not searching for help, you can still offer your own expertise to others and make connections (yes, social networking is important).

Online Communities

When people want to discuss a particular software topic or distribution, online communities are often formed. These communities do not (or to a less extend) organize meetings at a specific location (often called "in real life") but rather use the Internet as the meeting place ("online" meetings).

Online communities have the advantage that its members can be anywhere in the world and just like LUGs, they still offer services to its users, also most of the time free-of-charge:

- online help with installation, configuration and maintenance of the software

  In particular cases, communities can even offer interactive help through technologies such as SSH (Secure SHell - allows users to log on and work on another machine) and VNC (Virtual Network Computing - allows users to graphically log on and work on another machine, or see read-only sessions).

- courses and online presentations
- documentation, more specialized to the software title but often also localised (translated)

This is possible thanks to the various technologies available on the Internet, including

- Wiki (online collaboration software for developing documentation) software has become quite popular for developing and releasing documentation. The use of wiki's allows users to edit existing documentation or author new documentation online (with a simple browser) and the results of their editing is immediately visible to others.
- Online (web)forums, where people can participate in discussions by placing messages and reacting to other messages. The advantage of web forums is that they are accessible through your web

browser (which most firewalls still allow), can be consulted after the discussion has long been closed and where messages can be extended with images, attachments and formatted text.

- Mailinglists, which is similar (function-wise) to web forums, but then organized through e-mail. People subscribe to a mailing list and then receive all mails sent to that mailing list to their personal mailbox. Replies to these mails are sent back to the mailing lists where they are again distributed to all mailing list participants. Mailinglists are quite popular in free software communities as they are easily moderated and can be filtered. Also, mails often reach people faster than messages on a web forum so you could see a mailing list as a faster discussion medium.
- IRC (Internet Relay Chat) is a way of communicating with many people interactively. Most people know Instant Messaging software such as MSN or Google Talk. Well, IRC is somewhat older but still very much used as it supports chat rooms where several hundreds of people can participate. IRC is the fastest medium for participating in discussions and can be seen as a method for creating "online" meetings.
- People-centric social media, such as Google Plus or Facebook, where likeminded people collaborate and discuss their favorite topics. The advantage of these is that they are much better integrated with recent technological evolutions; people with modern cellphones (smartphones) are continuously available and can quickly interact with whatever is happening on the communities.

Support

Communities often perform the role of support people: if you have a question about their software project they are eager to answer and help. If you think the software is insufficient, they will help you expand it or have it work together with other tools (or even redirect you to other software projects if they feel you want something out of their favourite tool that the tool isn't made for).

Support can be given on many levels...

Documentation Guides

A documentation guide is often created with one goal: describe how to do something with the tool. Such guides are therefore often called HOWTOs. Much work is put in such HOWTOs because they should be correct, well formed but also complete. The better the HOWTO, the lesser questions are asked after reading it. If you ask the community how to perform a certain action and the action is described in such a HOWTO, you'll be redirected to that HOWTO (sometimes with a more crude reference to the RTFM term, or ``Read The Fucking Manual'' - although the third term is also often read as ``Fine'').

Other types of documentation are FAQs (*Frequently Asked Questions*) which are generally very small HOWTOs or answers to conceptual questions rather than technical ones. When you're new to a certain tool it is very interesting to read through the FAQs before you ask your question. Not only are chances high that you find your answer, you might find out more about the tool which can be very interesting.

Some communities also offer a knowledge base. Such systems can be seen as an aggregation of questions and answers, but unlike FAQs they might not be frequently asked. Knowledge bases often offer support solutions to specific setups.

Internet and Usenet Forums

Internet forums (web based) or Usenet forums (newsgroups) are a more interactive approach to obtain support. Internet forums have the additional advantage that you can add specific formatting in your questions: you can show command code, exceptions or errors better than in plain text. You can even include screen shots. These forums allow for any user to be helped quite fast: forums are read by many and the interface is simple enough to quickly see the new topics.

An additional advantage of internet forums is that, once a question has been asked and answered, it is stored in the database of the forum. Hence, the entire forum can be seen as a knowledge base with a

multitude of answers. Very popular topics are often made sticky, meaning that the topic remains on top even when no further discussion happens on it, increasing the chance that new users read the topic.

Usenet forums (or newsgroups) are another popular approach to support although it must be said that newsgroups are not used that often for free software tools. Usually you'll find a newsgroup when the project itself doesn't provide a forum (anyone can launch a new newsgroup) although it does happen that internet forums and usenet forums are linked: posts in one forum are merged with the other.

Mailinglists

A more direct approach are mailing lists, e-mail addresses where several dozens (or even hundreds) individuals listen to. A mailing list is often perceived to be a bit faster than forums because many developers frequent mailing lists but not forums due to the ease of use: mailing lists result in plain e-mails which can be easily filtered.

Most mailing lists are archived as well, allowing you to skim through the older topics in the list. Whereas forums are usually pure for user experience, mailing lists are used as the primary communication channel for development purposes. Some projects also have internal development mailing lists which aren't readable to the public. This isn't because they want to hide development stuff from the users: such mailing lists are used to communicate security issues, personal information (including account information) but also to talk about topics that are juridically difficult to defend if they are made public.

Chat

Chatting is almost the most direct form of communicating with each other. Many free software projects use IRC (Internet Relay Chat) as a central communication channel. Users can be quickly helped through IRC while developers can talk and discuss changes quickly.

Chat channels can be very popular. Gentoo's main chat channel (#gentoo on the freenode network) has between 800 and 1000 participants at any time.

Real-life Meetings

Once in a while, developer groups come together for real-life support or to discuss the evolution of their software. In many cases, real-life meetings offer a way for people to get hands-on, interactive help. We have talked about LUG meetings (where real-life meetings are often held) but also software communities have real-life meetings. Many of these meetings offer a way for developers to meet each other (for the first time), discuss topics and learn from each other.

In some cases, *hack fest*s are organized. During these meetings, developers aggregate together with a single goal: to develop new features or remove bugs from the software. Although this can well be done offline, hack fests allow developers to communicate freely and help other developers with their problems. Meeting in real life allows developers to easily show the problem they have (some problems can be difficult or too time consuming to write down).

Conferences

In the Free Software world, conferences are often organized. During these conferences

- talks are given about certain software titles (design, features, evolution, ...) or projects (infrastructure, offered services, used technologies, ...)
- booths are organized where projects can show themselves to the wide(r) public. Distributions frequently use booths to hand out installation CD/DVDs and show systems running the distribution.
- companies offer information on how they use (or develop) free software (and sometimes recruit developers)

FOSDEM

FOSDEM, or the *Free and Open Source Developers European Meeting*, takes place in Brussels, Belgium at the beginning of each year (around mid-february). During this conference, talks are given about coding and development of software, but you'll also find booths about various software projects/ distributions and developer rooms (where a single project can offer talks about project-specific topics).

FOSDEM is held during two days and has become a major conference in the Free Software community, especially in Europe as many other conferences are held in the USA.

FOSS.IN

FOSS.IN, or the *Free and Open Source Software conference in India*, is one of Asia's largest FOSS conferences. It occurs at the end of every year in Bangalore, India, featuring talks, discussions, workshops, meetings and more from international speakers, users and developers.

LinuxTag

LinuxTag is a free software exposition with primary focus on the Linux-based operating systems and solutions. Unlike FOSDEM, LinuxTag focuses more on the integration of Linux (and free software) in larger environments, offering booths to both commercial companies and non-commercial organisations.

It's slogan is "Where .COM meets .ORG". You can visit LinuxTag around spring every year.

Exercises

1. Try to find the online discussion methods (web forum, mailing lists, IRC) offered by the Gentoo Linux distribution.

Resources

A few more free software conferences:

- The Ottawa Linux Symposium [http://www.linuxsymposium.org] is held every year in Ottawa, Canada during summer break.
- Linux Kongress [http://www.linux-kongress.org] has almost always been held in Germany although a single instance was in Cambridge, England.
- Linux.conf.au [http://linux.conf.au/] is hosted in Australia in the beginning of every year
- Ohio Linux Fest [http://www.ohiolinux.org/] is held in Ohio every fall.
- Linux Fest Northwest [http://www.linuxfestnorthwest.org/] is held in Washington every spring.
- SCaLE (Southern California Linux Expo) [http://scale7x.socallinuxexpo.org/] is held late winter.
- Ontario Linux Fest [http://onlinux.ca/]
- LinuxWorld Conference and Expo [http://www.linuxworldexpo.com/]
- Freed.IN [http://freed.in/]

Chapter 4. Running Linux

Introduction

In order to work with Linux you need access to a Linux environment. The best environment is an installed Linux distribution on your system, but you can use other environments such as Linux Live CDs (bootable CDs with a ready-to-run operating system on it). Installing Linux is very distribution-specific and Gentoo's installation procedure is well documented on the Gentoo site [http://www.gentoo.org/doc/en/handbook]. I will not discuss the installation procedure yet. Later in this book, I will focus on the command-line Gentoo installation because then you'll have a much better view on the various installation steps. Right now, I'm going to talk about a few Linux-specific topics which are quite important to understand, but without going in too much detail: almost every topic will be discussed in great length later.

Users who are not used to working with Linux or Unix will find that it is quite different from other operating systems (notably: Microsoft Windows). In System Architecture I give a quick overview of the main differences. Further down, I elaborate on using the command line interface of Linux (which is definitely a must-know if you are going to work with Gentoo Linux). Finally, I cover how to find help on the commands.

System Architecture

Linux is a multi-user platform. The administrator user is called the *root* user and can do anything on the system. Although it is perfectly possible to work on your system as this administrator, it is advised to create individual accounts for everyone who needs access to the system and only grant those users access to important resources or administrative commands (there are tools that allow a user to switch from one user to another or run certain commands as a different user).

For your personal PC, this probably means you're going to create a single additional account for yourself. If the PC is shared with many people (say household PC or a work station at your office) you most likely want to create accounts for every possible person. Sharing an account isn't good practice: if you need to share files and directories with other people, there are better ways than sharing the same account.

The personal account you use is limited, yet definitely not unusable. A few of the differences between a non-privileged account and the root account are:

- the personal account can only modify files it has created itself, so there is no possibility of corrupting your system (as the operating system files are immutable for the personal accounts). This also means that the personal account cannot install any additional software on the system (that's okay - I will tell you how to elevate your privileges later on so that you can manage your system when you need to).
- the personal account cannot fill up a partition to the point that the system is endangered: by default, 5% of each partition is reserved for privileged users only
- if malicious events happen using your personal account, traces of what has happened might show up in log files that the personal account cannot tamper with
- ...

Every task a user starts results in a process on the system. For instance, a user that fires up a browser results in at least one process representing the browser program. By default, this process runs with the privileges of this user and has therefore only access to the files and resources of this user. Another reason why not to use the root account: you don't want a malicious script that gets executed in your browser, and which triggers an exploitable bug (this isn't fantasy, it might happen) cause your browser to start wiping out important system files...

Files and directories themselves are, permission-wise, represented with three sets of permissions: one for the owner, one for the owning group and one for everyone. More on the file privileges later.

User Accounts

As the root user is an administrator account with no privilege limitations whatsoever, any action ran by the root user can potentially hurt your system:

- removing the wrong files or directories (there is no undelete option)
- removing or manipulating system files (possibly resulting in a malfunctioning system - and a reboot won't solve this)
- disabling networking or changing network behaviour
- placing files on wrong places (possibly losing files or interfering with system operations)
- killing or otherwise manipulating the wrong process(es) (possibly resulting in service malfunction, shutdown or misbehaviour)
- ...

For this reason, user accounts should be created. A default user account:

- can only edit or remove files that belong to this user
- can not interfere with system-wide settings (including networking, device access, ...)
- can only place files inside his own (dedicated) home directory
- can only kill / manipulate processes that he has launched himself

Important

It is still possible for non-root user accounts to do harm. For instance, a malfunctioning program executed by a user might still remove all user files from the file system. Although this is a major disaster for end users (please, do not forget to take backups regularly) the system itself can still run as the operating system files are not touched.

Next to user accounts, any Linux installation has *system accounts* as well. These are regular accounts just like the user accounts (with the same limitations), but cannot be used to log on (they do not have a valid password), often have no dedicated home directory and are used by the operating system to run specific services with limited rights (increasing security) rather than administrative rights.

To ease managing user accounts (and permissions / access to resources), users are made part of groups. Access to files can be managed through group ownership whereas every member of this group can access (or even write to) the file. A user can be part of many groups.

Processes

A *process* is a running task on the system. Each process runs as a particular user: this not only means that that process can only access resources that are available for that user, but also that only this user (and the root user) can manipulate the process (for instance, kill the process).

Each process has one parent (except the top process, which is always the **init** process) and can have one or more child processes. If a process dies, all its children die as well (so never try to kill the **init** process as it would result in a total annihilation of all processes on the system).

Running processes do not necessarily consume CPU cycles. An idle system can easily be running a few dozen (or even up to a hundred) processes. This is because processes have various states:

actively running, sleeping, waiting, stopped, ... Luckily, the Linux Operating System is very efficient in balancing the requirements of the applications with the available resources so you, as an end user, do not notice that this many processes are running.

Note

One process can also have multiple threads. Threads are part of an application (but managed by the operating system) and allow running code within an application in parallel. A program with several threads is still a single process.

Files and File Structure

On Linux (and all the Unix and Unix-like operating systems) the file system is hierarchically structured. At the top, you have the *file system root*, which is named "`/`". Below you'll have files and directories, following the standard described in a previous section, the section called “File system Hierarchy Standard”. Every position within the structure has its meaning. An example is `/home/swift`, which is the home directory of the "swift" user account. Another example is `/usr/share/doc/googleearth-4.2.198.2451/`, the location where a specific version of the Google Earth program stores its package documentation.

Next to regular files (like text files or program-related files) and directories, Linux also has the notion of:

- *links*, which are other names for the same file (allowing for one file to have multiple references),
- *symbolic links*, which are pointers to other files (differs from links because it is merely a pointer, so it can point to something that doesn't exist) and are somewhat (but not quite) comparable to Microsoft Windows' shortcuts.
- *named pipes* or *FIFOs*, which are temporary buffers where one process can write to and another process can read from
- *unix domain sockets*, which are created by one process and where several other processes can write to or receive information from
- *character* and *block device file*, which represents a hardware device on the system, allowing to exchange data with the device through regular file operations

All these special files are represented as files on the Linux file system. Take `/dev/sda1` as an example. It represents the first partition (`1`) on the first (`a`) SCSI device (`sd`) known by the system. `/dev/sda1` is a block device file because read/write operations occur in blocks (certain amount of data) whereas character device operations are on a character-by-character basis.

I will be discussing device files, and how Linux manages them, in a later chapter.

Permissions

Files (and directories and any of the special files talked about previously - I am going to refer to those as "files" as well throughout this book) are associated with three sets of permissions: one for the owner, one for the group owner and one for all the rest. Each set contains several flags:

- read access to the file or directory
- write access to the file or directory
- execute rights on the file (probably because it is an application binary or a script) or entering rights on the directory (meaning you can jump inside the directory)

Next to these three sets, additional permissions can be set for executable files and directories, but these are far less important to understand at this moment:

- *set user/group id* on the file (when the file is executed, the process is not ran as the user/group who executed it but as the owner) or directory (only applies to the group set; in this case, files created inside the directory automatically get the group ownership of the directory, allowing to easily share files between accounts)

- *restriction deletion* flag (restricts deletion of files inside the affected directory unless the user is the owner of the file or directory)

The restriction deletion flag needs some explanation. It is used on world-writable directories (a well-known example is `/tmp`, where temporary files can be stored) and used to prevent users to remove files they didn't create. A world-writable directory would otherwise allow other users to remove files from that directory (as they too have write access to the directory). The restriction deletion flag is also known as the *sticky* flag.

Using the Command Line

Most Linux distributions, once booted up, provide you with a graphical logon screen, or even log you on automatically. With Gentoo Linux, this isn't the case. Rather, you'll be greeted with a command-line prompt asking you to identify yourself:

```
This is seaheaven.homeworld (Linux x86_64 3.8.5) 22:30:00

seaheaven login:
```

At this prompt, you are asked to enter your user name which you have undoubtedly created during the Linux installation. Next, you are asked for your account password. If the user name exists and the password matches, then you are greeted with a *command line prompt*. The next listing shows a typical prompt for the account with user name "captain" on a system with host name "seaheaven":

```
captain@seaheaven ~ $
```

The prompt is structured as follows:

1. the user name of the account logged on to the system

2. the host name of the system where the user is logged on to

3. the current location where the user is on the file system (~ means the users' home directory)

4. a prompt sign telling the user if he is a regular user ($) or root user (#).

In the following sections, I'll give you a quick introduction to the world of the Linux command line.

Navigating

The prompt is actually something rendered by a shell. A *shell* is an interactive program which takes on commands from the user input (keyboard) and executes those on the system. One prime example of commands are those used for navigating through the file system:

- **pwd** shows the present working directory (where the user is currently in)

- **cd** allows a user to change the directory he is in

- **ls** shows the contents of the current directory (listing) or any other location if asked

An example session is shown below.

```
captain@seaheaven ~ $ pwd
```

```
/home/captain
captain@seaheaven ~ $ ls
Documents     Movies       Music       Pictures
opentasks.txt
captain@seaheaven ~ $ cd Documents
captain@seaheaven Documents $ pwd
/home/captain/Documents
```

To navigate from one directory to another, there are a couple of ways to use the **cd** command. Note that the **pwd** command is shown to show the current location. It is not part of the process of navigating between directories!

- You can use a relative path to go down a tree. In the example, the "**cd Documents**" goes from `/home/captain` to `/home/captain/Documents` through the relative path "`Documents`".

  ```
  $ pwd
  /home/captain
  $ cd Documents
  $ pwd
  /home/captain/Documents
  ```

- You can go up through the special name '`..`' (dot dot), so "**cd ..**" would go from `/home/captain/Documents` to `/home/captain`.

  ```
  $ pwd
  /home/captain/Documents
  $ cd ..
  $ pwd
  /home/captain
  ```

- You can also use absolute paths. For instance, to go immediately to `/etc/init.d` (a special directory we'll talk about later) you would type "**cd /etc/init.d**".

  ```
  $ pwd
  /home/captain/Documents
  $ cd /etc/init.d
  $ pwd
  /etc/init.d
  ```

- A special character is the ~ (tilde) which means the home directory, so "**cd ~**" would go to your home directory. Even shorter, just entering "**cd**" would go to your home directory. The ~ can be used in two ways: either to denote your home directory (**cd ~/Documents** goes to `/home/captain/Documents`) or someone else's home directory. In the latter case, you append the user name to the ~ sign, so: "**cd ~raghat/public_html/**" translates to `/home/raghat/public_html`.

  ```
  $ pwd
  /etc/init.d
  $ cd ~/Documents
  $ pwd
  /home/captain/Documents
  ```

If for some reason the change directory command cannot succeed (no permission to enter the directory, or the directory doesn't exist) you will be notified of the failure.

```
$ pwd
/home/captain
$ cd /etc/cron.daily
bash: cd: /etc/cron.daily: Permission denied
$ pwd
/home/captain
```

Listing Content

To list the contents of a directory, you would use the **ls** command. By default, it shows the files and directories of the current directory, but ls is one of those commands that takes a multitude of options. It is not my intention to describe each and every single option, but there are a few which are quite important to know right now.

First of all, the "**-l**" option shows not only the files and directories of a certain directory (or the current directory) but also information about each file / directory:

```
captain@seaheaven ~ $ ls -l
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Documents
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Movies
drwxr-xr-x 2 captain users 4096 2013-10-31 22:25 Music
drwxr-xr-x 2 captain users 4096 2013-10-31 22:24 Pictures
-rw-r--r-- 1 captain users   66 2013-10-31 22:30 opentasks.txt
```

The information contains, amongst other things, the permission sets (three times three characters: Read, Write and eXecute), the owning user and group, the size (directories are in multiples of 4 kilobytes in this case) and their creation date.

Another interesting option is "**-a**", which lets the ls command show all files (including the hidden files). A *hidden file* on a Unix / Linux system starts with a dot. Inside your home directory you'll find a variety of hidden files by default, the example below shows just a subset of such files.

```
captain@seaheaven ~ $ ls -a
Documents    Movies    Music    Pictures
.history     .maildir  .profile .vimrc
```

Hidden files are not meant to really hide the files, but rather not to show them using regular listings so they do not clutter the listing output too much. Most hidden files are configuration files (like the .vimrc file which is the configuration file for the vim editor for the current user).

As with most command line utilities, you can combine arguments; you are even able to put them together, so **ls -l -a** is the same as **ls -la**. The order also doesn't matter, so **ls -la** and **ls -al** are both the same.

Analyzing ls -l output

You will find that you often use "**ls -l**" to get the necessary information about a file or directory. As such I give you a quick overview of the meaning of the various fields in **ls**' output:

```
$ ls -l 12-servicemanagement.xml
-rw-r--r-- 1 swift users 26976 Apr 22 21:21 12-servicemanagement.xml
|<+><+><+> | <-+-> <-+-> <-+-> <-----+----> <---------+------------>
| |  |  |  |   |     |     |         |                `- file name or directory
| |  |  |  |   |     |     |         `- last modification time
| |  |  |  |   |     |     `- size (in bytes)
| |  |  |  |   |     `- group owning the file or directory
| |  |  |  |   `- user owning the file or directory
| |  |  |  `- number of hard links to the file or directory
| |  |  `- privileges (in this case: read-only) for everybody
| |  `- privileges (in this case: read-only) for owning group ("users")
| `- privileges (in this case: read-write) for owning user ("swift")
`- type of file (in this case: regular file)
```

Basic File Manipulation

The most basic manipulations are copying, moving, renaming and deleting files, which we explain below...

Copying Files & Directories

To copy a file you would use the **cp** command. Its basic syntax is simple: *cp source destination*. The destination can be a directory in which case the file is copied to that directory. If the destination is a file name, the file is copied to that file(name).

```
captain@seaheaven ~ $ ls Documents
captain@seaheaven ~ $ cp opentasks.txt Documents/tasks.txt
captain@seaheaven ~ $ ls Documents
tasks.txt
captain@seaheaven ~ $ cp opentasks.txt Documents
captain@seaheaven ~ $ ls Documents
opentasks.txt      tasks.txt
```

By default, the **cp** command creates the destination file with the privileges and ownership of the user that is executing the command. In the next example, I copy the file `/etc/inittab` to the current directory (.):

```
$ ls -l /etc/inittab
-rw-r--r-- 1 root root 1890 Feb 15 20:39 /etc/inittab
$ cp /etc/inittab .
$ ls -l inittab
-rw-r--r-- 1 swift users 1890 Apr 22 22:49 inittab
```

You can ask **cp** to preserve all information (like access privileges, ownership and time stamps) although you might need to be root to do so.

To recursively copy an entire directory and its content, use the -r option. For instance, the following example would copy the folder workdocuments and all files and directories inside it to the current working directory.

```
$ cp -r /media/usb/workdocuments .
```

Moving and Renaming Files & Directories

To move a file, the **mv** command should be used. Its syntax is *mv source destination*. If the destination is a file name rather than a directory, then the **mv** command also performs the task of renaming files. In the next example, the `tasks.txt` file is moved from the `Documents` folder to the `/media/usb/workdocuments` directory:

```
captain@seaheaven ~ $ ls Documents/
opentasks.txt     tasks.txt
captain@seaheaven ~ $ mv Documents/tasks.txt /media/usb/workdocuments
captain@seaheaven ~ $ ls Documents/
opentasks.txt
captain@seaheaven ~ $ ls /media/usb/workdocuments
tasks.txt
```

Removing Files & Directories

The final task is removing files. The **rm** command uses the following syntax: *rm filename*. To remove an entire directory, you could just give the directory name, but that only works if the directory is empty. Otherwise, you need to add the "-r" option (which stands for recursive: the directory and every file and directory inside it).

```
captain@seaheaven ~ $ rm Documents/opentasks.txt
captain@seaheaven ~ $ rm tasks.txt
```

A popular set of options to **rm** is "**-rf**": it tells rm to remove all files and directories recursively, and also tells it not to ask the user for validation (force). Make sure you know what you are doing, Linux does not have an undo feature!

```
$ rm -rf Documents
```

Luckily, **rm** only works with the privileges of the user that is executing it, so you cannot accidentally remove system files if you aren't running as the privileged root user:

```
$ rm -f /etc/passwd
rm: cannot remove '/etc/passwd': Permission denied
```

Editing Text Files

An important task for any Linux user is knowing how to edit text files. There are many text editors around, such as the popular vim and emacs. Gentoo recommends nano as your first editor because it is easy to work with and understand. My preference is for vim.

Using Nano

As such, a quick primer in the use of nano. If you launch nano it shows you the main editor screen for a new file. If you launch nano with a file name as its argument, it opens up the text file, ready for editing. As an example, I open up the `opentasks.txt` file using nano:

```
captain@seaheaven ~ $ nano opentasks.txt
```

Figure 4.1. nano editor opened with the opentasks.txt file as its working document

The first line shows the name of the file opened while the last two lines show you a summary of the keyboard commands to manipulate the file. The ^ character means that you have to press the Control character (Ctrl), so to save the file you would press Ctrl+O. To exit the editor, press Ctrl+X.

Some text files are very sensitive to newlines - configuration files are the most important ones. By default, nano automatically wraps lines when you hit the sidebar. To inform nano that you want to keep lines as they are, add the "**-w**" argument: "**nano -w opentasks.txt**". This is the recommended approach when editing configuration files.

Viewing Text Files

If you want to view the content of a text file but don't want to launch the text editor, you can use **cat** or **less**. This also allows you to view the content of files without accidentally modifying them.

- With **cat**, you "dump" the content of a file to the terminal. If the file is large, all text will scroll by very quickly. You can try to scroll up using Shift+PgUp but this buffer is limited in length.

- With **less**, you view the content of a file page by page. The less command is called a pager and supports scrolling up / down (with the arrow keys or using the PgUp/PgDown keys), searching for specific text, ...

The **less** pager is not only used for viewing content of text files, but also for manual pages (which are discussed further down this chapter).

Table 4.1. Short summary of frequently used actions for the less pager

scroll down one line	e, j, <return> or arrow key down
scroll up one line	y, k or arrow key up
scroll down one page	f, <space> or PgDown
scroll up one page	b or PgUp
/<searchtext>	search forward for <searchtext>, use "**n**" for every next hit
?<searchtext>	search backward for <searchtext>, use "**?**" for every previous hit
h	display the help screen

In case you were wondering, the name **less** is a hint to an older, still supported pager called **more**. With **more**, you can also view the content of a file page by page, but **more** only supports scrolling down one line (<return>) or page (<space>). So **less** is more... than **more**.

Linking Files and Directories

Linux supports two types of links: symbolic links and hard links. You might wonder why I'm already telling you here in the introductionary chapter on running Linux. Well, that's because symbolic links are often used in Linux, whereas you'll find output related to hard links all over the place. In fact, we already had some output related to hard links earlier: in the output of the "**ls -l**" command.

Symbolic links, also known as *symlinks*, are the easiest to understand. They're not real files, but rather references to files somewhere on the file system. Their reference is a location string, which allows the link to be used across media and even to files that don't exist (any more). As an example, you could have a symbolic link that points to `/home/swift/opentasks.txt`, even when this file doesn't exist (any more).

Hard links, on the contrary, are immediate links. They actually refer to the same location on the disk as their target file (through inode's) and the file system will not release this location as free space until there are no references to this location. In fact, every file you see in the file system is a link to a location. With hard links, you just create additional links.

Due to their technology, hard links can only point to file locations on the same medium and they cannot handle directories.

To create links, use **ln** (hard links) or **ln -s** (symlinks):

```
$ ln targetfilename newfilename
$ ln -s targetfilename newfilename
```

Thanks to links, you can manage your files more easily. For instance, you can have an important text file located inside `~/Documents/Work/Thesis/myThesis.txt` but link to it from your home directory (`~/myThesis.txt` points to `~/Documents/Work/Thesis/MyThesis.txt`) so that you don't have to traverse all these directories every time you want to open/edit the file.

File / Command Completion

A powerful feature of most Linux shells is file and command completion.

When you want to edit a file called, for instance, opentasks.txt, you can start with typing "**nano o**" followed by the <tab> key. If `opentasks.txt` is the only file or directory that starts with an o, the shell will automatically expand the command to "`nano opentasks.txt`". If there are more files or directories that start with o, typing <tab> twice will show you a list of candidates.

The same is true for commands. If nano is the only command starting with na (it isn't, but suppose it is) then typing "na" followed by a <tab> expands the command to nano. Pressing <tab> twice if it isn't the only command displays all matches:

```
$ na<tab><tab>
namei           nautilus
nano            nasl
nasm            native2ascii
$ na
```

Switching Terminals

One of Unix / Linux' advantages is the support for multiple terminals. When you are logged on to your system, you are actually watching one of the (virtual) terminals, most likely the first one. You can switch from one terminal to another using Alt+F# where F# is F1, F2, ... If you are on a graphical session, you'll need to use Ctrl+Alt+F# first. The graphical screen is positioned at the first terminal that can not be used to log on to (most likely Alt+F7).

The support for multiple terminals allows you to log on to your system several times and spread the tasks you want to execute in parallel across the various terminals. For instance, if you are configuring a program, you might want to have the configuration file itself open in one terminal and the program in another.

Hint: if you are working on the command line, you can also use the **chvt** command to switch between terminals: **chvt 2** switches to the second terminal (similar to Alt+F2).

Logging Out

To log out from an existing session, enter **exit** or press Ctrl+d:

```
captain@seaheaven ~ $ exit

This is seaheaven.homeworld (Linux x86_64 3.8.5) 22:30:00

seaheaven login:
```

Shutting Down

Finally, if you want to shut down your system, you will first need to become root. You can do this by switching to another terminal and log on as root, but you can also use the su command. This command will ask you to enter the root password after which you are known to the system as the root user:

```
captain@seaheaven ~ $ su -
Password: (Enter root password)
root@seaheaven ~ #
```

Now, you can issue the shutdown command to shut down (-h) or reboot (-r) the system immediately:

```
root@seaheaven ~ # shutdown -h now
```

Getting Help

Linux might seem a bit complex at first, especially because the tools novice users use are the same tools that experts use, making the tools accept expert options starters wouldn't understand. Handbooks

such as this one can only help users find their way, but once in a while any user, beginner or advanced, will need something to fall back to. Luckily, there are several resources you can address...

Man Pages

The *manual page* is available on the Linux system itself and can be accessed through the **man** command. By entering **man** followed by the command you want information about you receive the manual page of that command, explaining what the command does, what syntax you should use, what options it supports and what commands or files are related to this command.

```
~$ man emerge
```

The manual page is usually structured as follows:

- name of the command with a one-line description
- synopsis of the command, showing the syntax the command accepts
- options of the command with an explanation of each option
- further information on the command, like usage restrictions, default behaviour, ...
- copyright and authorship information
- related manual pages

Manual pages not only exist for commands, but for files (configuration files or device files), system calls and library functions (programming related information) and even standards and conventions. These topics are categorized in the following sections with the accompanying numbers:

1. user commands
2. system calls
3. library functions
4. special files (device files)
5. file formats and protocols
6. games and funny programs
7. overview, conventions and miscellaneous topics
8. administrative and privileged commands

For information about these sections, open up the introduction page for each section. For instance, the introduction page for the games section can be accessed through **man 6 intro**. Knowing what sections are available can be important, especially when there are topics in different sections which have the same name. An example is passwd: there is a manual page for the **passwd** command (**man 1 passwd**) and one for the `passwd` file (**man 5 passwd**). If one would enter **man passwd**, the first section that has a manual page for the topic is displayed (in this case, the first section).

Once you are inside a manual page, you can navigate through it using the up/down arrow keys and the PgUp/PgDn keys. You can also search for some terms (press / followed by the term) and more. A full explanation of all the commands of the manual viewer (which, on most systems, is the **less** tool) can be obtained by pressing 'h'.

To quit viewing the manual page, press 'q'.

If you know that a manual page exists somewhere, but you do not know its name, you can use the **apropos** command (which is the equivalent of **man -k**). Giving a keyword, the command will give

a list of manual pages that are related to the keyword given. To build the database, you might need to run **makewhatis** first. On most systems, this is already scheduled to happen in the background on a daily basis.

```
~$ apropos ebuild
```

Info Pages

Another reference tool on your system is the **info** page help system. The **info** tool allows you to browse through info pages using a hierarchical approach (next/previous section, go upwards, ...) as well as using hyperlinks (select a topic in the info page to go to the section on that topic).

Let's take a look at the GCC info pages:

```
~$ info gcc
...
File: gcc.info,  Node: Top,  Next: G++ and GCC,  Up: (DIR)

Introduction
************

This manual documents how to use the GNU compilers, as well as their
features and incompatibilities, and how to report bugs.  It corresponds
to GCC version 4.1.2.  The internals of the GNU compilers, including
how to port them to new targets and some information about how to write
...
```

At the top of the info page you find where you are in the sequence of documents (as this is the first page, there is no real previous section nor upwards section). The next topic (which you can go to by pressing **n**) is "G++ and GCC".

Inside the text you will find text snippets following by two colons:

```
* G++ and GCC::     You can compile C or C++ programs.
* Standards::       Language standards supported by GCC.
* Invoking GCC::    Command options supported by `gcc'.
* C Implementation:: How GCC implements the ISO C specification.
* C Extensions::    GNU extensions to the C language family.
* C++ Extensions::  GNU extensions to the C++ language.
* Objective-C::     GNU Objective-C runtime features.
```

These texts are links which allow you to jump to that topic. Go with the cursor to "Invoking GCC" and press return:

```
File: gcc.info,  Node: Invoking GCC,  Next: C Implementation,  Prev: Standards,

3 GCC Command Options
*********************

When you invoke GCC, it normally does preprocessing, compilation,
assembly and linking.  The "overall options" allow you to stop this
process at an intermediate stage.  For example, the `-c' option says
not to run the linker.  Then the output consists of object files output
by the assembler.
```

You are now in the node called "Invoking GCC". The next (**n**) section is "C Implementation", the previous (**p**) is "Standards" and if you go up (**u**), you get to the top of the info document again.

Info pages are well used by various GNU projects (such as GCC). However, manual pages are more popular - perhaps because most developers find them easier to write.

Immediate Syntax Help

If you are not looking for an explanation on the command but want to know its syntax, most commands provide immediate help through the **-h** or **--help** arguments:

```
$ man -h
man, version 1.6e

usage: man [-adfhktwW] [section] [-M path] [-P pager] [-S list]
           [-m system] [-p string] name ...

  a : find all matching entries
  c : do not use cat file
  d : print gobs of debugging information
  D : as for -d, but also display the pages
  f : same as whatis(1)
  h : print this help message
  k : same as apropos(1)
  K : search for a string in all pages
  t : use troff to format pages for printing
  w : print location of man page(s) that would be displayed
      (if no name given: print directories that would be searched)
  W : as for -w, but display filenames only

  C file   : use `file' as configuration file
  M path   : set search path for manual pages to `path'
  P pager  : use program `pager' to display pages
  S list   : colon separated section list
  m system : search for alternate system's man pages
  p string : string tells which preprocessors to run
               e - [n]eqn(1)   p - pic(1)    t - tbl(1)
               g - grap(1)     r - refer(1)  v - vgrind(1)
```

Package-provided Documentation

Some packages provide more documentation in the form of guides (in HTML, PDF or other formats), README files and more. You will find most of this documentation at `/usr/share/doc`. Gentoo Linux compresses the documents using **bzip2** or **gzip** (both well known compression tools in the Unix world). To view the documents, you will first need to decompress those into a location you have write privileges to.

For instance, the zip package contains a document explaining what algorithm the tool uses to (de)compress data:

```
$ cd /usr/share/doc/zip-*
$ ls
algorith.txt.bz2   BUGS.bz2            CHANGES.bz2      MANUAL.bz2
README.bz2         timezone.txt.bz2    txtvsbin.txt.bz2 WHATSNEW.bz2
WHERE.bz2           ziplimit.txt.bz2
$ bunzip2 -c algoritm.txt.bz2 > ~/algorithm.txt
```

The resulting file, `algorithm.txt`, can be found in the home directory (`~` is an abbreviation for the home directory of the current user). In this case, the file is a regular text file which can be viewed through the **less** command, which is the same command used for manual pages.

In the above example, a redirection is used: the output of one command (**bunzip2 -c algorithm.txt.bz2**) is not shown on the screen, but rather redirected into a file (`algorithm.txt`). Redirection is discussed later on.

It is also possible to immediately read the (compressed) file using the **bzless** command:

```
$ bzless algorithm.txt.bz2
```

Online Documentation

Most projects have extensive documentation available. You are certainly advised to take a look at the projects' web sites or search through the internet for available guides and books. Sometimes, the best documentation is not written by the project itself but by satisfied users.

Exercises

1. Try to organize your home directory in logical sections. For instance, create directories where you put your personal documents, pictures, work-related documents and temporary documents.

2. Unix/Linux systems generally offer four compression methods. Whereas Zip is well known on the Microsoft Windows operating system, Unix/Linux uses GZip or BZip2 together with tar. How is "tar" used with gzip/bzip2? What is the fourth compression method?

3. As stated, Unix/Linux systems have no undo functionality: when you remove a file, it is removed. However, due to the technicalities of removing files (in essence only the reference to the file is removed, but its content remains on disk until it is overwritten) there are tools available that help you recover files. Try finding a few examples of such file recovery utilities for Linux.

Further Resources

- Introduction to Linux, A Hands-On Guide [http://www.tldp.org/LDP/intro-linux/html/index.html] by Machtelt Garrels (also available as PDF [http://www.tldp.org/LDP/intro-linux/intro-linux.pdf])

Chapter 5. The Linux File System

Introduction

The Linux file system is a hierarchically structured tree where every location has its distinct meaning. The file system structure is standardized through the file system hierarchy standard of which you'll find this chapter to be a description off. Lately however, more and more distributions are making a small change towards their file system layout (but all consistent) so the standard is in need of updating. When a setting deviates from the current standard, I will say so in this chapter.

Of course, a file system is always stored on media (be it a hard drive, a CD or a memory fragment); how these media relate to the file system and how Linux keeps track of those is also covered in this chapter.

Structure

The file system is a tree-shaped structure. The root of the tree, which coincidentally is called the *file system root* but is always depicted as being above all other, is identified by the slash character: "`/`". It is the highest place you can go to. Beneath it are almost always only directories:

```
~$ cd /
~$ ls -F
bin/     home/     opt/      srv/     var/
boot/    lib/      proc/     sys/
dev/     media/    root/     tmp/
etc/     mnt/      sbin/     usr/
```

The **ls -F** commands shows the content of the root location but appends an additional character to special files. For instance, it appends a "/" to directories, an "@" to symbolic links and a "*" to executable files. The advantage is that, for this book, you can easily see what type of files you have. By default, Gentoo enables colour-mode for the **ls** command, telling you what kind of files there are by the colour. For books however, using the appended character is more sane.

A popular way of representing the file system is through a tree. An example would be for the top level:

```
/
+- bin/
+- boot/
+- dev/
+- etc/
+- home/
+- lib/
+- media/
+- mnt/
+- opt/
+- proc/
+- root/
+- sbin/
+- srv/
+- sys/
+- tmp/
+- usr/
`- var/
```

The more you descend, the larger the tree becomes and it will soon be too difficult to put it on a single view. Still, the tree format is a good way of representing the file system because it shows you exactly how the file system looks like.

```
/
+- bin/
+- ...
+- home/
|  +- thomas/
|  |  +- Documents/
|  |  +- Movies/
|  |  +- Music/
|  |  +- Pictures/         <-- You are here
|  |  |  `- Backgrounds/
|  |  `- opentasks.txt
|  +- jane/
|  `- jack/
+- lib/
+- ...
`- var/
```

We've briefly covered navigating through the tree previously: suppose that you are currently located inside `/home/thomas/Pictures`. To descend even more (into the `Backgrounds` directory) you would type "**cd Backgrounds**". To ascend back (to `/home/thomas`) you would type "**cd ..**" (`..` being short for "parent directory").

Before we explain the various locations, let's first consider how the file system is stored on one (or more) media...

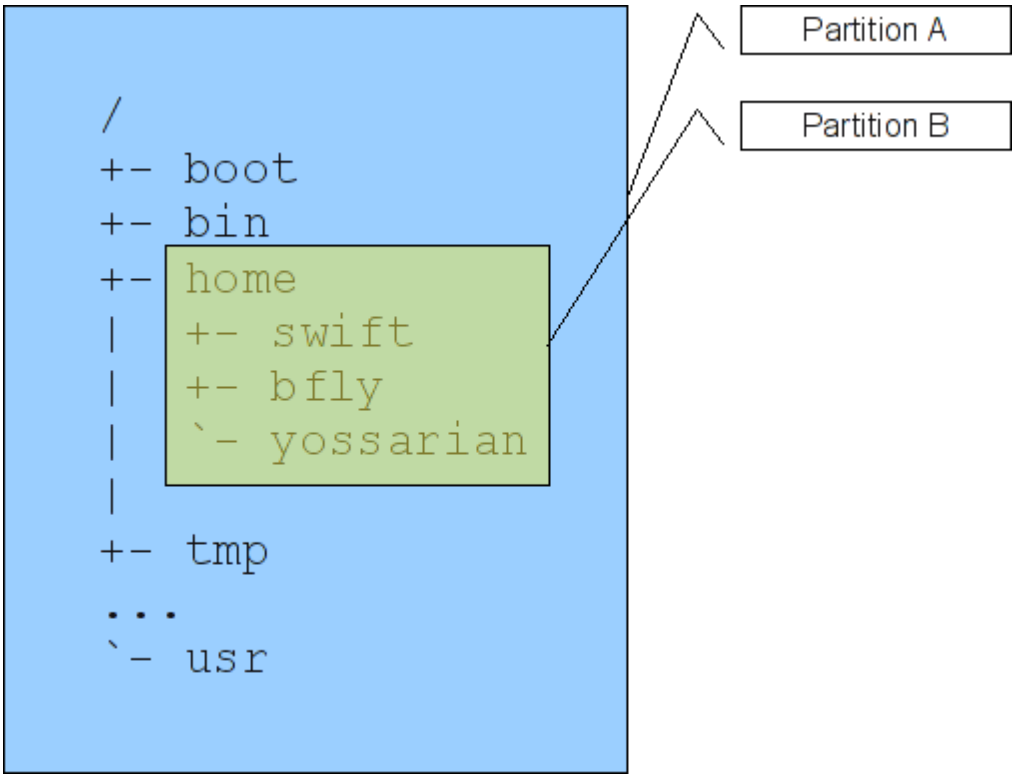
Mounting File Systems

The root of a file system is stored somewhere. Most of the time, it is stored on a partition of a disk. In many cases you would want to combine multiple partitions for a single file system. Combining one partition with the file system is called *mounting a file system*. Your file system is always seen as a tree structure, but parts of a tree (a *branch*) can be located on a different partition, disk or even other medium (network storage, DVD, USB stick, ...).

Mounting

Suppose that you have the root of a file system stored on one partition, but that all the users' files are stored on another. This would mean that `/`, and everything beneath it, is on one partition except `/home` and everything beneath that, which is on a second one.

Figure 5.1. Two partitions used for the file system structure



The act of mounting requires that you identify a location of the file system as being a mount point (in the example, `/home` is the mount point) under which every file is actually stored on a different location (in the example, everything below `/home` is on the second partition). The partition you "mount" to the file system doesn't need to know where it is mounted on. In fact, it doesn't. You can mount the users' home directories at `/home` (which is preferable) but you could very well mount it at `/srv/export/systems/remote/disk/users`. Of course, the reason why you would want to do that is beyond me, but you could if you want to.

The mount command by itself, without any arguments, shows you a list of mounted file systems:

```
$ mount
/dev/sda8 on /             type ext3       (rw,noatime)
proc      on /proc         type proc       (rw)
sysfs     on /sys          type sysfs      (rw,nosuid,nodev,noexec,relatime)
udev      on /dev          type devtmpfs   (rw,nosuid,relatime,size=10240k,mode=
devpts    on /dev/pts      type devpts     (rw,nosuid,noexec,relatime,gid=5,mod
/dev/sda7 on /home         type ext3       (rw,noatime)
none      on /dev/shm      type tmpfs      (rw)
/dev/sda1 on /mnt/data     type ext3       (rw,noatime)
usbfs     on /proc/bus/usb type usbfs      (rw,noexec,nosuid,devmode=0664,devgi
```

The above example, although bloated with a lot of other file systems we know nothing about yet, tells us that the file system can be seen as follows:

```
/             (on /dev/sda8)
+- ...
+- dev/       (special: "udev")
|  +- pts     (special: "devpts")
|  `- shm     (special: "none")
+- proc/      (special: "proc")
|  `- bus/
|     `- usb/ (special: "usbfs")
```

```
+- sys/       (special: "sys")
+- home/      (on /dev/sda7)
`- mnt/
   `- data/   (on /dev/sda1)
```

Ignoring the special mounts, you can see that the root of the file system is on device `/dev/sda8`. From `/home` onwards, the file system is stored on `/dev/sda7` and from `/mnt/data` onwards, the file system is stored on `/dev/sda1`. More on this specific device syntax later.

The concept of mounting allows programs to be agnostic about where your data is structured. From an application (or user) point of view, the file system is one tree. Under the hood, the file system structure can be on a single partition, but also on a dozen partitions, network storage, removable media and more.

File Systems

Each medium which can contain files is internally structured. How this structure looks like is part of the file system it uses. Windows users might remember that originally, Microsoft Windows used FAT16 and later on FAT32 before they all migrated to one of the many NTFS revisions currently in use by Microsoft Windows. Well, all these are in fact file systems, and Linux has its own set as well.

Linux however doesn't require its partitions to have one possible file system (like "only NTFS is supported"): as long as it understands it and the file system supports things like ownership and permissions, you are free to choose whatever file system you want. In fact, during most distribution installations, you are asked which file system to choose. The following is a small list of popular file systems around, each with a brief explanation on its advantages and disadvantages...

- The *ext2* file system is Linux' old, yet still used file system. It stands for extended 2 file system and is quite simple. It has been in use almost since the birth of Linux and is quite resilient against file system fragmentation - although this is true for almost all Linux file systems. It is however slowly being replaced by journalled file systems.

- The *ext3* file system is an improvement on the ext2 file system, adding, amongst other things, the concept of journalling.

- The *ext4* file system is an improvement on the ext3 file system, adding, amongst other things, support for very large file systems/files, extents (contiguous physical blocks), pre-allocation and delayed allocation and more. The ext4 file system is backwards compatible with ext3 as long as you do not use extents. Ext4 is frequently seen as the default file system of choice amongst administrators and distributions.

- The *reiserfs* file system is written from scratch. It provides journalling as well, but its main focus is on speed. The file system provides quick access to locations with hundreds of files inside (ext2 and ext3 are much slower in these situations) and keeps the disk footprint for small files small (some other file systems reserve an entire block for every file, reiserfs is able to share blocks with several files). Although quite popular a few years back, the file system has been seeing a lack of support through its popular years (harmful bugs stayed in for quite some time) and is not frequently advised by distributions any more. Its successor, *reiser4*, is still quite premature and is, due to the imprisonment of the main developer Hans Reiser, not being developed that actively any more.

- The *btrfs* file system is a promising file system. It addresses concerns regarding huge storage backend volumes, multi-device spanning, snapshotting and more. Although its primary target was enterprise usage, it also offers interesting features to home users such as online grow/shrink (both on file system as well as underlying storage level), object-level redundancy, transparent compression and cloning.

- The *xfs* file system is an enterprise-ready, high performance journaling file system. It offers very high parallel throughput and is therefore a common choice amongst enterprises.

- The *zfs* file system (ZFSonLinux) is a multi-featured file system offering block-level checksumming, compression, snapshotting, copy-on-write, deduplication, extremely large

volumes, remote replication and more. It has been recently ported from (Open)Solaris to Linux and is gaining ground.

A *file system journal* keeps track of file write operations by first performing the write (like adding new files or changing the content of files) in a journal first. Then, it performs the write on the file system itself after which it removes the entry from the journal. This set of operations ensures that, if at any point the file system operation is interrupted (for instance through a power failure), the file system is able to recover when it is back up and running by either replaying the journal or removing the incomplete entry: as such, the file system is always at a consistent state.

It is usually not possible to switch between file systems (except ext2 <> ext3) but as most file systems are mature enough you do not need to panic "to chose the right file system".

Now, if we take a look at the following mount output, we notice that there is a part of the line that says which "type" a mount has. Well, this type is the file system used for that particular mount.

```
$ mount
rootfs      on /                          type rootfs      (rw)
sysfs       on /sys                       type sysfs       (rw,seclabel,relatime)
selinuxfs   on /sys/fs/selinux            type selinuxfs   (rw,relatime)
/dev/md3    on /                          type ext4        (rw,seclabel,noatime,n
/dev/md4    on /srv/virt                  type ext4        (rw,noatime,data=order
proc        on /proc                      type proc        (rw,nosuid,nodev,noexe
tmpfs       on /run                       type tmpfs       (rw,rootcontext=system_
udev        on /dev                       type devtmpfs    (rw,seclabel,nosuid,re
/dev/mapper/volgrp-nfs    on /srv/nfs     type ext4        (rw,noatime,data=journa
/dev/mapper/volgrp-usr    on /usr         type ext4        (rw,noatime,data=journa
/dev/mapper/volgrp-home   on /home        type ext4        (rw,noatime,nosuid,nod
/dev/mapper/volgrp-opt    on /opt         type ext4        (rw,noatime,data=journa
/dev/mapper/volgrp-var    on /var         type ext4        (rw,noatime,data=journa
/dev/mapper/volgrp-vartmp on /var/portage type ext4      (rw,noatime,data=order
mqueue      on /dev/mqueue                type mqueue      (rw,seclabel,nosuid,no
devpts      on /dev/pts                   type devpts      (rw,seclabel,nosuid,no
shm         on /dev/shm                   type tmpfs       (rw,seclabel,nosuid,no
securityfs  on /sys/kernel/security       type securityfs  (rw,nosuid,nodev,noexe
debugfs     on /sys/kernel/debug          type debugfs     (rw,nosuid,nodev,noexe
none        on /selinux                   type selinuxfs   (rw)
tmpfs       on /var/tmp                   type tmpfs       (rw,nosuid,noexec,node
tmpfs       on /tmp                       type tmpfs       (rw,nosuid,noexec,node
rc-svcdir   on /lib64/rc/init.d           type tmpfs       (rw,nosuid,nodev,noexe
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,nodev,noexec,nosui
rpc_pipefs  on /var/lib/nfs/rpc_pipefs  type rpc_pipefs  (rw)
nfsd        on /proc/fs/nfsd              type nfsd        (rw,nodev,noexec,nosui
```

As you can see, all partitions (the non-special lines) are all typed as ext4. But what are those other file systems?

- *proc* is a special file system which doesn't exist on a device, but is a sort of gateway to the Linux kernel. Everything you see below `/proc` is something the kernel displays the moment you read it. It is a way to communicate with the kernel (and vice versa) using a very simple interface: file reading and file writing, something well supported.

  I will elaborate on `/proc` more later in this chapter.

  proc is known to be a *pseudo file system*: it does not contain real files, but runtime information.

- *sysfs* is a special file system just like proc: it doesn't exist on a device, and is a sort of gateway to the Linux kernel. It differs from proc in the way it is programmed as well as structured: sysfs is more structured and tailored towards computer-based parsing of the files and directories, whereas proc is more structured and tailored towards human-based reading/writing to the files and directories.

The idea is that proc will eventually disappear (although there is no milestone set yet since many people like the simple way `/proc` gives them information) and be fully replaced by the sysfs file system.

Like `/proc`, sysfs is known to be a pseudo file system and will be elaborated more later in this chapter.

- *tmpfs* is a temporary file system. Its contents is stored in memory and not on a persistent disk. As such, its storage is usually very quick (memory is a lot faster than even the fastest SSDs and hard disks out there). I do say usually, because tmpfs can swap out pages of its memory to the swap location, effectively making those parts of the tmpfs file system slower (as they need to be read from disk again before they can be used).

  Within Linux, tmpfs is used for things like the shared memory in /dev/shm and /tmp.

- *devtmpfs* is similar to the tmpfs file system, but contains device files managed by the kernel. The devtmpfs file system was brought to life to handle the concern of providing important device files before udev (the device manager) is able to start up and take control.

- *devpts* is a pseudo file system like proc and sysfs. It contains device files used for terminal emulation (like getting a console through the graphical environment using **xterm**, **uxterm**, **eterm** or another terminal emulation program). In earlier days, those device files were created statically, which caused most distributions to allocate a lot of terminal emulation device files (as it is difficult to know how many of those emulations a user would start at any point in time). To manage those device files better, a pseudo file system is developed that creates and destroys the device files as they are needed.

- *usbfs* is also a pseudo file system and can be compared with devpts. It also contains files which are created or destroyed as USB devices are added or removed from the system. However, unlike devpts, it doesn't create device files, but pseudo files that can be used to interact with the USB device.

  As most USB devices are generic USB devices (belonging to certain classes, like generic USB storage devices) Linux has developed a framework that allows programs to work with USB devices based on their characteristics, through the usbfs file system.

- *selinuxfs* is a pseudo file system that represents the SELinux subsystem in the Linux kernel. It is used by the SELinux libraries to interact with the SELinux security server, querying the SELinux policy and more. Linux systems that do not have SELinux enabled will not have this file system mounted.

- *mqueue* is a pseudo file system used for inter-process message queue support (POSIX message queues)

- *binfmt_misc* is a pseudo file system used to register executable formats. Through binfmt, the Linux kernel is able to execute arbitrary executable file formats by recognizing the registered executable formats and passing it on to userspace applications.

Many more special file systems exist (some are even mentioned in the **mount** output above), but I leave that to the interested reader to find out more about these file systems.

Partitions and Disks

Every hardware device (except the network interface) available to the Linux system is represented by a *device file* inside the `/dev` location. Partitions and disks are no exception. Let's take a serial ATA hard disk as an example.

A SATA disk driver internally uses the SCSI layer to represent and access data. As such, a SATA device is represented as a SCSI device. The first SATA disk on your system is represented as `/dev/sda`, its first partition as `/dev/sda1`. You could read `sda1` backwards as: "1st partition (1) on the first (a) scsi device (sd)".

```
~$ ls -l /dev/sda1
```

```
brw-rw---- 1 root disk 8,  1  Nov 12  10:10  /dev/sda1
```

A regular ATA disk (or DVD-ROM) would be represented by `/dev/hda` (hd stood for hard disk but is now seen as the identification of an ATA device).

```
$ ls -l /dev/hda
brw-rw---- 1 root cdrom 3, 0  Apr 23  21:00 /dev/hda
```

On a default Gentoo installation, the device manager (which is called **udev**) creates the device files as it encounters the hardware. For instance, on my system, the partitions for my first SATA device can be listed as follows:

```
$ ls -l /dev/sda*
brw-r----- 1 root disk 8, 0 Sep 30 18:11 /dev/sda
brw-r----- 1 root disk 8, 1 Sep 30 18:11 /dev/sda1
brw-r----- 1 root disk 8, 2 Sep 30 18:11 /dev/sda2
brw-r----- 1 root disk 8, 5 Sep 30 18:11 /dev/sda5
brw-r----- 1 root disk 8, 6 Sep 30 18:11 /dev/sda6
brw-r----- 1 root disk 8, 7 Sep 30 18:11 /dev/sda7
brw-r----- 1 root disk 8, 8 Sep 30 18:11 /dev/sda8
```

Inside the /dev location, there are also symbolic links (pointers) towards those device files, which can be used to identify the partitions or disks by other means. For instance, to list the disk devices by their *UUID (Universally Unique IDentifier)*:

```
$ ls -l /dev/disk/by-uuid
total 0
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 1628b93d-3448-4b8c-b72b-1d68e89bd2fa ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 5550c45f-9660-44f2-8e86-05a612d028a3 ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 77eb40be-f571-49c6-bbb0-a12677615fe3 ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9644f675-6eaf-4974-9e1a-0b8eafa931ae ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9beda062-6e15-4323-9ad1-53b6a9e39676 ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 9e7a1178-b0ad-4cd8-8977-a471a5d2b797 ->
lrwxrwxrwx. 1 root root  9 Dec 16 20:01 b06fa545-0d5a-4c9a-97cb-83b4e1799f9a ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 b80c76a3-d52f-4006-9bf5-62f4d7edc791 ->
lrwxrwxrwx. 1 root root 10 Dec 16 20:01 bdb15de1-3430-47b4-9e63-ee58557f1d17 ->
lrwxrwxrwx. 1 root root  9 Dec 16 20:01 c44503ce-e52e-452c-b4bc-767ddd1d3b27 ->
lrwxrwxrwx. 1 root root  9 Dec 16 20:01 d8a2bb27-15da-49bb-b205-3160c307835c ->
```

The advantage of using UUIDs is that they uniquely identify a partition or disk. If the disks in the system are later juggled around, or we are talking about removable devices, then by using the UUID we know for sure that we are looking at the right partition (and not another disk that got named `/dev/sda2` for instance).

The 'mount' Command and the fstab file

The act of mounting a medium to the file system is performed by the **mount** command. To be able to perform its duty well, it requires some information, such as the *mount point*, the file system type, the device and optionally some mounting options.

For instance, the mount command to mount `/dev/sda7`, housing an ext3 file system, to `/home`, would be:

```
# mount -t ext3 /dev/sda7 /home
```

One can also see the act of mounting a file system as "attaching" a certain storage somewhere on the file system, effectively expanding the file system with more files, directories and information.

However, if your system has several different partitions, it would be a joke to have to enter the commands every time over and over again. This is one of the reasons why Linux has a file system

definition file called `/etc/fstab`. The fstab file contains all the information mount could need in order to successfully mount a device. An example fstab is shown below:

```
/dev/sda8  /          ext4    defaults,noatime      0 0
/dev/sda5  none       swap    sw                    0 0
/dev/sda6  /boot      ext4    noauto,noatime        0 0
/dev/sda7  /home      ext4    defaults,noatime      0 0
/dev/sdb1  /media/usb auto    user,noauto,gid=users 0 0
```

The file is structured as follows:

1. The device to mount (also supports labels - we'll discuss that later)
2. The location to mount the device to (mount point)
3. The file system type, or auto if you want Linux to automatically detect the file system
4. Additional options (use "defaults" if you don't want any specific option), such as noatime (don't register access times to the file system to improve performance) and users (allow regular users to mount/umount the device)
5. Dump-number (you can leave this at 0)
6. File check order (you can leave this at 0 as well)

Thanks to this file, the previous **mount** command example is not necessary any more (as the mount is performed automatically) but in case the mount has not been done already, the command is simplified to:

```
# mount /home
```

If you ever need to remove a medium from the file system, use the *umount* command:

```
# umount /home
```

This is of particular interest for removable media: if you want to access a CD or DVD (or even USB stick), you need to mount the media on the file system first before you can access it. Likewise, before you can remove the media from your system, you first need to unmount it:

```
# mount /media/dvd
(The DVD is now mounted and accessible)
# umount /media/dvd
(The DVD is now not available on the file system any more and can be
 removed from the tray)
```

Of course, modern Linux operating systems have tools in place which automatically mount removable media on the file system and unmount it when they are removed. Gentoo Linux does not offer such tool by default (you need to install it) though.

Swap location

You can (and probably will) have a partition dedicated for paging: this partition will be used by Linux when there is insufficient physical memory to keep all information about running processes (and their resources). When this is the case, the operating system will start putting information (which it hopes will not be used soon) on the disk, freeing up physical memory.

This swap partition is a partition like any other, but instead of a file system usable by end users, it holds a specific file system for memory purposes and is identified as a swap partition in the partition table:

```
# fdisk -l /dev/sda
Disk /dev/sda: 60.0 GB, 60011642880 bytes
255 heads, 63 sectors/track, 7296 cylinders
```

```
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x8504eb57

   Device Boot  Start    End      Blocks   Id  System
/dev/sda1   *       1   1275    10241406   83  Linux
/dev/sda2        1276   7296    48363682+   5  Extended
/dev/sda5        1276   1525     2008093+  82  Linux swap / Solaris
/dev/sda6        1526   1532       56196   83  Linux
/dev/sda7        1533   2778    10008463+  83  Linux
/dev/sda8        2779   7296    36290803+  83  Linux
```

The swap partition is pointed by through the `/etc/fstab` file and enabled at boot-up.

To view the currently active swap partitions (or files, as swap files are supported as well), view the content of the `/proc/swaps` file or run the **swapon -s** command:

```
# cat /proc/swaps
Filename     Type      Size    Used  Priority
/dev/sda5    partition 2008084 0     -1
```

The Linux File System Locations

As said before, every location on the Linux file system has its specific meaning. We've already covered a few of them without explicitly telling that those are standard locations, such as `/home` which houses the local users' home directories. The Linux File system Standard covers all these standard locations, but this chapter would be very incomplete if it didn't talk about these as well.

System Required Locations

The system required locations are locations you cannot place on another file system medium because those locations are required by the mount command itself to function properly:

- `/bin` usually contains executable programs needed to bring the system up and running. Recently however, more and more distributions are moving all applications towards `/usr/bin` and are using symbolic links to transition towards this new structure.
- `/etc` contains all the configuration files for the system (not the user-specific configurations)
- `/lib` usually contains the system libraries necessary to successfully boot the system and run the commands which are located inside `/bin`. Recently however, these files are also being migrated towards `/usr/lib`.
- `/sbin`, just like `/bin`, contains executable programs. However, whereas `/bin` has programs which users can use as well, `/sbin` contains programs solely for system administrative purposes

Userland Locations

Userland locations are the locations which contain the files for the regular operation of a system (such as application data and the applications themselves). These can be stored on separate media if you want, but if you do, you will need to setup an initial ram disk to boot your system with. More about initial ram file systems later. The location for the userland locations is `/usr` (which comes from Unix System Resources).

- `/usr` is the root of the userland locations (and usually the mount point of the separate medium)
- `/usr/X11R6` contains all the files necessary for the graphical window server (X11); they are subdivided in binaries (`bin/`), libraries (`lib/`) and header definitions (`/include`) for programs relying on the X11 system.
- `/usr/bin` contains all the executable programs

- `/usr/lib` contains all the libraries for the above mentioned programs
- `/usr/share` contains all the application data for the various applications (such as graphical elements, documentation, ...)
- `/usr/local` is often a separate mount as well, containing programs specific to the local system (the `/usr` might be shared across different systems in large environments)
- `/usr/sbin` is, like `/usr/bin`, a location for executable programs, but just like `/bin` and `/sbin`, `/usr/sbin` contains programs for system administrative purposes only.

General Locations

General locations are, well, everything else which might be placed on a separate medium...

- `/home` contains the home directories of all the local users
- `/boot` contains the static boot-related files, not actually necessary once the system is booted (for instance, it includes the bootloader configuration and kernel image)
- `/media` contains the mount points for the various detachable storage (like USB disks, DVDs, ...)
- `/mnt` is a location for temporarily mounted media (read: not worth the trouble of defining them in fstab)
- `/opt` contains add-on packages and is usually used to install applications into which are not provided by your package manager natively (as those should reside in /usr) or build specific to the local system (`/usr/local`).
- `/tmp` contains temporary files for the system tools. The location can be cleansed at boot up.
- `/var` contains data that changes in size, such as log files, caches, etc.

Special Kernel-provided File Systems

Some locations on the file system are not actually stored on a disk or partition, but are created and managed on-the-fly by the Linux kernel.

- `/proc` contains information about the running system, kernel and processes
- `/sys` contains information about the available hardware and kernel tasks
- `/dev` contains device files

These locations will often also have other (pseudo) file systems mounted underneath.

The Root File System /

As said before, the root file system / is the parent of the entire file system. It is the first file system that is mounted when the kernel boots (unless you use an initial ramdisk), and your system will not function properly if the kernel detects corruption on this file system. Also, due to the nature of the boot process, this file system will eventually become writable (as the boot process needs to store its state information, etc.)

Some locations on the root file system are strongly advised to remain on the root file system (i.e. you should never ever mount another file system on top of that location). These locations are:

- /bin and /sbin as these contain the binaries (commands) or links to binaries that are needed to get a system up to the point it *can* mount other file systems. Although this functionality is gradually becoming less and less so, it would still break systems if you make separate mounts for these (small) locations.

- /lib as this contains the libraries that are needed by the commands in /bin.
- /etc as this contains the systems' configuration files, including those that are needed during the boot-up of the system.

  A prime example of a configuration file inside `/etc` is `fstab` (which contains information about the other file systems to mount at boot time).

The Variable Data Location /var

The var location contains variable data. You should expect this location to be used frequently during the life time of your installation. It contains log files, cache data, temporary files, etc.

For many, this alone is a reason to give /var its own separate file system: by using a dedicated file system, you ensure that flooding the /var location doesn't harm the root file system (as it is on a different file system).

The Userland Location /usr

The usr location contains the systems' day-to-day application files. A specific property of the location is that, if you are not updating your system, it could be left unmodified. In other words, you should be able to have only read-only access to the /usr location. Most distributions however do not support this feature anymore and assume that the /usr location is writable by the administrator at all times.

Having /usr on a separate file system also has other advantages (although some might be quite far-fetched ;-)

- If you are performing system administration tasks, you could unmount /usr so that end users don't run any programs they shouldn't during the administrative window.
- By placing /usr (and some other locations) on separate media, you keep your root file system small which lowers the chance of having a root file system corruption that will make booting impossible.
- You can use a file system that is optimized for fast reading (writing doesn't require specific response times)

The advantages however are becoming less and less relevant nowadays. Instead, distributions are focusing more towards initial ram file systems (a small, in-memory file system used to boot the system), which will be discussed later in this book.

The Home Location /home

Finally, the /home location. This location contains the end users' home directories. Inside these directories, these users have full write access. Outside these directories, users usually have read-only rights (or even no rights at all). The structure inside a home directory is also not bound to specific rules. In effect, the users' home directory is the users' sole responsibility.

However, that also means that users have the means of filling up their home location as they see fit, possibly flooding the root file system if /home isn't on a separate partition. For this reason, using a separate file system for /home is a good thing.

Another advantage of using a separate file system for /home is when you would decide to switch distributions: you can reuse your /home file system for other Linux distributions (or after a re-installation of your Linux distribution).

Permissions and Attributes

By default, Linux supports what is called a *discretionary access control* (*DAC*) permission system where privileges are based on the file ownership and user identity. However, projects exist that

enable *mandatory access control* (*MAC*) on Linux, which bases privileges on roles and where the administrator can force security policies on files and processes.

As most MAC-based security projects (such as RSBAC [http://www.rsbac.org], LIDS [http://www.lids.org] and grSecurity [http://www.grsecurity.net]) are not part of the default Linux kernel yet, I will talk about the standard, discretionary access control mechanism used by almost all Linux distributions. SELinux [http://www.nsa.gov/selinux], which is part of the default Linux kernel, will also not be discussed. If you are interested in running a SELinux powered system, I recommend to use Gentoo Hardened [http://www.gentoo.org/proj/en/hardened] which supports SELinux. There is also a Gentoo Hardened SELinux Handbook [http://www.gentoo.org/proj/en/hardened/selinux/selinux-handbook.xml] which is worth reading through.

Read, Write and Execute

The Linux file system supports various permission flags for each file or directory. You should see a flag as a feature or privilege that is either enabled or disabled and is set independently of the other flags. The most used flags on a file system are the read (r), write (w) and execute (x) flags. Their meaning differs a bit based on the target.

However, supporting these flags wouldn't make a system secure: you want to mix these privileges based on who works with the file. For instance, the system configuration files should only be writable by the administrator(s); some might not even be readable by the users (like the file containing the user passwords).

To enable this, Linux supports three kinds of privilege destinations:

- the owner of the file (1st group of privileges)
- the group owner of the file (2nd group of privileges)
- everybody else (3rd group of privileges)

This way, you can place one set of privileges for the file owner, another set for the group (which means everybody who is member of the group is matched against these privileges) and a third one set for everybody else.

In case of a file,

- the read privilege informs the system that the file can be read (viewed)
- the write privilege informs the system that the file can be written to (edited)
- the execute privilege informs the system that the file is a command which can be executed

As an example, see the output of the **ls -l** command:

```
$ ls -l /etc/fstab
-rw-r--r-- 1 root root 905 Nov 21 09:10 /etc/fstab
```

In the above example, the fstab file is writable by the root user (rw-) and readable by anyone else (r--).

In case of a directory,

- the read privilege informs the system that the directory's content can be viewed
- the write privilege informs the system that the directory's content can be changed (files or directories can be added or removed)
- the execute privilege informs the system that you are able to jump inside the directory (using the **cd** command)

As an example, see the output of the **ls -ld** command:

```
$ ls -ld /etc/cron.daily
drwxr-x--- 2 root root 4096 Nov 26 18:17 /etc/cron.daily/
```

In the above example, the cron.daily directory is viewable (r), writable (w) and "enterable" (x) by the root user. People in the root group have view- and enter rights (r-x) whereas all other people have no rights to view, write or enter the directory (---).

Viewing Privileges

To view the privileges on a file, you can use the long listing format support of the ls command. For instance, to view the permissions on the systems' passwd file (which contains the user account information):

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 3108 Dec 26 14:41 /etc/passwd
```

This file's permissions are read/write rights for the root user and read rights for everybody else.

The first character in the permission output shows the type of the file:

- '-': regular file
- 'd': a directory
- 'l': a symbolic link
- 'b': a block device (like /dev/sda1)
- 'c': a character device (like /dev/console)
- 'p': a named pipe
- 's': a unix domain socket

The rest of the permission output is divided in three parts: one for the file owner, one for the file owning group and one for all the rest. So, in the given example, we can read the output '-rw-r--r--' as:

1. the file is a regular file
2. the owner (root - see third field of the output) has read-write rights
3. the members of the owning group (also root - see fourth field of the output) have read rights
4. everybody else has read rights

Another example would be the privileges of the `/var/log/sandbox` directory. In this case, we also use **ls**' **-d** argument to make sure ls shows the information on the directory rather than its contents:

```
$ ls -ld /var/log/sandbox
drwxrwx--- 2 root portage 4096 Jul 14 18:47 /var/log/sandbox
```

In this case:

1. the file is a directory
2. the owner (root) has read, write and execute rights
3. the members of the owning group (portage) also have read, write and execute rights
4. everybody else can't do anything (no read, no execute and certainly no write rights)

Another method to obtain the access rights is to use the **stat** command:

```
$ stat /etc/passwd
```

```
  File: `/etc/passwd'
  Size: 3678        Blocks: 8          IO Block: 4096   regular file
Device: 808h/2056d  Inode: 3984335     Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2013-03-18 21:46:06.000000000 +0100
Modify: 2013-03-18 21:46:06.000000000 +0100
Change: 2013-03-18 21:46:06.000000000 +0100
```

In the output of the **stat** command, you notice the same access flags as we identified before (-rw-r--r-- in this case), but also a number. This number identifies the same rights in a short-hand notation.

To be able to read the number, you need to know the values of each right:

- execute rights gets the number 1
- write rights gets the number 2
- read rights gets the number 4

To get the access rights of a particular group (owner, group or everybody else), add the numbers together.

For a file with privileges (-rw-r--r--), this gives the number 644:

- 6 = 4 + 2, meaning read and write rights for the owner
- 4 = 4, meaning read rights for the group
- 4 = 4, meaning read rights for everybody else

The first 0 that we notice in **stats**' output identifies the file as having no very specific privileges.

Specific Privileges

There are a few specific privileges inside Linux as well.

The restricted deletion flag, or sticky bit, has been identified before. When set on a directory, it prevents people with write access to the directory, but not to the file, to delete the file (by default, write access to a directory means that you can delete files inside that directory regardless of their ownership). The most well-known use for this flag is for the /tmp location:

```
$ stat /tmp
  File: `/tmp'
  Size: 28672      Blocks: 56         IO Block: 4096   directory
Device: 808h/2056d Inode: 3096577     Links: 759
Access: (1777/drwxrwxrwt)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2010-01-10 17:44:04.000000000 +0100
Modify: 2013-04-02 00:04:36.000000000 +0200
Change: 2013-04-02 00:04:36.000000000 +0200
```

Another specific privilege that we have identified before is the setuid or setgid flag. When set on an executable (non-script!), the executable is executed with the rights of the owner (setuid) or owning group (setgid) instead of with the rights of the person that is executing it. That does mean that people with no root privileges can still execute commands with root privileges if those commands have the setgid flag set. For this reason, the number of executables with the setuid/setgid bit set need to be limited and well audited for possible security exposures. A nice example for this flag is /bin/mount:

```
$ stat /bin/mount
  File: `/bin/mount'
  Size: 59688      Blocks: 128        IO Block: 4096   regular file
Device: 808h/2056d Inode: 262481      Links: 1
Access: (4711/-rws--x--x)  Uid: (    0/    root)   Gid: (    0/    root)
```

```
Access: 2010-02-06 13:50:35.000000000 +0100
Modify: 2013-01-02 13:50:35.000000000 +0100
Change: 2013-01-02 13:50:43.000000000 +0100
```

Changing Privileges

To change the privileges of a file or directory, you should use the **chmod** command (**ch**ange **mod**e). Its syntax is easy enough to remember well. First, the target permissions:

- 'u' for user,
- 'g' for group, and
- 'o' for everybody else (others)

Then you can set (=), add (+) or remove (-) privileges. For instance, to make `/etc/passwd` writable for the members of the owning group:

```
# chmod g+w /etc/passwd
```

You can also combine privileges. For instance, if you want to remove write privileges for the owning group and remove read privileges for the others:

```
# chmod g-w,o-r /etc/passwd
```

Finally, you can use the numeric notation if you want as well:

```
# chmod 644 /etc/passwd
```

Changing Ownership

When you need to change the ownership of a file or directory, use the **chown** (**ch**ange **own**er) or **chgrp** (**ch**ange **gr**ou**p**) command. For instance, to change the owner of a file to the user "jack":

```
# chown jack template.txt
```

To change the owner of a file, you need to be root though - it will not help if you are the current owner. This is not true for the group though: if you are a member of the target group, you can change the owning group:

```
$ ls -l bar
-rw-r--r-- 1 swift  users   0 May 13 20:41 bar
$ chgrp dialout bar
$ ls -l bar
-rw-r--r-- 1 swift  dialout 0 May 13 20:41 bar
```

If you need to change the owner and group, you can use a single **chown** command: just separate the target owner and group with a colon, like so:

```
# chown jack:dialout template.txt
```

Attributes

Some file systems allow you to add additional attributes to files. These attributes might have influence on the permissions / usage of these files, or on how the operating system works with these files. Not many distributions use these attributes, because not all file systems support them.

Listing and Modifying Attributes

To view the attributes of a file, you can use the **lsattr** command (list attributes); to modify the attributes, use **chattr** (change attributes). As Gentoo does not have an example file, lets' create one first:

```
# touch /tmp/foo
# chattr +asS /tmp/foo
```

Now let's see what **lsattr** has to say:

```
# lsattr /tmp/foo
s-S--a---------  /tmp/foo
```

Not a big surprise, given the **chattr** command before. But what does it mean? Well, **man chattr** gives us the information we need, but here is it in short-hand:

- s: when the file is deleted, its blocks are zeroed and written back to disk (unlike regular files where only the reference to the file is deleted)
- S: when changes are made to the file, the changes are immediately synchronized to disk (no memory caching allowed)
- a: the file can only be appended (data is added to the file); changes are not allowed to existing content. Very useful for log files.

Another very interesting attribute is the immutable flag (i) that doesn't allow the file to be deleted, changed, modified, renamed or moved.

POSIX ACLs

Next to the discretionary access controls applicable to a Linux file (user, group and others), it is possible to add more access controls on files and directories through *POSIX ACLs*.

With the **getfacl** command, the access controls on a file or directory are shown, together with the POSIX access controls (if applicable). The **setfacl** command can be used to add or remove POSIX ACLs from the file or directory.

For instance, to allow the user minidlna read access on a file that he otherwise has no access to:

```
$ setfacl -m u:minidlna:r TEMPFILE
$ getfacl TEMPFILE
# file: home/swift/TEMPFILE
# owner: swift
# group: users
user::rw-
group::r--
other::r--
user:minidlna:r--
```

Supporting POSIX ACLs requires specific file system support, so it might be necessary to enable this in the kernel. Also, the file system should be mounted with the "acl" mount option.

Generic extended attributes

Files can have additional extended attributes assigned to them. POSIX ACLs for instance uses an extended attribute called `system.posix_acl_access`, whereas SELinux uses an extended attribute called `security.selinux`. Extended attributes are metadata assigned to files that are used by one or more applications for a particular function.

When the extended attribute is in the security namespace (the name starts with "`security.`") then this is a security-sensitive attribute and can only be modified by administrators or properly privileged users.

To list all extended attributes assigned to a file, use **getfattr**:

```
$ getfattr -m . -d TEMPFILE
```

```
# file: home/swift/TEMPFILE
security.selinux="staff_u:object_r:user_home_t"
```

Usually users do not need to modify extended attributes directly; instead, the application(s) supporting these extended attributes will take care of this. But it is and remains possible to directly modify extended attributes (again, if you have the proper permissions) using **setfattr**.

Locating Files

With all these locations, it might be difficult to locate a particular file. Most of the time, the file you want to locate is inside your home directory (as it is the only location where you have write privileges). However, in some cases you want to locate a particular file somewhere on your entire system.

Luckily, there are a few commands at your disposal to do so.

mlocate

The **locate** command manages and uses a database of files to help you find a particular file. Before you can use **locate**, you first need to install it (the package is called sys-apps/mlocate) and then create the file database. Also, this database is not automatically brought up to date while you modify your system, so you'll need to run this command (which is the same for creating a new database or updating an existing one) every now and then:

```
# updatedb
```

A popular way of keeping this database up to date is to use the system scheduler (called cron) which is discussed later.

When your database is build and somewhat up to date, you can locate any particular file on your filesystem using **locate**:

```
# locate make.conf
/etc/portage/make.conf
(...)
/usr/portage/local/layman/make.conf
```

As you can see, the locate command returns all files it has found where the string (in this case, "make.conf") is used in the filename, even when the file name is different.

The name *mlocate* is the name of the project that maintains the package. Earlier in history, the package of choice for the locate functionality was *slocate*.

find

The **find** command is a very important and powerful command. Unlike **locate**, it only returns live information (so it doesn't use a database). This makes searches with **find** somewhat slow, but **find**'s power isn't speed, but the options you can give to find a particular file...

Regular find patterns

The most simple **find** construct is to locate a particular file inside one or more directories. For instance, to find files or directories inside /etc whose name is dhcpd.conf (exact matches):

```
$ find /etc -name dhcpd.conf
/etc/dhcp/dhcpd.conf
```

To find files (not directories) where dhcpd is in the filename, also inside /etc directory:

```
$ find /etc -type f -name '*dhcpd*'
```

```
/etc/conf.d/dhcpd
/etc/init.d/dhcpd
/etc/udhcpd.conf
/etc/dhcp/dhcpd.conf
```

To find files in the /etc directory who have been modified within the last 7 days (read: "less than 7 days ago"):

```
$ find /etc -type f -mtime -7
/etc/mtab
/etc/adjtime
/etc/wifi-radar.conf
/etc/genkernel.conf
```

You can even find files based on their ownership. For instance, find the files in /etc that do not belong to the root user:

```
$ find /etc -type f -not -user root
```

Combining find patterns

You can also combine find patterns. For instance, find files modified within the last 7 days but whose name does not contain .conf:

```
$ find /etc -type f -mtime -7 -not -name '*.conf'
/etc/mtab
/etc/adjtime
```

Or, find the same files, but the name should also not be mtab:

```
$ find /etc -type f -mtime -7 -not \( -name '*.conf' -or -name mtab)
/etc/adjtime
```

Working with the results

With find, you can also perform tasks on the results. For instance, if you want to view the "ls -l" output against the files that find finds, you can add the -exec option. The string after -exec should contain two special character sequences:

- `'{}'` represents the file found by the find command. The command given to the -exec option is executed and '{}' is substituted with the filename.
- `\;` ends the command in the -exec clause.

```
$ find /etc -type f -mtime -7 -exec ls -l '{}' \;
```

On the Internet, you'll also find the following construction:

```
$ find /etc -type f -mtime -7 | xargs ls -l '{}'
```

The result is the same, but its behaviour is somewhat different.

When using -exec, the find command executes the command for every file it encounters. The xargs construction will attempt to execute the command as little as possible, based on the argument limits.

For instance, if the find command returns 10000 files, the command given to -exec is executed 10000 times, once for every file. With xargs, the command might be executed only a few dozen times. This is possible because xargs appends multiple files for a single command as it assumes that the command given can cope with multiple files.

Example run for find -exec:

```
ls -l file1
ls -l file2
...
ls -l file10000
```

Example run for xargs:

```
ls -l file1 file2 ... file4210
ls -l file4211 file4212 ... file9172
ls -l file9173 file9174 ... file10000
```

Exercises

1. Create a directory hierarchy somewhere in a temporary location where you can write in (for instance, a `tmp` directory in your home directory) as follows:

   ```
   $ mkdir -p tmp/test/to/work/with/privileges
   ```

   Now, recursively remove the read privileges for any user (not owner or group) for the entire structure.

2. Check out the privileges of the `/tmp` directory. How would you set these permissions on, for instance, your own `tmp` directory?

Chapter 6. Working with Processes

Process Trees

Parent and Child Relationships

Each Linux (and Unix) process has a parent (except for the top process) and can have one or more children. The relationship is crafted when a process is launched: the process that launched the new process becomes the parent of that process. As a user, you might not know what process you are currently working in. Every program is a process, being it the shell you're typing the commands in or the graphical environment you're working with.

For instance, a user who has a terminal open can have the following process structure for this terminal:

```
init
`- xterm
   `- bash
```

You can obtain a tree of running processes using the **pstree** command:

```
$ pstree
init-+-acpid
     |-4*[agetty]
     |-agiletrack---java---19*[{java}]
     |-apache2---8*[apache2]
     |-bonobo-activati---{bonobo-activati}
     |-5*[dbus-daemon]
     |-dhcpcd
     |-gconfd-2
     |-gnome-keyring-d
     |-gnome-power-man
     |-gnome-screensav
     |-gnome-settings----{gnome-settings-}
     |-4*[gnome-vfs-daemo]
     |-gnome-volume-ma
     |-gpg-agent
     |-hald---hald-runner-+-hald-addon-acpi
     |                    |-hald-addon-cpuf
     |                    `-hald-addon-stor
     |-java---15*[{java}]
     |-login---bash---startx---xinit-+-X
     |                               `-gnome-session-+-gnome-panel
     |                                               |-metacity
     |                                               |-nautilus
     |                                               `-{gnome-session}
[...]
```

Now, not every process launched immediately becomes a child of the process where it was launched from. Some processes might immediately become child of the root process, most often called **init**. The root process is the first process launched by the kernel when it boots up. It is responsible for starting the necessary services at boot time and prepare the system for its duties.

Processes that become child of the root process usually do this because they don't want to be terminated when their parent process exits or dies: when this happens, the child processes become orphaned and the init process will terminate these processes as well. So, becoming a child of the init process will ensure that the process remains available. In the above example you'll find a good example: the **dhcpcd**

command governs the IP address of the network interface through the DHCP protocol. If the process didn't continuously run, your IP address would be dismissed after a few minutes (or hours).

Process Ownership

When a process is launched (usually through a command the user entered) it, by default, obtains the user id and group id of its parent process. When a user logs on to the system, the **login** process launches a shell process with the user id and group id of the user that logged on, so every command the user launches takes the user id and group id of that user, since the parent process of every launched command is either the before mentioned shell process or one of its child processes.

Some processes however explicitly ask the Linux kernel to use a different user id and group id. This is accomplished by setting the *setuid* or *setgid* flags on the process file itself. With *setuid* (set user id) and *setgid* (set group id) the owner of the process is the owner of the file rather than the user that launched the process.

An example is the **passwd** command, used to change the password of a user:

```
$ ls -l /bin/passwd
-rws--x--x 1 root root  28956 Jul 15  2007 passwd
```

As you can see, the command file itself is owned by root. It also has the setuid bit set (see the s in `-rws--x--x`). If a user runs the **passwd** command, the command itself has root privileges rather than the privileges for the user. For the **passwd** command, this is necessary because it needs to update the password files (`/etc/passwd` and `/etc/shadow`) which are only writable by the root user (the `/etc/shadow` file is not even readable for regular users).

Viewing Process Information

Various tools exist to obtain process information. The next few chapters give a nice overview of these tools...

Process Lists

The main program to create a process list is the **ps** command. If ran inside a shell, it shows the processes that are running inside the session (meaning the processes launched from the shell, including the shell itself):

```
$ ps
  PID TTY          TIME CMD
24064 pts/3    00:00:00 bash
24116 pts/3    00:00:00 ps
```

The columns shown are:

1. PID - process id of the process
2. TTY - controlling terminal (this is Unix heritage where users were logged on through terminals, pts is a pseudoterminal)
3. TIME - the execution time the process took. In the above example, both commands hardly took any CPU time on the system (bash is the shell, which is most of the time waiting for input so not consuming any CPU time, the other one is ps which gave its results in less than a second)
4. CMD - the process name itself (the command)

Of course, several arguments to ps exist which change its behaviour. For instance, with **ps -e** you see the same information, but for all processes running on the system. With **ps -f** a few more columns are added, including the parent process id and the time the process started.

You can also limit the processes to see based on the user (**ps -u user name**), command name (**ps -C command**), really running processes (taking cpu time at the moment: **ps -r**) and more. For more information, see the ps manual page.

Another command that is often used to obtain process list information is the **top** program. The top command is an interactive command that shows you a process list, sorted by one or more values (default is CPU usage) and refreshes this list every 5 seconds (this is of course configurable):

```
top - 10:19:47 up 6 days,  6:41,  5 users,  load average: 1.00, 1.27, 0.92
Tasks: 120 total,   1 running, 119 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.2%us,  0.7%sy,  0.0%ni, 95.6%id,  0.3%wa,  0.1%hi,  0.0%si,  0.0%st
Mem:   1545408k total,  1490968k used,    54440k free,   177060k buffers
Swap:  2008084k total,      132k used,  2007952k free,   776060k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4458 haldaemo  16   0  5488 3772 2388 S  2.0  0.2   4:23.69 hald
27255 swift     15   0  2272 1064  768 R  2.0  0.1   0:00.01 top
    1 root      15   0  1612  544  468 S  0.0  0.0   0:00.48 init
    2 root      12  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      39  19     0    0    0 S  0.0  0.0   0:00.45 ksoftirqd/0
    4 root      10  -5     0    0    0 S  0.0  0.0   0:01.95 events/0
    5 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   60 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   61 root      11  -5     0    0    0 S  0.0  0.0   0:25.77 kacpid
   62 root      11  -5     0    0    0 S  0.0  0.0   0:09.60 kacpi_notify
  171 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 ata/0
  172 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 ata_aux
  173 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd
  176 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
  178 root      10  -5     0    0    0 S  0.0  0.0   0:00.01 kseriod
  196 root      10  -5     0    0    0 S  0.0  0.0   0:01.13 kswapd0
  197 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
```

There is plenty of information in the top screen...

```
top - 10:19:47 up 6 days,  6:41,  5 users,  load average: 1.00, 1.27, 0.92
```

The first line shows you the uptime of the system (this system is running for 6 days, 6 hours and 41 minutes), the number of logged on users (beware, this is not the number of different users - if a user launches 3 xterms inside a graphical session he will be shown as four logged on users) and the load average.

The load average is something many people misinterpret. The load average shows the number of processes that were running or asking for CPU time during the given interval. In the above example, this means that:

- in the last minute, an average of 1 process was asking for or using CPU time
- in the last 5 minutes, an average of 1.27 processes were asking for or using CPU time
- in the last 15 minutes, an average of 0.92 processes were asking for or using CPU time

For a single CPU system, you most likely don't want a number higher than 1 in the long run (for instance, the 15-minute span). The more CPUs, the higher the load average can become.

```
Tasks: 120 total,   1 running, 119 sleeping,   0 stopped,   0 zombie
```

The number of processes running on this system (120) of which 119 are sleeping (not performing any duties), 1 running (the top command itself), 0 stopped (a process in the stopped state can still be revived but is, at this moment, not accepting input or performing any tasks) and 0 zombie.

A zombie process is not really a real process: the process itself has already finished, but its parent process doesn't know this yet, so the kernel is keeping some process information until the parent process asks for the child process state.

```
Cpu(s):  3.2%us,  0.7%sy,  0.0%ni, 95.6%id,  0.3%wa,  0.1%hi,  0.0%si,  0.0%st
```

CPU state information, showing the CPU usage percentages: user processes (us), system/kernel CPU usage (sy), niced processes (ni), idle CPU (id), waiting for I/O (wa), hardware interrupts (hi), software interrupts (si) and virtual cpu stealing (st).

Most of the states are self-explanatory. The niced processes is for processes the user reniced and is a subset of the user processes percentage. The virtual CPU stealing is the percentage of time a virtual CPU waits for a real CPU and is not interesting for regular Linux/Unix users (as they don't work with virtualization).

```
Mem:   1545408k total,  1490968k used,    54440k free,   177060k buffers
Swap:  2008084k total,      132k used,  2007952k free,   776060k cached
```

Memory usage: of the 1.5 Gbyte of memory available, 1.45Gbyte is in use and 54Mbyte is free. Of the used memory, 177 Mbyte is used by the kernel for internal buffers. Also, 776 Mbyte of the used memory actually consists out of cached data which can potentially be cleared if a process would require more memory than currently available.

The swap space itself is hardly used: of the 2Gbyte of swap space defined, only 132 kbyte is in use.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4458 haldaemo  16   0  5488 3772 2388 S  2.0  0.2   4:23.69 hald
...
```

The rest of the screen gives the process listing itself. The columns shown are:

1. Process ID (PID) of the process
2. Username (USER) showing the owner of the process
3. Priority value (PR) of the process (the higher the value, the higher the priority). Priorities are exclusively determined by the Linux kernel.
4. Nice value (NI) of the process (is a user sets a nice value, or renices a tool, it tells the Linux kernel how "nice" the program is - the higher the nice value, the nicer it is so (generally) the lower the priority should be).
5. The virtual memory (VIRT) the process is occupying. This includes the memory it is actually using, mapped memory from devices, files mapped into memory and shared memory.
6. The resident (really used) memory (RES) the process is using.
7. The amount of possibly shared memory (SHR). It is "possibly" because the memory is shareable, but not automatically used by others already.
8. Process state (S), which can be any of S (sleeping), R (running), D (uninterruptible sleep), T (traced or stopped) or Z (zombie).
9. CPU usage (%CPU)
10. Memory usage (%MEM - based on RES)
11. Runtime (TIME+)
12. Command (COMMAND)

Process Information

You can also be interested in more detailed process information such as the files (or connections) the process has currently open.

With **lsof** you can view this information. Just give the process id with it (lsof -p PID) and you get all this information. However, lsof can do much more. For instance, with lsof you can see what process is listening on a particular port:

```
# lsof -i :443
COMMAND   PID   USER   FD   TYPE DEVICE SIZE NODE NAME
apache2  4346   root    3u  IPv4  11484       TCP *:https (LISTEN)
```

Another tool that can do the same is **fuser**:

```
# fuser -v 443/tcp
                     USER        PID ACCESS COMMAND
443/tcp:             root       4346 F.... apache2
```

The same can be accomplished with files. For instance, to see what processes are using a particular file with fuser, just give the file name (**fuser -v /path/to/file**).

Backgrounding Processes

Processes can be started in the background, either because the process immediately detaches it from the running session (daemons) or because the user asks to run it in the background.

Daemons are processes that do not stay in the running session. The moment you launch a daemon process, you immediately get your prompt back as if the process has finished. However, this isn't true: the process is still running, but it is running in the background. Most daemons do not have the possibility to reattach to the session. Whether or not a process is a daemon depends on the process itself as this is a pure programmatic decision.

Backgrounded processes however are processes that stay in the running session, but do not "lock" the input devices (keyboard). As a result, the user gets the prompt back and can continue launching other processes or do other tasks. To background a process, a user can add a "&" sign at the end of the command line. For instance, to put the command "eix-update" in the background:

```
# eix-update &
```

You can see what processes are running in your session in the background using the **jobs** command:

```
# jobs
[1]- Running           eix-update &
```

You can put a job back into the foreground using the **fg** command. If you just enter fg, it'll put the last job put in the background back. If you want to select a different job, use the number that jobs returned. For instance, to return the 3rd job back to the foreground:

```
# fg %3
```

If you want to put a process that you are running in the background, use Ctrl-Z. Ctrl-Z also pauses the process, so if you want to continue the process in the background, enter "**bg**" afterwards as well:

```
# eix-update
(...)
(Press Ctrl-Z)
[1]+ Stopped         eix-update
# bg
```

```
[1]+ eix-update &
```

There are a few things you must keep in mind when using jobs:

- A (non-daemon) process is attached to a running session. The moment you terminate your session, all jobs that were running in that session (both foreground and background processes) are terminated as well.
- Although processes can be ran in the background, their output is still forwarded to your terminal. If you do not want this, you can redirect the output of a command using the > redirect. For instance, to redirect the standard output (default - 1) of update-eix to a log file and do the same for the error output (2):

  ```
  # eix-update > /var/tmp/update-eix.log 2>&1 &
  ```

  Another popular redirect is to ignore output completely:

  ```
  # eix-update > /dev/null 2>&1 &
  ```

Process Behaviour

Programs are most often launched when a user selects a tool or executes a command. They can also be invoked automatically by a running program or by the Linux kernel (although the init tool is probably the only one ever invoked by the kernel autonomously).

The next few sections give pointers on process behaviour and how you can modify it (if appropriate).

Command Return Codes

The simplest example of launching a program is a simple command you enter in the command line. Once the program has finished, it leaves behind its *return code* (or *exit code*) informing you how well it did its job.

A return code is always an integer in the range of 0 to 255. Some programs might attempt to return a code larger than 255 (or even negative). Although not technically restricted, this is not a good idea as some applications only expect a return code between 0 to 255 and might even "wrap" return codes to this range. If a program would ever have a return code of 512 for instance, it might be mapped into 0.

Every program that has successfully finished its job will (or should) return code 0. A non-zero return code means that the application has failed to finish its tasks (completely).

Inside any POSIX-compliant shell (POSIX has a standard for Unix environments, including how a shell should function) such as **ksh** or **bash** you can obtain the return code of the last command using $?:

```
$ ls -l
...
$ echo $?
0
$ ls -z
ls: invalid option -- z
Try `ls --help' for more information
$ echo $?
2
```

These return codes are important as they are the means to investigate if all commands went successfully or not, allowing you to write quite intelligent shell scripts which trigger several commands and include logic to handle command failures.

Priority and Niceness

On Linux, you can't set the priority of a process yourself: the Linux kernel does that for you, based on information about the process itself, including but not limited to if the process is I/O-bound (as such programs are most of the time user-interactive), its previous CPU consumption, possible locks it is holding and more.

You can, however, inform the kernel on what you think the process' priority ought to be. For this, you can set a *nice* value for the application. The value, in the range of -20 to 19, informs the Linux kernel about how nice the program should be towards the rest of the system. Negative numbers (-1 to -20) are not that nice; the Linux kernel will thus assign those a larger time slice and you'll notice that such programs usually get a higher priority. However, only the root user can assign a negative nice number to a program. Positive numbers (1 to 19) make a process more nice to the system; they will receive a lower time slice and usually a lower priority.

Thanks to this system you can launch long-lasting, non-interactive commands in the background without worrying about the inpact to your interactive user experience. The **nice** tool allows you to start up a command with a particular nice value.

For instance, to start a Gentoo system upgrade with the highest possible nice value (as this is something you usually want to perform in the background):

```
# nice -n 19 emerge -uDN @world
```

If a process is already running, you can change its nice value with the renice tool (for instance, to increase the nice value of the process with process id 219 with 5):

```
# renice +5 219
```

Sending Signals (and Killing Processes)

Some processes allow you to send certain signals to them. A signal is a simple integer between 0 and 64; each of them is also given a particular name. The **kill** tool can be used to send signals to processes, but also to obtain a list of available signals:

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Its name might already inform you about its usual task: killing processes. By default, if you want to terminate a process but you can't communicate with the process directly (like hitting "Quit" or "Exit"), you should send a signal 15 (SIGTERM) to the program. This is also what **kill** does by default.

However, if the process doesn't listen to this signal or has gone haywire, you can use the SIGKILL signal. The SIGKILL signal doesn't actually reach the application (ever) but immediately terminates the process. Its number is 9:

```
$ kill -9 219
```

Exercises

1. How do you obtain the process ID of a running process?
2. How can you run a process in background and still be able to terminate the session without terminating the process (without the process being a daemon)?
3. What is a <defunct> process?

Further Resources

- Bash redirection [http://www.gnu.org/software/bash/manual/bashref.html#Redirections]

Chapter 7. Configuring a Linux Kernel

Introduction

The Linux kernel is responsible for the majority of the hardware support. It contains the drivers to access and communicate with hardware. However, there are also quite some user-space tools required in order to successfully work with the hardware. For instance, support for printing inside the kernel is limited to support to the LPT printer port or USB, but the printer drivers themselves are user-space.

Another example are scanners: inside the kernel, you need either USB or SCSI support, but there are no drivers for scanners themselves. Those are available in the user-space.

Terms like *user-space* (everything running as a regular process or data found in non-kernel-reserved memory) and *kernel-space* (in-kernel tasks or data in kernel-reserved memory) are not that frequently used in documentation about applications, but for drivers, the distinction is quite important. Everything in kernel-space can have quite an impact to performance (it must be fully tuned and proofread by various developers before it is accepted). As such, if particular drivers can be made part of a user-space design (this is not often possible) it is preferred to make them user-space drivers.

To support all the hardware in your system well, you first need to identify what hardware is inside your system. Then, you need to find the correct configurations inside the Linux kernel that match against the hardware. Before building the Linux kernel, you also need to include support for the non-hardware related features you need (such as filesystem support). Once the kernel is built, you can store it to disk and update the boot loader to use the new kernel. Finally, you install and configure the user-space tools needed to deal with your hardware.

Obtaining Hardware Information

Introduction

If you don't know what hardware is inside your system (and I'd be very surprised if you know all components, including vendor and type of your system) you will be glad to know Linux has quite some interesting tools to tell you what it found. The results of these tools are regardless of whether or not your kernel has support for these devices (although some small support is required, but most, if not all default kernels have this included).

Device Information

Processor

Information regarding the processor(s) in your system can be found inside the `/proc/cpuinfo` file. This file is not a real file but a representation of what the Linux kernel finds in your system. In effect, every file in the `/proc` file system is a representation of the kernel's information and every time you read it it gives you the last known state.

```
$ cat /proc/cpuinfo
processor    : 0
vendor_id    : GenuineIntel
cpu family   : 6
model        : 13
model name   : Intel(R) Pentium(R) M processor 1.73GHz
stepping     : 8
cpu MHz      : 800.000
cache size   : 2048 KB
```

```
fdiv_bug     : no
hlt_bug      : no
f00f_bug     : no
coma_bug     : no
fpu          : yes
fpu_exception: yes
cpuid level  : 2
wp           : yes
flags        : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
               mca cmov pat clflush dts acpi mmx fxsr sse sse2 ss
               tm pbe nx est tm2
bogomips     : 1597.26
clflush size : 64
```

The information might overwhelm you - don't let it. The most interesting information here is the model name as it effectively tells you what brand and type of CPU this is. If you have multiple CPUs inside your system (or multiple cores) then you'll find multiple entries - one for each CPU / core.

Memory

Similar to CPU information, you can view the information from the `/proc/meminfo` file to obtain information regarding your memory. Or, you can use the **free** command to obtain the current memory usage statistics - including the total amount of memory your system has.

PCI Devices

PCI device information can be obtained using the **lspci** tool. Just entering **lspci** already gives you a nice overview of the PCI devices found inside your system. I recommend to use **lspci -k** as it will also display which kernel drivers are already used to manage the PCI device:

```
# lspci -k
00:00.0 Host bridge: Intel Corporation Mobile 915GM/PM/GMS/910GML Express Proce
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: agpgart-intel
00:02.0 VGA compatible controller: Intel Corporation Mobile 915GM/GMS/910GML Ex
        Subsystem: Fujitsu Technology Solutions Device 107d
00:02.1 Display controller: Intel Corporation Mobile 915GM/GMS/910GML Express G
        Subsystem: Fujitsu Technology Solutions Device 107d
00:1c.0 PCI bridge: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI E
        Kernel driver in use: pcieport
00:1c.1 PCI bridge: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI E
        Kernel driver in use: pcieport
00:1d.0 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: uhci_hcd
00:1d.1 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: uhci_hcd
00:1d.2 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: uhci_hcd
00:1d.3 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: uhci_hcd
00:1d.7 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) US
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: ehci_hcd
00:1e.0 PCI bridge: Intel Corporation 82801 Mobile PCI Bridge (rev d4)
```

```
00:1e.2 Multimedia audio controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (I
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: Intel ICH
00:1e.3 Modem: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) AC'97 Mode
        Subsystem: Fujitsu Technology Solutions Device 107d
00:1f.0 ISA bridge: Intel Corporation 82801FBM (ICH6M) LPC Interface Bridge (re
        Subsystem: Fujitsu Technology Solutions Device 107d
00:1f.1 IDE interface: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) ID
        Subsystem: Fujitsu Technology Solutions Device 107d
00:1f.2 SATA controller: Intel Corporation 82801FBM (ICH6M) SATA Controller (re
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: ahci
00:1f.3 SMBus: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) SMBus Cont
        Subsystem: Fujitsu Technology Solutions Device 107d
06:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8169 Gigabit E
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: r8169
06:04.0 Network controller: Intel Corporation PRO/Wireless 2200BG [Calexico2] N
        Subsystem: Intel Corporation Device 2702
        Kernel driver in use: ipw2200
        Kernel modules: ipw2200
06:09.0 CardBus bridge: Texas Instruments PCIxx21/x515 Cardbus Controller
        Subsystem: Fujitsu Technology Solutions Device 107d
06:09.2 FireWire (IEEE 1394): Texas Instruments OHCI Compliant IEEE 1394 Host C
        Subsystem: Fujitsu Technology Solutions Device 107d
06:09.3 Mass storage controller: Texas Instruments PCIxx21 Integrated FlashMedi
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: tifm_7xx1
        Kernel modules: tifm_7xx1
06:09.4 SD Host controller: Texas Instruments PCI6411/6421/6611/6621/7411/7421/
        Subsystem: Fujitsu Technology Solutions Device 107d
        Kernel driver in use: sdhci-pci
        Kernel modules: sdhci-pci
```

This gives all the information you might need to know about your PCI devices. For instance, to find out what graphical controller you have (video card), look for the "VGA compatible controller" entry.

The kernel driver/module information offered in the output is only available if the mentioned driver is in fact used on the system. When you are installing a Linux system from another Linux environment, chances are that this is indeed already the case.

USB Devices

Similar to the **lspci** tool, there is also an **lsusb** tool showing you which USB devices are attached to your system:

```
# lsusb
Bus 001 Device 001: ID 1d6b:0002
Bus 005 Device 001: ID 1d6b:0001
Bus 004 Device 001: ID 1d6b:0001
Bus 003 Device 002: ID 04b0:012e Nikon Corp. Coolpix 5600 (ptp)
Bus 003 Device 001: ID 1d6b:0001
Bus 002 Device 001: ID 1d6b:0001
```

To find out which device file is used, check the dmesg output (dmesg returns the kernel message output):

```
# dmesg | tail
sd 4:0:0:0: [sdb] Mode Sense: 18 00 00 08
```

```
sd 4:0:0:0: [sdb] Assuming drive cache: write through
sd 4:0:0:0: [sdb] 1990656 512-byte hardware sectors: (1.01 GB/972 MiB)
sd 4:0:0:0: [sdb] Write Protect is off
sd 4:0:0:0: [sdb] Mode Sense: 18 00 00 08
sd 4:0:0:0: [sdb] Assuming drive cache: write through
 sdb: sdb1
sd 4:0:0:0: [sdb] Attached SCSI removable disk
sd 4:0:0:0: Attached scsi generic sg1 type 0
usb-storage: device scan complete
```

In this example, the device is known as /dev/sdb and has one partition (/dev/sdb1).

Configuring a Linux Kernel

There are two ways you can configure a Linux kernel in Gentoo Linux. One is the easiest method, yet not 100% reliable regarding supported hardware: it is a script Gentoo provides that builds a Linux kernel with a generic configuration and lots of loadable modules to include as much hardware support as possible. It works well for standard setups (both hardware and system setup) but might fail for more complex systems. The second method is the most reliable - you configure everything manually yourself. If it fails, it's most likely your own fault.

But before we focus on these methods, a small introduction to the Linux kernel modules administration-wise.

Kernel Modules

The Linux kernel is a file loaded into memory by the boot loader. However, Linux supports loadable kernel modules.

A *Linux kernel module* is a part of the Linux kernel which can be dynamically loaded and unloaded from memory. As such, it is possible to build a kernel which supports certain hardware, but doesn't load support for these modules until absolutely necessary.

Kernel modules are often used for detachable devices (such as USB devices) but also for distributions who want to provide a kernel to their users regardless of the hardware these users own: during the boot-up sequence, the distribution will load the modules needed to support the hardware found and leave all other modules untouched.

It is important to understand though that these modules are read from somewhere - you can't put the driver for your disk in a module and store that module on disk - the Linux kernel doesn't know what a disk is at that point and can therefore not find the module. Similar, don't put support for a particular file system (such as ext3) in a module and store that module on an ext3 file system.

To be able to use kernel modules even though they might be required to access devices such as disks, you can create an intermediate root disk (*initial root disk* or *initrd* or *initial ram file system* or *initramfs*). This contains the modules that might be needed in order to successfully boot the system. Support for the initrd or initramfs format is built in-kernel (in-kernel is the term denoting that support is not built as a kernel module but immediately made part of the Linux kernel image loaded by the boot loader).

The boot loader itself is responsible for placing the initrd or initramfs in memory and informing the Linux kernel where the initial root disk/file system can be found.

Working with Modules

The three most important commands to use for kernel modules manipulation are **lsmod**, **modprobe** and **rmmod**.

To list the currently loaded modules, use **lsmod**:

```
# lsmod
Module                  Size  Used by
ieee80211_crypt_wep     3584  1
i915                   29184  2
drm                    69664  3 i915
ieee80211_crypt_tkip    8896  0
pcmcia                 31764  0
sdhci_pci               6976  0
sdhci                  14660  1 sdhci_pci
yenta_socket           22860  1
ipw2200               133492  0
mmc_core               42068  1 sdhci
rsrc_nonstatic          8960  1 yenta_socket
pcmcia_core            31248  3 pcmcia,yenta_socket,rsrc_nonstatic
tifm_7xx1               5568  0
tifm_core               6420  1 tifm_7xx1
```

The **lsmod** output gives you what modules are loaded, the size it takes in memory and how many/what other modules use it.

To remove a loaded module, first ensure that the module is not used by any other module any more (i.e. the "Used by" listing should be empty in the **lsmod** output) and then use **rmmod**:

```
# rmmod ipw2200
```

If you want to load a module in memory, you should use **modprobe**[1]:

```
# modprobe ipw2200
```

One of the advantages of using modules is that you can pass on additional options to a module, effectively changing its behaviour. But first some information on getting more information on modules.

An interesting command is **modinfo**, which displays information about a module. Although its output might seem cryptic, it nevertheless gives some interesting pointers to what options a module supports.

```
# modinfo uvcvideo
filename:       /lib/modules/3.1.6/kernel/drivers/media/video/uvc/uvcvideo.ko
version:        v0.1.0
license:        GPL
description:    USB Video Class driver
author:         Laurent Pinchart <laurent.pinchart@skynet.be>
srcversion:     6B23A0D849FE5EC0262441F
alias:          usb:v*p*d*dc*dsc*dp*ic0Eisc01ip00*
alias:          usb:v1C4Fp3000d*dc*dsc*dp*ic0Eisc01ip00*
...
depends:
vermagic:       3.1.6 SMP preempt mod_unload
parm:           clock:Video buffers timestamp clock
parm:           nodrop:Don't drop incomplete frames (uint)
parm:           quirks:Forced device quirks (uint)
parm:           trace:Trace level bitmask (uint)
parm:           timeout:Streaming control requests timeout (uint)
```

The above example shows that the uvcvideo module uses the GPL license, is the USB Video Class driver, ... and supports parameters clock, nodrop, quirks, trace and timeout. Of course, it does

[1]There is another command called insmod - this tool is less intelligent than modprobe: it only attempts to load a kernel module given its file name and it doesn't try to load the prerequirement modules (modules can prerequire other modules).

not give you information what these parameters mean, but that is what you have Google [http://www.google.com] for, right?

Anyway, suppose you want to load the uvcvideo module with the *nodrop=1* argument (you might be instructed to do so from a bugreport or forum posting). You can do this using modprobe:

```
# modprobe uvcvideo nodrop=1
```

This does not make the option permanent though: when you reboot, the module will be loaded in without this option. And repeatedly rmmod'ing the module just to load it again with a parameter is quite resource-intensive. The solution is to tell modprobe that a particular option should be set by default. This can be accomplished by setting the necessary directives in `/etc/modprobe.conf` or in a file inside the /etc/modprobe.d directory. The latter is more often used: people or projects create a file inside `/etc/modprobe.d` named after the module itself, which makes management easier.

The content of `/etc/modprobe.d/uvcvideo` in the previous example would then be:

```
options uvcvideo nodrop=1
```

Loading Modules

By default, Linux will load the modules you need for your system. It relies on hardware detection (modern hardware always exposes information about itself on a standardized manner) combined with the drivers of the kernel modules, which describe for which hardware they are made (and which other modules they require). What happens then is that udev (we'll talk about udev in the next chapter) gets informed by the Linux kernel about new hardware being detected. Udev then triggers a modprobe for that hardware, and based on the module information, modprobe knows which module to load.

The module information is generated by **depmod** when kernel modules are installed (copied) to a system. That's a "nice-to-know" item, because it will happen transparently for the user (you'll most likely never need to run **depmod** yourself).

However, not all drivers know up-front which hardware they support (some hardware doesn't expose this information) or they collide with other drivers and, as a result, a system is configured not to autoload any module for that hardware. When you are in such a situation, you can still instruct your system to automatically load a particular kernel module at boot time. All you need to do is add the kernel module name to `/etc/conf.d/modules`. The file is self-explanatory, but below you can find an example for auto-loading the ipw2200 module.

```
modules="ipw2200"
```

You can also ask your system not to automatically load a particular module. To accomplish this, add the kernel module name as a blacklist inside `/etc/modprobe.d/blacklist.conf`

```
blacklist uvcvideo
```

Blacklisting a module doesn't mean that the module cannot be loaded any more. It means that the automated load, based on the hardware information, is disabled. In other words, the link between a hardware device id and the module is blacklisted. A manual **modprobe** will still do the trick, and if the hardware isn't managed by another module yet, then this module will handle the hardware device.

Using Gentoo's genkernel Script

Gentoo has a script called **genkernel** which builds a complete Linux kernel based on some default configuration. Most hardware support is put in loadable modules although the configuration attempts to put all required device support in-kernel.

To use genkernel, we first need to install it. As root, execute the following command (software installation is covered later in this book):

```
# emerge genkernel
```

The installation shouldn't take long. Once finished, run the genkernel tool with the all argument (this assumes that a Linux kernel source code is already installed - if Gentoo has been installed, this is already the case):

```
# genkernel all
```

Once finished, the script will have created the Linux kernel itself, the various modules and an initrd file containing the necessary modules that might be required to boot the system. The Linux kernel file and initial root disk are then used by the boot loader to boot up Linux.

Simple, isn't it?

Manually Configuring a Kernel

Building a Linux kernel can take a while, but the majority of time comes in the configuration of the kernel. Especially the first Linux kernel configuration might even take over an hour to complete because there are so many options to select (or deselect). Luckily, configurations can be saved and reused later on, so subsequent configurations will be much faster.

Even more, Pappy McFae, a vivid Gentoo user, is providing kernel seeds to Gentoo users. A *kernel seed* is, in his words:

> [a part of a kernel configuration where] all the tough stuff has already been set up. The basics for a standard desktop system configuration, basic network operation, and other settings have been set for stable systemic usability. All you, the user, have left to do is to plug in the required hardware devices, and you have a functional kernel.

His idea is really worth pursuing, so don't hesitate to take a look at http://www.kernel-seeds.org/. On the site, you can then navigate to your architecture (currently only x86 and x86_64 are supported), select the kernel tree you have installed (most likely gentoo if you have installed the gentoo-sources) and download the configuration file for the selected kernel version.

Next, copy the configuration file to the kernel configuration location:

```
# cp /path/to/3.8.5-gentoo-r1-x86-07.config /usr/src/linux/.config
```

This will make sure that the kernel configuration utility uses the settings from the file as the "default".

Loading the Kernel Configuration Utility

The Linux kernel provides its own configuration utility; if you go to the Linux source code and type in "make menuconfig" you'll be greeted with the index of the Linux kernel configuration:

```
# cd /usr/src/linux
# make menuconfig
```

Note

The visual representation (more specifically, the location of the menu entries) might be different on your system. The location of settings changes very often and chances are that the kernel version you are using is different from the one this chapter is based upon.

```
---------------  Linux Kernel Configuration  ---------------
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N>
excludes, <M> modularizes features. Press <Esc><Esc> to exit,
<?> for Help, </> for Search. Legent: [*] built-in,
```

```
[ ] excluded, <M> module, < > excluded module

      General setup --->
  [*] Enable loadable module support --->
  -*- Enable the block layer --->
      Processor type and features --->
      Power management and ACPI options --->
      Bus options (PCI etc.) --->
      Executable file formats / Emulations --->
  [*] Networking support --->
      Device Drivers --->
      Firmware Drivers --->
      File systems --->
      Kernel hacking --->
      Security options --->
  -*- Cryptographic API --->
  [*] Virtualization --->
      Library routines --->
---
  Load an Alternate Configuration File
  Save an Alternate Configuratio File

  <Select>  <Exit>  <Help>
```

This tool doesn't do much more than edit a configuration file called `/usr/src/linux/.config` so it will not reconfigure your current kernel setup. As such, I recommend you to try it out. If you want to be absolutely sure that you don't mess anything up, you can make a backup of the current configuration file prior to starting the menuconfig:

```
# cp .config .config~
```

A new feature, available from Linux 2.6.32 onwards, is the make option **localmodconfig** (and a derivative called **localyesconfig**). These options are interesting if you are currently running a LiveCD or different distribution with a default distribution kernel (a fits-all kernel that uses kernel modules) as they will enable the drivers that are currently loaded (when you execute the commands) in the kernel configuration, giving a nice configuration to start from. This is especially true with Pappy's kernel seeds ;-)

```
# make localmodconfig
```

The **localmodconfig** enables the drivers as kernel modules. The **localyesconfig** target enables the drivers as in-kernel selections (no modules).

You should understand that your current Linux kernel (the one you're booted with) is stored elsewhere in the system (probably in `/boot`) and that this kernel doesn't need to know its configuration file any more - it has been built using a particular configuration and will remain as-is.

Configuring your own kernel is most likely a trial-and-error process: you configure a new kernel, boot it, play around and if you're not satisfied (or it doesn't boot at all), just reboot with the old kernel (having several kernels next to each other is not a problem).

The menuconfig utility has an extensive help system built in. For instance, select "Enable loadable module support" and select <Help> (or press "?"):

```
------------   Enable loadable module support ----------
CONFIG_MODULES:

Kernel modules are small pieces of compiled code which can
be inserted in the running kernel, rather than being
permanently built into the kernel. You use the "modprobe"
```

```
tool to add (and sometimes remove) them. If you say Y here,
many parts of the kenel can be built as modules (by
answering M instead of Y where indicated): this is most
useful for infrequently used options which are not required
for booting. For more information, see the man pages for
modprobe, lsmod, modinfo, insmod and rmmod.

If you say Y here, you will need to run "make
modules_install" to put the modules under /lib/modules/
where modprobe can find them (you may need to be root to do
this).

If unsure, say Y.

Symbol: MODULES [=y]
Prompt: Enable loadable module support
  Defined at init/Kconfig:607
  Location:
    -> Loadable module support
```

As you can see, the system gives information about the selected option as well as an indication whether or not you should select this ("If unsure, say Y.").

The system also has a search system. Press "/" and type in "initrd" to look for the configuration entry where you can enable initrd support inside the kernel:

```
------------   Search Results   ------------
Symbol: BLK_DEV_INITRD [=n]
Prompt: Initial RAM filesystem and RAM disk (initramfs/initrd) support
  Defined at init/Kconfig:326
  Depends on: BROKEN || !FRV
  Location:
    -> General setup
```

The search results give you a one-liner explanation of the found results as well as where you can find the option (here: in the "General setup" tab). It also shows you when the option is selectable (when the configuration BROKEN is set, or when FRV is not selected - FRV is an architecture, just as x86 and SPARC are - so for a non-FRV architecture it is always selectable).

In certain cases it can also tell you what new options it will allow once selected.

Recommended Linux Kernel Configurations

Although this might be a bit boring for some people, I'll discuss what I see as recommended Linux kernel configurations regardless of the hardware support you add on later...

General Setup

In "General Setup", miscellaneous configurations for the Linux kernel can be found.

```
[ ] Prompt for development and/or incomplete code/drivers
()  Local version - append to kernel release
[ ] Automatically append version information to the version string
    Kernel compression mode (Gzip) -->
[*] Support for paging of anonymous memory (swap)
[*] System V IPC
[ ] BSD Process Accounting
[ ] Auditing Support
    RCU Subsystem --->
```

```
<*> Kernel .config support
[*]   Enable access to .config through /proc/config.gz
(16) Kernel log buffer size (16 => 64KB, 17 => 128 KB)
[ ] Control Group support --->
[ ] enable deprecated sysfs features to support old userspace tools
[ ] Kernel->user space relay support (formerly relayfs)
-*- Namespaces support
[ ]   UTS namespace
[ ]   IPC namespace
[ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
[ ] Optimize for size
[ ] Configure standard kernel features (for small systems) --->
    Kernel Performance Events And Counters --->
[ ] Disable heap randomization
    Choose SLAB allocator (SLUB (Unqueued Allocator)) --->
[ ] Profiling support
[ ] Kprobes
    GCOV-based kernel profiling --->
[ ] Slow work debugging through debugfs
```

The following configurations are recommended to be enabled in this section:

- Support for paging of anonymous memory (swap)

  You will need this enabled if you want to enable swap space on your system. You generally want this, unless you're confident that you have enough memory (RAM) in your system for every possible situation. Swap space is used when free memory is needed but not available. In that case, the Linux kernel will move out old pages of memory (which are most likely not going to be used in the near future) to the swap space.

- System V IPC

  IPC (Inter Process Communication) allows programs to share and exchange information between them. Many programs on a Linux system will not start if System V IPC support isn't enabled in the kernel. System V IPC allows programs to use message queues, semaphores and shared memory segments.

- RCU Subsystem

  RCU (Read Copy Update) is a synchronisation primitive supported by the Linux kernel which offers fast access to shared resources (programming-terms) in case of a many-read and infrequent write access behaviour. That might sound terribly geekish and programmer-specific - and it is - but if you have multiple cores or processors in your system, it is wise to enable it. Otherwise, set it to UP-kernel (UniProcessor).

  ```
      RCU Implementation (Tree-based hierarchical RCU) --->
  [ ] Enable tracing for RCU
  (32) Tree-based hierarchical RCU fanout value
  [ ] Disable tree-based hierarchical RCU auto-balancing
  ```

- Kernel .config support

  Although definitely not mandatory for a kernel, building in .config support allows you to obtain the configuration for a running kernel from the kernel itself. This can come in handy if you don't keep track of the configurations you use(d) for kernels. You can, for instance, base a new kernel configuration on this configuration to get a good start.

  The subselection to support `/proc/config.gz` is an easy-to-use interface to the kernel configuration of a running kernel: extract /proc/config.gz (for instance, **zcat /proc/config.gz > /usr/src/linux/.config** and you have this kernel's configuration at hand.

You also notice that initramfs support is not enabled: I'm no proponent of initrd's, it is in my opinion better that a user configures his kernel to his system rather than hoping that an initrd will help with configuration failures.

Enable Loadable Module Support

I recommend enabling loadable module support if you use or will use detachable devices (such as USB devices).

```
--- Enable loadable module support
[ ]   Forced module loading
[*]   Module unloading
[ ]   Module versioning support
[ ]   Source checksum for all modules
```

The following settings are recommended:

- Module unloading

  You will probably want to unload kernel modules if you don't need them any more.

Enable the Block Layer

I recommend enabling the block layer as you'll most likely want to use block devices or Linux kernel components that use functions from the block layer (such as SCSI or SCSI emulating devices, the ext3 file system or USB storage).

```
--- Enable the block layer
[ ]   Block layer SG support v4
[ ]   Block layer data integrity support
      IO Schedulers --->

------------   IO Schedulers   ------------
<*> Deadline I/O scheduler
<*> CFQ I/O scheduler
    Default I/O scheduler (CFQ) --->
```

The important setting here are the IO schedulers. These control how (and when) the Linux kernel writes or reads data to/from disks. There are different IO schedulers available because, depending on the system's use, a specific implementation can give a nice performance boost. The CFQ scheduler is a good implementation for desktop systems.

Processor Type and Features

The following settings are the recommended settings. However, these settings depend heavily on the system you have (as your CPU is most likely different from the one I use):

```
[*] Tickless System (Dynamic Ticks)
[*] High Resolution Timer Support
[*] Symmetric multi-processing support
[ ] Support sparse irq numbering
[ ] Enable MPS table
[ ] Support for extended (non-PC) x86 platforms
[*] Single-depth WCHAN output
[ ] Paravirtualized guest support
[*] Disable bootmem code
[ ] Memtest
    Processor family (Core 2/newer Xeon) --->
[ ] AMD IOMMU support
```

```
(8) Maximum number of CPUs
[*] SMT (Hyperthreading) scheduler support
[*] Multi-core scheduler support
    Preemption Model (Preemptible Kernel (Low-Latency Desktop)) --->
[ ] Reroute for broken boot IRQs
[*] Machine Check / overheating reporting
[*]   Intel MCE features
[ ]   AMD MCE features
< > Machine check injector support
< > Dell laptop support
[ ] Enable X86 board specific fixups for reboot
< > /dev/cpu/microcode - Intel IA32 CPU microcode support
< > /dev/cpu/*/msr - Model-specific register support
< > /dev/cpu/*/cpuid - CPU information support
[ ] Numa Memory Allocation and Scheduler Support
    Memory model (Sparse Memory) --->
[*] Sparse Memory virtual memmap
[ ] Allow for memory hot-add
[ ] Enable KSM for page merging
(65536) Low address space to protect from user allocation
[ ] Enable recovery from hardware memory errors
[ ] Check for low memory corruption
[ ] Reserve low 64K of RAM on AMI/Phoenix BIOSen
-*- MTRR (Memory Type Range Register)
[ ]   MTRR cleanup support
[ ] EFI runtime service support
[*] Enable seccomp to safely compute untrusted bytecode
    Timer frequency (1000 HZ) --->
[ ] kexec system call
[ ] kernel crash dumps
[ ] Build a relocatable kernel
-*- Support for hot-pluggable CPUs
[ ] Compat VDSO support
[ ] Built-in kernel command line
```

The following settings are recommended:

- Tickless System (Dynamic Ticks)

  Unless you need the shortest latency possible, using dynamic ticks will ensure that timer interrupts only fire when needed.

- High Resolution Timer Support

  Most relatively modern systems (Pentium III and higher) have high resolution timers, allowing for more precise timing. Not really mandatory, but some applications like mplayer can benefit from using hi-res timers.

- Symmetric multi-processing support

  If you have multiple (identical) CPUs or your CPU has multiple cores, enable this.

- Single-depth WCHAN output

  WCHAN is the abbreviation for "waiting channel" and identifies where tasks are currently waiting for. With this enabled, the calculation for the waiting channel is simplified at the expense of accuracy. Most users don't need this level of accuracy and the simplifications means less scheduling overhead.

- Disable bootmem code

This optimizes some complex initial memory allocation fragments within the Linux kernel.

- Processor family (Pentium M)

  I have selected "Pentium M" here as this is my CPU type (see the `/proc/cpuinfo` information). You should select the processor family of your CPU here.

- SMT (Hyperthreading) scheduler support

  This should be selected if you have a modern Pentium chip with hyperthreading support. It is not mandatory though (the kernel will run fine without it) but might improve scheduling decisions made by the kernel.

- HPET Timer Support

  This enables support for the High Precision Event Timer, which can be seen as a time-source resource on somewhat more modern systems. Especially if you have more than 1 core/CPU, enabling this offers "cheaper" time access than without HPET Timer support.

- Multi-core scheduler support

  Enable this if you have a CPU with multiple cores inside; it will improve the CPU scheduler performance.

- Preemption Model (Preemptible Kernel (Low-Latency Desktop))

  Preemption means that a priority process, even when currently in kernel mode executing a system call, can yield his CPU time to another process. The user will notice this as if his system is running somewhat more 'smoothly' as applications might react faster to user input.

  There are three models available:

  - No Forced Preemption, or
  - Voluntary Kernel Preemption, where low-priority processes can voluntarily yield CPU time, or
  - Preemptible Kernel, where all processes can yield CPU time (as long as they're not in a critical kernel region at that moment)

- Machine Check / overheating reporting

  MCE allows the processor to notify the kernel when problems are detected (like overheating); based on its severity, the Linux kernel can report the issue or take immediate action.

- Intel MCE features

  This is part of the "Machine Check / overheating reporting" section, and enables Intel-specific MCE features. I enable this, as I have an Intel-based system.

- Memory Model (Sparse Memory)

  If you have a 32-bit processor, selecting Flat Memory is what you need. CPUs with a larger address space support (like 64-bit CPUs) most likely only allow you to select "Sparse Memory" as you are not likely to have more than a few thousand terabytes of RAM ;-) When "Sparse Memory" is selected, "Sparse Memory virtual memmap" should be selected as well.

- MTRR (Memory Type Range Register) support

  With MTRR support, applications such as the X server can control how the processor caches memory accesses, boosting performance for reads/writes to certain memory ranges.

- Enable seccomp to safely compute untrusted bytecode

  As recommended by its help description, we enable this in case an application might want to use it. It has no impact if no such applications exist on your system, and if they do, you most likely want the added security measures this provides.

Power Management and ACPI Options

The power management options provide power-saving features for Linux, not only the APM / ACPI support, but also suspend-to-ram and standby support.

```
[*] Power Management support
[ ]   Power Management Debug Support
[*] Suspend to RAM and standby
[*] Hibernation (aka 'suspend to disk')
(/dev/sda5) Default resume partition
[ ] Run-time PM core functionality
[*] ACPI (Advanced Configuration and Power Interface) Support --->
[ ] SFI (Simple Firmware Interface) Support --->
    CPU Frequency Scaling --->
-*- CPU idle PM support
    Memory power savings --->
```

The following options are of particular interest:

- Power Management Support

  Enable this to be able to select one or more of the other power management options.

- Suspend to RAM and standby

  If you will have moments where you temporarily leave your system but don't want to shut it down and boot it back later, you can opt to have the system suspend itself into memory - in this case, many powerconsuming devices are shut down but you don't lose any information as everything remains in memory (and memory remains powered up).

- Hibernation (aka 'suspend to disk')

  In hibernation, all devices shut down. The current state of your system (such as your memory content) is saved into your swap space. When you boot your system back, the Linux kernel will detect this in the swap space and load all information back into memory so you can continue where you left off.

  With suspend to disk enabled, set the default resume partition to your swap partition.

- ACPI (Advanced Configuration and Power Interface) Support

  Within this section you can configure several aspects of the ACPI support. Enabling ACPI can be of great help to reduce power consumption as it is a powerful technology. Sadly, not every device follows the ACPI guidelines strictly. You will find topics on the internet where boot failures or network irregularities can be solved by disabling a part of the ACPI support inside Linux.

  ```
  --- ACPI (Advanced Configuration and Power Interface Support)
  [*]   Deprecated /proc/acpi files
  [*]   Deprecated power /proc/acpi directories
  < > ACPI 4.0 power meter
  [*]   Future power /sys interface
  [*]   Deprecated /proc/acpi/event support
  <*> AC Adapter
  <*> Battery
  ```

```
<*> Button
<*> Video
<*> Fan
<*> Processor
<*>   Thermal Zone
[ ] Debug Statements
< > PCI slot detection driver
< > Smart Battery System
```

Within the ACPI configuration you should select the components for which you want support. On regular desktops, you most likely don't have a battery so support for that (and AC Adapters) won't be necessary.

I select a few "deprecated" settings as I know the reporting tools I use (for battery status etc.) still rely on these files, directories and events to function correctly.

- CPU Frequency Scaling

  If you own a laptop you'll most likely want to enable CPU Frequency scaling as it will slow down the CPU speed (and the power consumption with it) when the CPU isn't used.

```
[*] CPU Frequency scaling
[ ]   Enable CPUfreq debugging
<*>   CPU frequency translation statistics
[ ]      CPU frequency translation statistics details
      Default CPUFreq governor (performance) --->
-*- 'performance' governor
< > 'powersave' governor
< > 'userspace' governor for userspace frequency scaling
< > 'ondemand' cpufreq policy governor
< > 'conservative' cpufreq governor
      *** CPUFreq processor drivers ***
< >   Processor Clocking Control interface driver
<*>   ACPI Processor P-States driver
< >   AMD Opteron/Athlon64 PowerNow!
< >   Intel Enhanced SpeedStep (deprecated)
< >   Intel Pentium 4 clock modulation
```

  In the above, only the "performance" governor is selected, as the laptop will always be used as a workstation. However, you definitely want to enable additional governors for other purposes as well. A governor can be seen as a policy when and how the CPU frequency needs to be changed.

Bus options (PCI etc.)

A bus is a physical connection between several devices. The most popular bus technology within a computer nowadays is PCI (or PCI Express) but a few other bus technologies exist (for instance PCMCIA).

```
[*] PCI Support
[*]   Support mmconfig PCI config space access
      PCI access mode (Any) --->
[*] PCI Express support
[ ]   Root Port Advanced Error Reporting support
[ ] Message Signaled Interrupts (MSI and MSI-X)
[ ] PCI Debugging
< > PCI Stub driver
[ ] Interrupts on hypertransport devices
[ ] PCI IOV support
< > PCCard (PCMCIA/CardBus) support --->
```

```
< > Support for PCI Hotplug --->
```

In the above example I only selected PCI, mmconfig PCI config space access and PCI-X support; laptop users will most likely enable PCCard support as well. Within the submenu of the PCCard configuration you will be asked to select the supporting bridge. A bridge is a component that links one bus technology with another. A PCMCIA bridge allows PCMCIA devices to connect to your system. Most systems with PCMCIA support have a CardBus yenta-compatible bridge.

Although I own a laptop, I have no PC cards of any kind nor do I suspect I will need them quickly, so I leave out support for that.

Executable File Formats / Emulations

Within this section you can select what binaries (format for executable files with machine instructions inside) Linux should support.

```
[*] Kernel support for ELF binaries
[ ] Write ELF core dumps with partial segments
< > Kernel support for MISC binaries
[*] IA32 Emulation
< > IA32 a.out support
```

The binary format used by Linux is ELF. Very old Linux systems and a couple of BSD operating systems use a.out binaries but it isn't necessary to include support for those any more. If you are configuring for a 64-bit system, definitely enable IA32 Emulation. You'll need it. Trust me.

Networking

Inside the networking configuration tab you configure the various aspects related to your network.

```
[*] Networking support
      Networking options --->
[ ]   Amateur Radio support --->
< >   CAN bus subsystem support --->
< >   IrDA (infrared) subsystem support --->
< >   Bluetooth subsystem support --->
-*-   Wireless --->
< >   WiMAX Wireless Broadband support --->
< >   RF switch subsystem support --->
```

Within the 'Networking options', you will need to enable support for the networking technologies (not hardware) you want to support.

```
----- Networking Options -----
<*> Packet socket
<*> Unix domain sockets
< > PF_KEY sockets
[*] TCP/IP networking
[ ]   IP: multicasting
      ...
[ ] Security Marking
[*] Network packet filtering framework (Netfilter) --->
< > Asynchronous Transfer Mode (ATM)
< > 802.1d Ethernet Bridging
[ ] Distributed Switch Architecture support --->
< > 802.1Q VLAN Support
< > DECnet Support
< > ANSI/IEEE 802.2 LLC type 2 Support
< > The IPX protocol
```

```
< > Appletalk protocol support
< > Phonet protocols family
[ ] QoS and/or fair queuing --->
[ ] Data Center Bridging support
    Network testing --->
```

The most notable options here are:

- Packet socket

  This allows programs to interface with the network devices immediately (without going through the network protocol implementation on the Linux kernel). It is required by tools such as tcpdump / wireshark (popular network analysing tools). You don't need to enable this, but I often perform network analysis myself so I need to have this enabled.

- Unix domain sockets

  Sockets are a standard mechanism in Unix for processes to communicate with each other. This is an important setting that you must leave on.

- TCP/IP networking

  Although you don't have to select any of the subfeatures that are shown when you enable this, TCP/IP networking support is definitely a must-have.

- Network packet filtering framework (Netfilter)

  Enable this if you are planning on configuring a firewall on your system or have your system act as a gateway for others. Enable the 'IP tables support' found under 'IP: Netfilter Configuration' and select:

  ```
  <*> IPv4 connection tracking support (required for NAT)
  [*]   proc/sysctl compatibility with old connection tracking
  <*> IP tables support (required for filtering/masq/NAT)
  <*>   Packet filtering
  <*>      REJET target support
  <*>   LOG target support
  < >   ULOG target support
  <*>   Full NAT
  <*>     MASQUERADE target support
  <*>   Packet mangling
  ```

Users of a wireless network card will, under 'Networking', also select the Wireless configuration.

```
--- Wireless
<*>  cfg80211 - wireless configuration API
[ ]     nl80211 testmode command
[ ]     enable developer warnings
[ ]     cfg80211 regulatory debugging
[*]     enable powersave by default
[ ]     cfg80211 DebugFS entries
[ ]     cfg80211 wireless extensions compatibility
[*] Wireless extensions sysfs files
-+-  Common routines for IEEE802.11 drivers
[ ]   lib80211 debugging messages
< >  Generic IEEE 802.11 Networking Stack (mac80211)
```

I've selected these options because IEEE 802.11 is the standard for wireless networking:

- cfg80211 - wireless configuration API

You need to enable this if you have a wireless card

- enable powersave by default

  Enables powersaving features of the wireless cards - definitely a must-have if you have wireless on a laptop as this reduces power consumption dramatically.

Device Drivers

Within this section you can configure support for the various devices in your system. It is with this configuration that the output of the **lspci** command (and other system information) is needed. The next example is merely that - an example. As this is very specific to your system, it is not possible to provide a general example that suits everybody. For completeness sake, I'll give the configuration for my own system with the motivation of the selection of each item.

```
    Generic Driver Options --->
< > Connector - unified userspace <-> kernelspace linker --->
< > Memory Technology Device (MTD) support --->
< > Parallel port support --->
-*- Plug and Play support --->
[*] Block devices --->
[ ] Misc devices --->
< > ATA/ATAPI/MFM/RLL support (DEPREACATED) --->
    SCSI device support --->
<*> Serial ATA and Parallel ATA drivers --->
[ ] Multiple devices driver support (RAID and LVM) --->
[ ] Fusion MPT device support --->
    IEEE 1394 (FireWire) support --->
< > I2O device support --->
[ ] Macintosh device drivers --->
[*] Network device support --->
[ ] ISDN support --->
< > Telephony support --->
    Input device support --->
    Character devices --->
{*} I2C support --->
[ ] SPI support --->
    PPS support --->
[ ] GPIO support --->
< > Dallas's 1-wire support --->
-*- Power supply class support --->
< > Hardware Monitoring support --->
-*- Generic Thermal sysfs driver --->
[ ] Watchdog Timer Support --->
    Sonics Silicon Backplane --->
    Multifunction device drivers --->
[ ] Voltage and Current Regulator Support --->
< > Multimedia support --->
    Graphics support --->
<*> Sound card support --->
[*] HID Devices --->
[*] USB support --->
<M> MMC/SD/SDIO card support --->
[ ] LED Support --->
[ ] Accessibility support --->
< > InfiniBand support --->
[ ] EDAC (Error Detection And Correction) reporting --->
< > Real Time Clock --->
```

```
[ ] DMA Engine support --->
[ ] Auxiliary Display support --->
< > Userspace I/O drivers --->
    TI VLYNQ --->
[ ] Staging drivers --->
[*] X86 Platform Specific Device Drivers --->
```

As you can see, most of the options are not selected and almost all of them provide subsections. This is of course expected, as device driver support is a huge part of the Linux source code, and the end user will only select a very small part of it.

Block devices

Block devices are devices where you can access data in blocks (as opposed to characters). Not all block devices are configurable through this menu (a well-known block device type, hard disk, is configured elsewhere) as you notice from the available configuration options:

```
--- Block devices
< >   Normal floppy disk support
< >   Compaq SMART2 support
< >   Compaq SMart Array 5xxx support
< >   Mylex DAC960/DAC1100 PCI RAID Controller support
<*>   Loopback device support
< >     Cryptoloop Support
< >   DRBD Distributed Replicated Block Device support
< >   Network block device support
< >   Promise SATA SX8 support
< >   Low Performance USB Block driver
< >   RAM block device support
< >   Packet writing on CD/DVD media
< >   ATA over Ethernet support
[ ]   Very old hard disk (MFM/RLL/IDE) driver
```

- Loopback device support

  The only block device I enable is loopback support. This allows me to mount images (files) just like they were devices.

SCSI device support

Although my system doesn't have SCSI, it has Serial ATA (SATA) with disks attached to it. SATA support in Linux is brought through the SCSI subsystem, so I need to configure SCSI device support.

```
< > RAID Transport Class
-*- SCSI device support
< > SCSI target support
[ ] legacy /proc/scsi/ support
    *** SCSI support type (disk, tape, CD-ROM) ***
<*> SCSI disk support
< > SCSI tape support
< > SCSI OnStream SC-x0 tape support
< > SCSI CDROM support
< > SCSI generic support
< > SCSI media changer support
    *** Some SCSI devices (e.g. CD jukebox) support multiple LUNs ***
[ ] Probe all LUNs on each SCSI device
[ ] Verbose SCSI error reporting (kernel size +=12K)
[ ] SCSI logging facility
[ ] Asynchronous SCSI scanning
    SCSI Transports --->
```

```
[ ] SCSI low-level drivers --->
< > SCSI Device Handlers --->
< > OSD-Initiator library
```

- SCSI disk support

  SCSI disk support is needed for the SATA disks.

Serial ATA and Parallel ATA drivers

Serial ATA support is needed to be able to access my disks.

```
--- Serial ATA and Parallel ATA drivers
[ ]   Verbose ATA error reporting
[*]   ATA ACPI Support
[ ]   SATA Port Multiplier support
<*>   AHCI SATA support
< >   Silicon Image 3124/3132 SATA support
[*]   ATA SFF support
< >     ServerWorks Frodo / Apple K2 SATA support
<*>     Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support
< >     Marvell SATA support
      ...
```

- ATA ACPI Support

  Enables retrieving ACPI related files (performance, security, power management ...) from the ACPI BIOS and save them on the disk controller.

- Intel ESB, ICH, PIIX3, PIIX4 PATA/SATA support

  The only selection made in this configuration is the support for my SATA chip set of which lspci told me it was an Intel ICH6 chip set:

  ```
  # lspci | grep SATA
  00:1f.2 SATA controller: Intel Corporation 82801FBM (ICH6M)
          SATA Controller (rev 04)
  ```

  All the other options are drivers for other chip sets.

Network device support

Inside network device support we configure the drivers for the networkcards.

```
--- Network device support
<*>   Dummy net driver support
< >   Bonding driver support
< >   EQL (serial line load balancing) support
< >   Universal TUN/TAP device driver support
< >   Virtual ethernet pair device
< >   General Instruments Surfboard 1000
< >   ARCnet support --->
[ ]   Ethernet (10 or 100Mbit) --->
[*]   Ethernet (1000 Mbit) --->
[ ]   Ethernet (10000 Mbit) --->
[ ]   Token Ring driver support --->
[*]   Wireless LAN --->
      USB Network Adapters --->
[ ]   Wan interfaces support --->
< >   FDDI driver support
< >   PPP (point-to-point protocol) support
```

```
< >   SLIP (serial line) support
[ ]   Fibre Channel driver support
< >   VMWare VMXNET3 ethernet driver
```

- Dummy net driver support

  This driver allows me to create an interface which takes on all packets and just ignores them. This seems to be a weird driver, but it can come in handy from time to time. Also, this has no impact on my kernel size so I don't mind enabling this for the few times I actually use it.

- Ethernet (1000 Mbit)

  I have a Realtek 8169 ethernet card (which is a 1Gbit network card) as mentioned by **lspci**:

  ```
  # lspci | grep Ethernet
  06:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd.
          RTL-8169 Gigabit Ethernet (rev 10)
  ```

  As such I select the 'Realtek 8169 gigabit ethernet support' option in the configuration screen.

- Wireless LAN

  As my system is a laptop with onboard wireless network card, I need to enable WLAN support as well.

  ```
  --- Wireless LAN
  < >   Cisco/Airnet 34X/35X/4500/4800 ISA and PCI cards
  ...
  <M>   Intel PRO/Wireless 2200BG and 2915ABG Network Connection
  [*]     Enable promiscuous mode
  -*-       Enable radiotap format 802.11 raw packet support
  [*]       Enable creation of a RF radiotap promiscuous interface
  [ ]     Enable QoS support
  ...
  ```

  - Wireless LAN (IEEE 802.11)

    The network card I have is an 802.11-something so I need to enable this.

  - Intel PRO/Wireless 2200BG and 2915ABG Network Connection

    lspci says that my wireless card is an Intel PRO/Wireless 2200BG one, so I need to enable support for it:

    ```
    # lspci | grep Wireless
    06:04.0 Network controller: Intel Corporation PRO/Wireless
            2200BG Network Connection (rev 05)
    ```

  - Enable promiscuous mode

    I need promiscuous mode when I want to analyse the wireless network I work on.

Input device support

Input devices are the devices you know to interact with your system, such as a keyboard and a mouse.

```
-*- Generic input layer (needed for keyboard, mouse, ...)
< >   Support for memoryless force-feedback devices
< >   Polled input device skeleton
< >   Sparse keymap support library
      *** Userland interfaces ***
```

```
-*-   Mouse interface
[*]     Provide legacy /dev/psaux device
(1024)  Horizontal screen resolution
(768)   Vertical screen resolution
< >   Joystick interface
<*>   Event interface
< >   Event debugging
      ** Input Device Drivers ***
-*-   Keyboards --->
[*]   Mice --->
[ ]   Joysticks/Gamepads --->
[ ]   Tables --->
[ ]   Touchscreens --->
[ ]   Miscellaneous devices --->
    Hardware I/O ports --->
```

- Generic input layer (needed for keyboard, mouse, ...)

  As the title says already, I need this for keyboard/mouse support

- Mouse interface

  Enable mouse support

- Horizontal screen resolution / Vertical screen resolution

  Actually, this setting is ignored as it is only really used if your pointing device is a digitizer or tablet rather than a simple mouse.

- Event interface

  This enables evdev support, which is somewhat mandatory if you want to work with graphical interfaces (for instance, the xorg configuration requires this).

- Keyboards

  Keyboard support is automatically selected, but in the subconfiguration you don't need to select anything unless you have a very special keyboard.

- Mice

  Within the mouse configuration, I enable 'PS/2 mouse' as my mouse is a PS/2 one.

- Hardware I/O ports

  Inside this configuration section, 'Serial I/O support' should be automatically selected as it is used by the keyboard/mice support.

Character devices

Character devices are devices that are accessed character per character. An example of a character device is a terminal.

```
-*- Virtual terminal
[ ]   Support for binding and unbinding console drivers
[ ] /dev/kmem virtual device support
[ ] Non-standard serial port support
< > HSDPA Broadband Wireless Data Card - Globe Trotter
    Serial drivers --->
-*- Unix98 PTY support
[ ]   Support multiple instances of devpts
```

```
[*] Legacy (BSD) PTY support
(256) Maximum number of legacy PTY in use
< > IPMI top-level message handler --->
<*> Hardware Random Number Generator Core support
< >   Timer IOMEM HW Random Number Generator support
<*>   Intel HW Random Number Generator support
< >   AMD HW Random Number Generator support
< >   AMD Geode HW Random Number Generator support
< >   VIA HW Random Number Generator support
< > /dev/nvram support
< > Enhanced Real Time Clock Support (Legacy PC RTC driver)
< > Generic /dev/rtc emulation
< > Siemens R3964 line discipline
< > Applicom intelligent fieldbus card support
< > ACP Modem (Mwave) support
< > NatSemi PC8736x GPIO Support
< > NatSemi Base GPIO Support
< > AMD CS5535/CS5536 GPIO (Geode Companion Device)
< > RAW driver (/dev/raw/rawN)
[ ] HPET - High Precision Event Timer
< > Hangcheck timer
< > TPM Hardware Support --->
< > Telecom clock driver for ATCA SBC
```

- Virtual terminal

  Support for virtual terminals is automatically selected. You need it as you'll work with virtual consoles all the time in Linux: if you're opening a terminal window, you're working in a virtual console.

- Unix98 PTY Support

  This should be automatically selected; it is the support for virtual PTYs which you definitely need.

- Legacy (BSD) PTY support

  Enables support for virtual PTYs, but then a different kind. Although not selecting this option won't break your kernel, you'll most likely get a few (cosmetic) errors every time you open a terminal. So better enable this.

- Hardware Random Number Generator Core support

  To have support for the hardware random number generator, select this and the specific generator in the next list.

- Intel HW Random Number Generator support

  My generator provider is an Intel one (as my CPU is an Intel CPU).

Graphics support

Graphical card support is configured here as well as framebuffer support (allowing applications to access the graphics hardware through a well-defined interface).

```
<*> /dev/agpgart (AGP Support) --->
-*- VGA Arbitration
(2)   Maximum number of GPUs
[ ] Laptop Hybrid Graphics - GPU switching support
<M> Direct Rendering Manager (XFree86 4.1.0 and higher DRI support) --->
{M} Lowlevel video output switch controls
```

```
{*} Support for frame buffer devices --->
< > CyberPro 2000/2010/5000 support
< > Arc Monochrome LCD board support
[ ] Asiliant (Chips) 6900 display support
[ ] IMS Twin Turbo display support
< > VGA 16-color graphics support
[*] VESA VGA graphics support
< > N411 Apollo/Hecuba devkit support
< > Hercules mono graphics support
...
[ ] Backlight & LCD device support --->
    Display device support --->
    Console display driver support --->
[ ] Bootup logo --->
```

- /dev/agpgart (AGP Support)

  I know my laptop has an on-board AGP card. lspci tells me what card it is:

  ```
  # lspci | grep VGA
  00:02.0 VGA compatible controller: Intel Corporation Mobile
          915GM/GMS/910GML Express Graphics Controller (rev 03)
  ```

  As such, I also enable 'Intel 440LX/BX/GX, I8xx and E7x05 chip set support'. You might believe that I am in error because the title doesn't mention 915GM (as shown in lspci's output) but if I read the help for the option, I read that I915 is supported through this driver as well.

- Direct Rendering Manager (XFree86 4.1.0 and higer DRI support)

  DRM is needed by XFree86 to improve graphic performance (including 3D support). Within the subconfiguration, I enable drivers for my Intel graphical card:

  ```
  --- Direct Rendering Manager (XFree86 4.1.0 and higher DRI support)
  < >   3dfx Banshee/Voodoo3+
        ...
  <M>   Intel 830M, 845G, 852GM, 855GM, 865G
  < >     i830 driver
  <M>     i915 driver
  < >   Matrox g200/g400
        ...
  ```

- Support for frame buffer devices

  I want frame buffer support because that allows me to display more characters than just 80x25 when working in the command-line mode (console). In the subconfiguration, I enable 'VESA VGA graphics support' which enables standard VESA support for framebuffer access.

- Console display driver support

  Within the console display driver support, I enable framebuffer support for the console:

  ```
  -*- VGA test console
  [ ]   Enable Scrollback Buffer in System RAM
  {*} Framebuffer Console support
  [ ]   Map the console to the primary display device
  [ ]   Framebuffer Console Rotation
  [ ] Select compiled-in fonts
  ```

Sound

To support my sound card, I enable sound card support and the sound system I want to use.

```
<*> Sound card support
[ ]   Preclaim OSS device numbers
<*>   Advanced Linux Sound Architecture --->
< >   Open Sound System (DEPRECATED) --->
```

ALSA (Advanced Linux Sound Architecture) is the latest sound system supported by Linux. OSS is deprecated and ALSA provides OSS compatibility for those applications that still require OSS.

```
<*> Advanced Linux Sound Architecture
<*>   Sequencer support
< >     Sequencer dummy client
< >   OSS Mixer API
< >   OSS PCM (digital audio) API
[ ]   OSS Sequencer API
[ ]   Dynamic device file minor numbers
[ ]   Support old ALSA API
[ ]   Verbose procfs contents
[ ]   Verbose printk
[ ]   Debug
[ ]   Generic sound devices --->
[*]   PCI sound devices --->
[ ]   USB devices --->
< >   ALSA for SoC audio support --->
```

- PCI sound devices

  Under the PCI devices, select the audio card you have. Again, lspci can show you what device you have:

  ```
  # lspci | grep Audio
  00:1e.2 Multimedia audio controller: Intel Corporation
          82801FB/FBM/FR/FW/FRW (ICH6 Family) AC'97 Audio
          Controller (rev 04)
  ```

  With this information, I know that I need to select 'Intel/SiS/nVidia/AMD/ALi AC97 Controller'.

HID Devices

A HID device (Human Interface Device) is a device that takes input from the user. An important class of devices that use the HID interface are USB keyboards and mice.

```
--- HID Devices
-*-   Generic HID support
[ ]     /dev/hidraw raw HID device support
      *** USB Input Devices ***
<*>   USB Human Interface Device (full HID) support
[ ]   PID device support
[ ]   /dev/hiddev raw HID device support
      Special HID drivers --->
```

- USB Human Interface Device (full HID) support

  I select this as I often use a USB mouse on my laptop.

USB support

USB devices come in various types and classes; the USB support screen is therefore quite a large device driver configuration screen with many options.

```
--- USB support
<*> Support for Host-side USB
```

```
[ ]   USB verbose debug messages
[ ]   USB announce new devices
      *** Miscellaneous USB options ***
[ ]   USB device filesystem (DEPRECATED)
[ ]   USB device class-devices (DEPRECATED)
...
      *** USB Host Controller Drivers ***
< >   Cypress C67x00 HCD support
<*>   EHCI HCD (USB 2.0) support
< >   ISP116X HCD support
<*>   OHCI HCD support
<*>   UHCI HCD (most Intel and VIA) support
< >   SL811HS HCD support
< >   R8A66597 HCD support
      *** USB Device Class drivers ***
< >   USB Modem (CDC ACM) support
<*>   USB Printer support
    *** NOTE: USB_STORAGE enables SCSI, and 'SCSI disk support'
    *** may also be needed; see USB_STORAGE Help for more information
<*> USB Mass Storage support
< >   USB Mass Storage verbose debug
      ...
[ ] The shared table of common (or usual) storage devices
    *** USB Imaging devices ***
< > Microtek X6USB scanner support
[ ] USB Monitor
    *** USB port drivers ***
< > USB Serial Converter support --->
    *** USB Miscellaneous drivers ***
< > EMI 6|2m USB Audio interface support
    ...
```

- Support for Host-side USB

  This enables general USB support (technology-wise)

- USB device filesystem

  With this enabled, the Linux kernel will create information files inside /proc/bus/usb about each device. This information can come in handy to debug USB device support but is also used by tools to provide more information about a USB device.

- EHCI HCD (USB 2.0) support

  There are a few standards for USB controllers. For USB 2.0 support, you need to enable EHCI HCD support.

- UHCI HCD (most Intel and VIA) support

  UHCI is Intels' interface for USB 1.0 and 1.1 support.

- USB Printer support

  As I do want to print occasionally, I need to enable USB printer support (as my printer is a USB printer).

- USB Mass Storage support

  USB Mass Storage support is needed to be able to access USB sticks, USB disks and other USB media that I want to be able to access as a remote disk. This includes most digital cameras.

MMC/SD card support

My laptop supports MMC/SD cards so I want to support this in my kernel as well.

```
[ ] MMC debugging
[ ] Assume MMC/SD cards are non-removable (DANGEROUS)
    *** MMC/SD Card Drivers ***
<M> MMC block device driver
[*]   Use bounce buffer for simple hosts
< > SDIO UART/GPS class support
< > MMC host test driver
    *** MMC/SD HOst Controller Drivers ***
<M> Secure Digital Host Controller Interface support
<M> SDHCI support on PCI bus
< > Winbond W83L51xD SD/MMC Card Interface
< > ENE CB710 MMC/SD Interface support
< > VIA SD/MMC Card Reader Driver
```

- MMC block device driver

  This driver enables the Linux kernel to mount an MMC/SD card as a file system.

- Use bounce buffer for simple hosts

  The Linux kernel help system has informed me that this helps performance on certain controllers. Although I don't know if I really need this, I've enabled this as I was unsure, and the help says I need to say 'Y' if I am unsure.

- Secure Digital Host Controller Interface support

  Inside the help of this option it says that this enables SD controller support for controllers manufactured by Texas Instruments, Ricoh and Toshiba. lspci informs me that I have a Texas Instruments device, so I enable this:

  ```
  # lspci | grep SD
  06:09.4 SD Host controller: Texas Instruments
          PCI6411/6421/6611/6621/7411/7421/7611/7621 Secure
          Digital Controller
  ```

File Systems

Our next stop is file system support. A file system is the formatting structure used on a (disk)partition as we mentioned before. It is important here that you build support for the file systems in the kernel (and not as a module) as you could otherwise end up with a kernel that needs file system support to mount the file system... where your kernel modules are installed.

```
<*> Second extended fs support
[ ]   Ext2 extended attributes
[ ]   Ext2 execute in place support
<*> Ext3 journalling file system support
[ ]   Default to 'data=ordered' in ext3
[ ]   Ext3 extended attributes
< > The Extended 4 (ext4) filesystem
[ ] JDB (ext3) debugging support
< > Reiserfs support
< > JFS filesystem support
< > XFS filesystem support
< > OCFS2 file system support
[*] Dnotify support
[*] Inotify file change notification support
```

```
[*]   Inotify support for userspace
[ ] Quota support
< > Kernel automounter support
< > Kernel automounter version 4 support (also supports v3)
< > FUSE (Filesystem in Userspace) support
    Caches --->
    CD-ROM/DVD Filesystems --->
    DOS/FAT/NT Filesystems --->
    Pseudo filesystems --->
[ ] Miscellaneous filesystems --->
[*] Network File Systems --->
    Partition Types --->
-*- Native language support --->
< > Distributed Lock Manager (DLM) --->
```

- Second extended fs support

  My /boot partition uses the ext2 file system...

- Ext3 journalling file system support

  ... and all other partitions I have use the ext3 file system

- Dnotify support

  Some applications might need Dnotify support (a notification system where the kernel sends a signal to a userspace application to notify it about file changes).

- 'Inotify file change notification support' and 'Inotify support for userspace'

  Inotify is a better implementation of a file notification system than Dnotify and is used by various applications.

- CD-ROM/DVD Filesystems

  Within this subsection, enable 'ISO 9660 CDROM file system support' as well as 'Microsoft Joliet CDROM extensions' (to support the larger file name scheme used by Microsoft).

- DOS/FAT/NT FIlesystems

  If you are never going to work on FAT/NTFS file systems, you don't need this, but I occasionally attach a FAT/NTFS formatted disk on my laptop to help people.

  ```
  <*> MSDOS fs support
  <*> VFAT (Windows-95) fs support
  (437) Default codepage for FAT
  (iso8859-15) Default iocharset for FAT
  <*> NTFS file system support
  [ ]   NTFS debugging support
  [ ]   NTFS write support
  ```

  You notice that I don't enable NTFS write support. This is because the in-kernel NTFS write support is very limited (thank you Microsoft for hiding how NTFS works).

- Pseudo filesystems

  Pseudo file systems are virtual file systems where the kernel maps information onto virtual files which you can read or write.

  ```
  -*- /proc file system support
  [*]   /proc/kcore support
  ```

```
[*] Virtual memory file system support (former shm fs)
[ ]   Tmpfs POSIX Access Control Lists
[ ] HugeTLB file system support
< > Userspace-driven configuration filesystem
```

Apart from the /proc file system support, I also enable 'Virtual memory file system support' (also known as tmpfs) which allows you to map a portion of your virtual memory as a file system (every file you create inside a tmpfs file system is stored in memory or swap space; when unmounted, the files are lost).

Tmpfs is often used for the /tmp location.

- Network File Systems

  Network file systems allow you to access files on remote sites as if they were local (rather than using tools / technologies like FTP to store or retrieve them).

```
<*> NFS file system support
[*]   Provide NFSv3 client support
[ ]     Provide client support for the NFSv3 ACL protocol extension
[ ]   Allow direct I/O on NFS files
<*> NFS server support
[ ]   Provide NFSv3 server support
[*]   Provide NFS server over TCP support
< > SMB file system support (OBSOLETE, please use CIFS)
<*> CIFS support (advanced network filesystem, SMBFS successor)
[ ]   CIFS statistics
[ ]   Support legacy servers which use weaker LANMAN security
[ ]   CIFS extended attributes
[ ]   Enable additional CIFS debugging routines
< > NCP file system support (to mount NetWare volumes)
< > Code file system support (advanced network fs)
```

  - NFS file system support

    I use NFS to share Gentoo's portage tree with other systems (and even virtual machines) so I need to enable NFS support

  - Provide NFSv3 client support

    With this I can act as an NFS client (to mount remote NFS shares)

  - NFS server support

    With this I can act as an NFS server (to provide remote NFS shares)

  - Provide NFS server over TCP support

    Enable NFS servers with TCP support; the Linux kernel help system tells me that this is interesting when the network is lossy or congested. As I'm using a wireless network, this seems to be a good choice.

  - CIFS support

    CIFS enables support for mounting SAMBA shares as well as Windows shares (and authenticate my own system on a Windows network).

Cryptographic API

Some kernel subsystems require in-kernel cryptographic algorithm support. The algorithms that are needed will be automatically selected, so you don't have to configure anything here.

Building a Linux Kernel

Once your kernel configuration is finished, save the configuration (select 'Exit' and confirm that you want to save the new kernel configuration). Next, build the main kernel and the modules by entering the **make** command.

```
$ make
```

The make command reads in a specific configuration file (the Makefile) which contains the instructions for building the Linux kernel completely. This can take a while; when finished, the kernel image will be stored in (for x86 architecture) `arch/i386/boot` and is called `bzImage`.

By default, the make command will launch the build instructions one at a time. However, most systems have multiple cores (or even CPUs) at their disposal, so it makes sense to - when possible - run multiple instructions next to each other. This can be accomplished using the **-j#** command (where '#' is the number of parallel build instructions).

A common value is the number of cores available on the system, so for a 4-core single-processor system:

```
$ make -j4
```

Next, you need to install the kernel modules (if any) on your system. Type in **make modules_install** to have the **make** command automatically copy the kernel modules to the correct location (`/lib/modules/`*`kernelversion`*):

```
# make modules_install
```

Finally, copy over the kernel image to the /boot location. If /boot is a separate partition (as it is on my system) which isn't automatically mounted, mount it first:

```
# mount /boot
# cp /usr/src/linux/arch/x86/boot/bzImage /boot/kernel-3.10.7
```

You can also use **make install** right after (or before) the **make modules_install** step. This will copy the kernel to `/boot`, calling it `/boot/vmlinuz`, copying the previous kernel to `/boot/vmlinuz.old`. If you rather have it not overwrite kernels, you can also first create a symbolic link `/boot/vmlinuz` pointing to the correct kernel (say `/boot/vmlinuz-3.8.5`). A make install will then save the new kernel (as a new file) and update the link instead, keeping the old kernel image available for future use. I actually recommend to either manually copy the kernel, or use the symbolic link approach.

To safely support make install, do the following steps only once:

```
# cp /usr/src/linux/arch/x86/boot/bzImage /boot/vmlinuz-3.10.7
# cd /boot
# ln -s vmlinuz-3.10.7-r2 vmlinuz
```

Rebuilding a Kernel

Suppose that you have successfully built a kernel and are happily working with it. At some point in time, you'll need to upgrade your kernel. Luckily, there is no reason to perform the above steps over and over again.

Go to the source location of your new kernel sources (most likely, still `/usr/src/linux`) and get your current kernel configuration. If you've followed the above configuration directives, you'll have access to these settings in `/proc/config.gz`:

```
# cd /usr/src/linux
# zcat /proc/config.gz > .config
```

The **zcat** command will "load" your current kernel configuration into the `.config` file, which is the file used by the kernel building process to find the (new) configuration settings. Next, tell the kernel build tool to use this configuration as if it is your old(er) configuration. This will result in the build process to only ask for validation when new configuration items are found (in the new sources) that weren't available previously.

```
# make oldconfig
```

When this is finished, continue with the **make** (or **make -j#** with # the number of parallel build instructions allowed) and **make modules_install** commands, finishing off with the **mount** and copy commands to install the new kernel at the `/boot` location.

Initial ram file systems

I have touched the concept of initrd and initramfs already in the earlier sections. Its purpose is clear: to offer the booting system the necessary tools, files, libraries and modules so that the boot process can be continued on a fully prepared (and all hardware properly detected) system.

Although initrd might still be in use, most distributions nowadays suggest the use of initramfs. Whereas an initrd is a full device (for which you pass on an image), an initramfs is an archive that is extracted on an *tmpfs* file system. This offers more flexibility and less resource consumption within the kernel.

To generate an initial ram file system, I recommend that you use the **genkernel** application (even if you configured and build your kernel manually). Its usage is quite simple:

```
# genkernel --install --lvm initramfs
```

When **genkernel** has built its initial ram file system archive, you will find it in the `/boot` location, cleverly named starting with `initramfs-`.

Configuring the Boot Loader

Before you can use the new kernel, you need to edit your boot loader configuration to use the new kernel. The bootloader I recommend is GRUB, the GRand Unified Bootloader, but others exist (especially for non-x86 compatible architectures which might need different boot loaders). In this section, I'll focus on GRUB2 (the previous GRUB, nowadays called GRUB Legacy, is no longer maintained).

Installing GRUB

Before you can start using GRUB, you need to install GRUB onto your MBR, or Master Boot Record. That is a special location on your disk (the first 512 bytes of a disk) which is loaded by the BIOS and executed (if it contains executable material).

By installing GRUB into the MBR, GRUB is automatically started after the BIOS POST processing (Power On Self Test).

First of all, you need to install GRUB:

```
# emerge grub
```

Next, we install GRUB onto the MBR:

```
# grub-install /dev/sda
```

If the disk is not configured with sufficient space after the MBR but before the first partition, you will need to force the installation. GRUB will use a special support mode in that case:

```
# grub-install -f /dev/sda
```

Configuring GRUB

Unlike GRUB Legacy, GRUB is configured through variables in `/etc/default/grub`, which are used to auto-generate the configuration file.

```
# nano -w /etc/default/grub
```

The variable that, if you need to set anything, is the most likely candidate, is `GRUB_CMDLINE_LINUX`. This variable content is added to the kernel line and is used to pass on additional kernel parameters. For instance, to pass on the parameter "dolvm" (to support a logical volume as root file system, needed for some init scripts):

```
GRUB_CMDLINE_LINUX="dolvm"
```

In most cases though, you will not need to set anything. For instance the root file system, which needed to be set individually in the past, is automatically detected by the GRUB2 configuration command.

So, let's generate the configuration file:

Important

Make sure that the output of the command shows that linux (and perhaps initrd) images are detected. If none are detected, then GRUB will not be able to boot the system as it has not found a Linux kernel to boot with.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-3.10.7
Found initrd image: /boot/initramfs-genkernel-x86_64-3.10.7
done
```

That's it - GRUB is now installed and configured.

Troubleshooting Boot Failures

One of the worst issues you can face is an unbootable system. Yes, it is bad, because without being able to boot, how can you fix the system (or debug / troubleshoot)? Before continuing with the most frequent boot failures (and how to deal with them), first a quick look at how to generally act when a system fails to boot...

When a system fails to boot...

A boot failure can either be due to:

- kernel configuration issue, or
- system configuration issue, or
- hardware malfunction

It's pretty easy to guess which one is the hardest to detect but easiest to resolve (hint: it's the hardware malfunction one). The other ones, well, they require a bit preparation in order to easily troubleshoot and, eventually, solve.

Kernel Parameters

GRUB allows you to boot a kernel / system with a modified set of parameters for a single time (i.e. the changes are not persisted). This allows you to (re)boot a kernel with additional parameters.

1. When you are at the GRUB screen, highlight the entry you want to boot, and press **e** (for edit).
2. Highlight the line starting with "kernel", and press **e** again.
3. Edit the line accordingly (a kernel parameter is often a word or key=value set appended to the line), and press Enter to save the changes (remember, it is just for this single boot)
4. Press **b** (for boot) to boot the kernel.

You will find that several of the kernel boot failures can be helped with one or more kernel parameters being added.

Rescue Live Linux Environment

If a system fails to boot due to a system configuration issue, I recommend to boot from a rescue environment (SysRescCD [http://www.sysresccd.org] comes to mind). Since USB sticks are widely spread and easy to work with, I suggest that you configure your system to be able to boot from USB *and test it*. It is faster than booting from most CD/DVD drives (sorry, I still don't have a Blu-Ray drive at the time of writing) and a rescue USB stick is easier to carry with you.

Having a rescue live Linux environment around you is really worth it. Configuration failures are easily made, and can have serious consequences. Gentoo Linux doesn't have a roll-back scenario (yet) that allows you to boot with previous settings.

Kernel Boot Failures

Kernel boot failures can have multiple causes, but usually the output on the screen allows you to identify the issue rapidly.

ACPI Issues

ACPI is a common source of boot failures. It isn't that the (ACPI) standard is bad, or that Linux drivers often implement it wrongly. Rather, some hardware is known not to follow the ACPI standard to the letter, making it difficult to write drivers that comply to the ACPI standard with this hardware.

Issues with ACPI support are often seen through kernel oops messages (which is a dump to the screen offering information for kernel developers regarding the problem) that point to `acpi_*` functions:

```
[     0.873537]  [<ffffffff8134ecba>] ? acpi_video_register+0x3f/0x71
[     0.873596]  [<ffffffff8100c36a>] ? do_one_initcall+0x34/0x1a0
[     0.873656]  [<ffffffff8175147d>] ? kernel_init+0x164/0x1be
[     0.876714]  [<ffffffff8100c36a>] ? child_rip+0xa/0x20
[     0.876773]  [<ffffffff81751319>] ? kernel_init+0x0/0x1be
[     0.876832]  [<ffffffff8100c360>] ? child_rip+0x0/0x20
```

Whenever you suspect that ACPI is the source of your issue, boot the same kernel, but with the *`noacpi`* parameter.

Unable to mount root-fs

Most likely one of the most occurring issue (but once you solved it, you most likely are never going to see it again):

```
Unable to mount root fs on unknown-block(0,0)
```

or

```
VFS: Cannot open root device "sda3" or unknown-block(8,3)
Please append a correct "root=" boot option; here are the available partitions:
```

```
sda driver: sd
  sda1 sda2
```

The digits "0,0" or "8,3" can be different in your case - it refers to the device that the kernel tries to access (and which fails). Generally speaking one can say that, if the first digit is 0, then the kernel is unable to identify the hardware. If it is another digit (like 8), it is unable to identify the file system (but is able to access the hardware).

The problem here is that the kernel that you are booting cannot translate the "root=/dev/..." parameter you gave it (inside the boot loader configuration) into a real, accessible file system. Several reasons can result in such a failure:

- the kernel configuration is missing drivers for your HDD controller (cases 1 [https://forums.gentoo.org/viewtopic-t-810063.html], 4 [https://forums.gentoo.org/viewtopic-p-6075078.html#6075078], 5 [https://forums.gentoo.org/viewtopic-t-801465.html])
- the kernel configuration is missing drivers for the bus used by your HDD controller
- the kernel configuration is missing drivers for the file system you are using
- the device is misidentified in your root= parameter (cases 2 [https://forums.gentoo.org/viewtopic-t-809455.html], 3 [https://forums.gentoo.org/viewtopic-t-805518.html])

Resolving the issue is easy if you know what the reason is. You most likely don't, so here's a quick check-up.

Open the kernel configuration wizard (the **make menuconfig** part) so that you can update the kernel configuration accordingly.

- Check if you have built in (and not as a module) support for the bus / protocol that your harddisk controller uses.

  Most likely this is PCI support, SATA support (which is beneath SCSI device support), ...

- Check if you have built in (and not as a module) support for the HDD controller you use.

  One of the most frequent cases: you selected support for your harddisk controller protocol (IDE, SATA, SCSI, ...) but forgot to select the HDD controller driver itself (like Intel PIIX). Try running the following **lscpi** command, and paste its output on http://kmuto.jp/debian/hcl/. The site will show you which kernel drivers you need to select for your system. Within the menuconfig, you can type "/" to open the search function, and type in the driver name to find out where it resides.

  ```
  # lspci -n
  ```

- Check if you have built in (and not as a module) support for the file system(s) you use.

  Say your root file system uses btrfs (which I definitely don't recommend) but you didn't select it, or selected it to be built as a module, then you'll get the error you see. Make sure the file system support is built in the kernel.

- Check if the kernel parameter for *root=* is pointing to the correct partition.

  This isn't as stupid as it sounds. When you are booted with one kernel, it might list your disks as being /dev/sda whereas your (configured) kernel is expecting it to be /dev/hda. This is not because kernels are inconsistent with each other, but because of the drivers used: older drivers use the hda syntax, newer sda.

  Try switching hda with sda (and hdb with sdb, and ...).

  Also, recent kernels give an overview of the partitions they found on the device told. If it does, it might help you identify if you misselected a partition (in the example given at the beginning of

this section, only two partitions are found whereas the kernel was instructed to boot the third). If it doesn't, it is most likely because the kernel doesn't know the device to begin with (so it can't attempt to display partitions).

- Check if the kernel that is being boot by the boot loader is the correct kernel.

  I have seen people who, after building a first kernel (which doesn't boot), forget that they have to mount `/boot` before the overwrite the kernel with a new one. As a result, they copy the kernel to the root file system (/) whereas the boot loader still expects the kernel image to be on the `/boot` partition.

System Boot Failures

The following set of subsections explain various failures you might get in contact with. These failures are less (or not) related with the kernel build process, but since they occur at boot-time (and due to lack of a better location in this book) I've included them in this chapter...

X (graphical environment) comes up, but keyboard or mouse doesn't work

You might have this issue when you have just upgraded your graphical environment or HAL process. By itself not a very easy thing to debug, but you'll need to get access to your system, and rebooting doesn't help (as you'll get into the same situation anyhow).

Luckily, you can ask Gentoo to boot the system without starting the graphical environment (leaving you with the command-line login from which you can fix things). Reboot until you get into the GRUB menu. Then press e(dit) on the boot item, e(dit) again on the kernel line, and add "nox" at the end of the line. This option will inform Gentoo not to load X.

Now that you are booted without the graphical environment, you can fix your system.

One reason this might happen is that X has been upgraded but you need to rebuild the various xf86-input-* packages. To get the list of installed xf86-input-* packages, you can use qlist:

```
# qlist -I xf86-input
x11-drivers/xf86-input-evdev
x11-drivers/xf86-input-keyboard
x11-drivers/xf86-input-mouse
x11-drivers/xf86-input-synaptics
```

You can reinstall those packages using the following command:

```
# emerge -1 `qlist -CI xf86-input`
```

Mind the use of the ` character at both beginning and ending of the **qlist** stanza.

Exercises

1. How is it possible that one bootloader is able to boot an operating system through a different bootloader?

Chapter 8. Hardware Support

Introduction

Some hardware is automatically enabled once you have configured it inside the Linux kernel: access to PCI chip sets, graphical card, disks, USB storage, etc. Yet, most hardware requires additional work. After all, the Linux kernel provides you with a programmatic interface to access the devices, but you still need the necessary tooling to get the device to function properly. Good examples are network cards and printers, but also sound cards.

ALSA - Advanced Linux Sound Architecture

ALSA is an open source project that provides audio functionality to the Linux kernel. It supports professional audio hardware (next to the consumer audio hardware, including sound cards) and provides a powerful, standard interface which allows, for instance, multiple software access to a single audio device. For programmers, ALSA's API is well documented and you'll quickly find that the ALSA library provides thread-safe access to the device(s).

The project also provides tools to manage the audio devices such as a simple mixer program (**alsamixer**), modular sound drivers which allow users to fine-tune the drivers' configuration aspects and of course support for the older OSS/Free implementation (Linux' previous open sound system).

Installing ALSA

Installing ALSA consists of two distinct steps:

1. Configure the Linux kernel with ALSA support and with support for your sound card(s)

2. Install the ALSA utilities

If you configure your kernel, you can either opt to install the sound card drivers in your kernel or as a kernel module. ALSA's configuration utility (**alsaconf**) assumes that you use kernel modules for your sound cards. However, this is not a requirement - you can still configure ALSA sound card drivers if they are built inside the kernel. The interface to do so however is a bit more complex.

To install the ALSA utilities, it is sufficient to emerge alsa-utils:

```
# emerge alsa-utils
```

Basic ALSA Configuration

The basic ALSA configuration starts with detecting your sound card and enabling the channels on it (sound channels) as ALSA will, by default, mute the channels (this is for precautionary reasons - you don't want to blow your speakers the first time you launch your computer do you?).

The first part (detecting the sound card) can be done using **alsaconf**. The **alsaconf** tool will attempt to detect your sound card(s), load the necessary modules and configure those with sane settings. It will save whatever it found to a general file which is read by your favourite distribution (which is undoubtedly Gentoo ;-) at start up so you don't have to rerun **alsaconf** after every boot.

```
# alsaconf
```

With your sound card(s) detected, launch **alsamixer** to view the available channels. The **alsamixer** tool will show you all channels associated with your sound card. You will find that, by default, all channels are muted. Unmute them, but bring the volume of the channels to a safe setting. Don't worry, you can increase them whenever you want later.

```
# alsamixer
```

Inside alsamixer, you can

- move from one channel to the other with the arrow keys (left/right)
- increase/decrease the volume of each channel with the arrow keys (up/down)
- mute/unmute the channel using the 'M' key
- exit the application using the Escape key (or Alt+Q)

If your sound card has particular features you can't seem to find inside the mixer application, you will need to turn to the **alsactl** command. The **alsactl** utility supports multiple devices and allows you to tweak every supported feature of your sound card. Its interface is quite simple: use alsactl to dump the sound card information to a file, then edit the file to your likings. Once finished, use alsactl to read the (modified) file back.

```
# alsactl -f /path/to/asound.state store
(Now edit /path/to/asound.state)
# alsactl -f /path/to/asound.state restore
```

If you have changed the file to such an extend that you can't get the sound to work again, you can re-initialize the settings using **alsactl init**.

Finally, if you have multiple devices, use a sequence number to identify them. You can find your list of numbers in /proc/asound/cards:

```
$ cat /proc/asound/cards
 0 [ICH6           ]: ICH4 - Intel ICH6
                      Intel ICH6 with Cx20468-31 at irq 17
```

The number (I only have one card, so mine is 0) can then be passed on to the various alsa utilities, like so:

```
$ alsamixer -c 0
```

Keeping your Changes

When you booted your system, you unmuted the channels and set the mixer channels according to your likings. However, if you do nothing more now, you'll have to redo all this again after every boot. To solve this, you need to store the current settings in a state file (yes, using alsactl) and automatically read those in at boot time.

This is exactly what the alsasound init script does (as provided by Gentoo's alsa-utils package). So, add alsasound to your boot runlevel, save your current settings and then start the initialization script:

```
# rc-update add alsasound boot
# alsactl -f /var/lib/alsa/asound.state store
# /etc/init.d/alsasound start
```

Using Sound Servers

I mentioned before that ALSA supports multiple software access to a single device. With the above configuration, you're still not able to do so. To provide such multiplexing capabilities, you can create a new audio device (some sort of mixer) which aggregates information to/from the device and sends/reads it from as many software processes as you like.

This is one of the tasks that sound servers do: these programs manage access to the sound card (interfaces) and allow multiple software processes to use the sound facilities of your system. Some well known sound servers are esd, aRTs (deprecated), JACK and PulseAudio.

- esd (Enlightenment Sound Daemon) is GNOME's sound management daemon. esd, also known as ESounD, not only supports the previously mentioned mixing, but can also manage network-transparent audio: audio played on one system can be heard on another. To this end, any application supporting esd can stream its audio to any system running esd on the network.
- aRTs (Analog RealTime Synthesizer) is KDE's former sound daemon. Although development has been abandoned, you will still find references to aRTs here and there on the Internet. Its main power was its real-time audio streaming capabilities.
- JACK (JACK Audio Connection Kit) is a real-time sound server which supports various operating systems (including GNU/Linux and Apple's OS X). It also supports network-transparent audio, real-time mixing, etc.
- PulseAudio (PulseAudio) is another sound daemon. It is meant to be a replacement for esd but with a wider support field (including Microsoft Windows and POSIX-compliant operating systems).

If you'd like to use one of these sound servers (you do need to pick one if you don't want to get confused), install one of the following packages:

- esd can be installed from media-sound/esound, although most people will already have it installed if they are running GNOME (it is a dependency of the GNOME installation)
- JACK can be installed with media-sound/jack
- PulseAudio can be installed from media-sound/pulseaudio.

Enable the corresponding USE flag (esd, jack or pulseaudio) and update your system. Portage will automatically rebuild those packages that are influenced by the USE flag change and incorporate support for the selected sound daemon in those packages:

```
# nano -w /etc/portage/make.conf
(Edit USE, add the appropriate USE flag)
# emerge --update --deep --newuse @world
```

You can also ask euse which packages are affected by a USE flag change:

```
# euse -I pulseaudio
```

If you want to know which packages all use a specific USE flag (even uninstalled packages), use **euse -i**:

```
# euse -i pulseaudio
```

CUPS - former "Common Printing Unix System"

If you need to connect your Linux system to a printer, using the CUPS tool is advised. With CUPS you can both connect to locally attached printers (USB, LPT) as well as remote (through Windows sharing or IPP). You can also use CUPS to build a print server yourself, although this is definitely outside the scope of this book.

Installing CUPS

Before you start installing the software, you will first need to make sure that your kernel configuration supports the printer:

- for locally attached printers using the (old) LPT interface, look for "Parallel port support -> PC-style hardware" and "Parallel printer support -> IEEE 1284 transfer modes")

- for locally attached printers using the USB interface, look for "USB Printer support" (as well as all other USB-required settings such as one of the xHCI supports)
- for remote printers using the Windows sharing (SMB-CIFS protocol), look for "Network File Systems -> SMB file system support" and "CIFS support")
- for remote printers using the IPP protocol, you generally do not need to enable any additional settings in the kernel

If you notice that you have not selected the right configuration yet, you'll need to rebuild the kernel and reboot (see our chapter on "Configuring a Linux Kernel").

Next, install the net-print/cups package, making sure you select the correct USE flags (this is discussed in a different chapter).

```
~# emerge net-print/cups
```

Don't worry if you do not have all USE flags correct from the first run. As I will mention later, it is always possible to update USE flags afterwards and then have Gentoo rebuild those packages affected by that change.

If your printer is locally attached, you need to start the CUPS service:

```
~# /etc/init.d/cupsd start
```

Also, make sure it is started upon every (re)boot:

```
~# rc-update add cupsd default
```

Configuring CUPS

CUPS offers a web interface to configure CUPS (and configure your printer). You can reach it through http://localhost:631. In the Administration page, enter your root login and password information and you can get started with the configuration. The Gentoo Printing HOWTO [http://www.gentoo.org/doc/en/printing-howto.xml] offers a great walk through of the configuration.

You probably hoped for a more elaborate discussion on printer configuration. Perhaps in the far future I will discuss printer configuration more, but for the time being I'm going to limit this and refer to Gentoo's guide and the main CUPS [http://www.cups.org] site.

Managing Device Files

Almost every device on your system is represented by a device file. The **udev** device manager discovers attached devices, creates device files in `/dev` (yes, you can create them - take a look at the **mknod** manpage) and often also creates symbolic links to those device files so you can find the correct device file more easily.

The **udev** tool receives events from the Linux kernel; the moment such an event is received, **udev** matches the device attributes as offered by sysfs (you can browse through `/sys` if you want to see what *sysfs* offers) against a set of rules. These rules you can view at `/lib/udev/rules.d` (provided by the udev distribution) and `/etc/udev/rules.d` (provided by third-party packages and, of course, your own rules if you write them yourself).

Gentoo offers a set of default rules which should be sufficient for most users. For instance, they create links to the (removable) disks inside `/dev/disk/by-id`, `by-path` and `by-uuid`, which should allow you to have a device link for fstab which will be the same regardless of when you plug it in (in case of a hot pluggable device, of course). This is important, because if you have, for instance, two USB storage devices, the order in which they are plugged in defines the `/dev/sd*` device naming. By using the links at `/dev/disk/by-*` you can make sure that the correct device is targeted.

Further Resources

- The ALSA [https://wiki.gentoo.org/wiki/ALSA] article on the Gentoo Wiki.
- Writing udev rules [http://www.reactivated.net/writing_udev_rules.html], written by Daniel Drake
- Gentoo Printing HOWTO [http://www.gentoo.org/doc/en/printing-howto.xml], another excellent resource by Gentoo, now on printer configuration.

Chapter 9. Software Management

Gentoo Portage

In this section, we describe how to install software on Gentoo and what Gentoo's Portage and Portage tree are. This chapter is very Gentoo-specific - in fact, if you're not using Gentoo or a Gentoo-derived Linux distribution, you'll find most of this chapter entertaining yet not usable. If you are in this scenario, please do check the documentation of your distribution: software management is a part of a distributions' responsibilities which isn't standard(ised) in the free software community yet (because there are so many different ways, policies, politics and functionalities).

I'm also not saying standardisation here is needed right now - the functionalities offered by the various distributions are too different to make standardisation possible at this point.

Introduction

Gentoo is primarily a source-code based Linux distribution. This means that, unlike most operating systems (including other Linux distributions), Gentoo does not just place compiled files on your file system but actually fetches the source code for the software you need from the Internet, configures the compilation environment and builds the software before it finally moves the resulting build onto the file system.

Of course, Gentoo does support binary package installations, and this on two levels:

1. Software titles that would take a very long time to build on a users' system (such as the popular LibreOffice.org office suite) are available as binary packages. In this case, Gentoo downloads the binary files and installs those on the file system.

2. Prebuilt packages are supported as well: software built on a different Gentoo system, stored in a binary format file and made available to other Gentoo systems *with* the same settings (otherwise the binary packages might not be compatible with the system).

The second level is a more advanced approach as Gentoo doesn't provide binary packages itself - you'll need to create and maintain a build environment yourself or use a third-party repository. As such, I am going to focus on source-code installations and the first binary package level (which is automatically supported).

Package Management: Portage

The software that is responsible for managing installed software is called a package manager. It is the interface to the user to install and uninstall software, upgrade the system, search through the available software and help with the configuration of the various software titles.

For Gentoo, the default package manager is called Portage. I say "default" because there are other package managers available for Gentoo as well (like Paludis and pkgcore). These other package managers generally have the same features, but behave a bit differently. They are all compatible though, thanks to a standard approach to the packaging structure Gentoo uses.

The package manager manages a local package repository called the Portage tree. This package repository is a set of package metadata (so no source code or binary package) about software that can be installed on Gentoo.

You can find the Portage tree at `/usr/portage`.

Package Structure: Ebuilds

The package metadata files are called *ebuilds*. Ebuilds contain information about the package:

- the name of the package, the internet site where you can find information about the software and the version of the software package
- the *dependencies* of the package (what other software is needed in order to successfully build or run this software)
- the location where the software source code (or binary files in case of binary packages - first level) can be downloaded from
- the build instructions for the software (remember, Gentoo is primarily a source code based distribution)
- the license the software uses
- the USE flags the package supports (USE flags are described later)
- the configuration instructions needed to finalize the installation after building and installing the software
- ...

Because all this information is simple, the ebuild file is a text file. This allows users to easily read what an ebuild actually does (although this does require knowledge about scripting and the ebuild format itself which isn't in the scope of this chapter).

These packages are the main source for software installation and management. In order to keep track of new software releases, you will need to update the collection of ebuilds (Portage tree).

USE Flags

A USE flag is one of Gentoo's main strengths. USE flags are simple terms (such as "dvd", "X", "mysql" ...) and are used by ebuilds to detect what options you want enabled (or disabled) while building software.

Most applications have optional support for one or more tools and libraries. For instance, at the time of writing, the Mozilla Firefox application in Gentoo supports the following use flags (I dropped the various linguas flags which enable/disable support for certain language locales in the output to make it more readable):

```
$ equery uses firefox
 + + alsa                : Add support for media-libs/alsa-lib (Advanced Linux
 - - bindist             : Disable official Firefox branding (icons, name) whi
 - - custom-cflags       : Build with user-specified CFLAGS (unsupported)
 - - custom-optimization : Fine-tune custom compiler optimizations, setting th
 + + dbus                : Enable dbus support for anything that needs it (gps
 - - debug               : Enable extra debug codepaths, like asserts and extr
 - - gstreamer           : Add support for media-libs/gstreamer (Streaming med
 - - jit                 : Enable just-in-time compilation for improved perfor
 - - libnotify           : Enable desktop notification support
 + + minimal             : Prevent sdk and headers from being installed
 - - pulseaudio          : Add support for PulseAudio sound server
 - - startup-notification : Enable application startup event feedback mechanism
 - - system-cairo        : Use the system-wide x11-libs/cairo Use system cairo
 - - system-jpeg         : Use the system-wide media-libs/libjpeg-turbo Use sy
 - - system-sqlite       : Use the system-wide dev-db/sqlite installation with
 - - wifi                : Enable wireless network functions
[ Legend : U - final flag setting for installation]
[        : I - package is installed with flag     ]
[ Colors : set, unset                             ]
 * Found these USE flags for www-client/firefox-23.0:
```

```
 U I
```

Other distributions choose what support they want enabled and what not when building the software. Because with Gentoo, you build the software yourself, you have the opportunity to enable or disable these features based on your USE flags.

You can set your global USE flags inside `/etc/portage/make.conf`. To disable a USE flag, prepend it with the minus sign.

```
# nano -w /etc/portage/make.conf
(...)
USE="acpi apm pmu sasl howl -cdr -dvdread ..."
```

Because of the vast amount of USE flags that Gentoo supports, it is not your job to know and understand every single one of them. Gentoo is shipped with a sane default for your USE variable. To find out what your current USE flags are (based on Gentoo's default USE flags plus the USE flags you enabled or disabled in `/etc/portage/make.conf`), use **emerge --info**, filtering on the USE term:

```
$ emerge --info | grep USE
USE="X acl acpi alsa apache2 apm arts berkdb bitmap-fonts cairo cdr
cli cracklib crypt cscope css cups dbus dri dvd dvdr dvdread eds
emboss encode esd evo fam firefox fortran gdbm gif gnome gpm
gstreamer gtk hal howl iconv imap innodb ipv6 isdnlog java jpeg kde
kerberos ldap libwww mad maildir midi mikmod mp3 mpeg mudflap mysql
ncurses nls nptl nptlonly nsplugin odbc ogg opengl openmp oss pam
pcre pdf perl pmu png pppd python qt3 qt3support qt4 quicktime readline
reflection sasl sdl session spell spl ssl svg tcpd tetex tiff truetype
truetype-fonts type1-fonts unicode vcd vhosts vorbis win32codecs x86
xml xorg xv zlib" ALSA_CARDS="ali5451 als4000 atiixp atiixp-modem
bt87x ca0106 cmipci emu10k1 emu10k1x ens1370 ens1371 es1938 es1968
fm801 hda-intel intel8x0 intel8x0m maestro3 trident usb-audio via82xx
via82xx-modem ymfpci" ALSA_PCM_PLUGINS="adpcm alaw asym copy dmix
dshare dsnoop empty extplug file hooks iec958 ioplug ladspa lfloat
linear meter mulaw multi null plug rate route share shm softvol"
APACHE2_MODULES="actions alias auth_basic authn_alias authn_anon
authn_dbm authn_default authn_file authz_dbm authz_default
authz_groupfile authz_host authz_owner authz_user autoindex
cache dav dav_fs dav_lock deflate dir disk_cache env expires
ext_filter file_cache filter headers include info log_config logio
mem_cache mime mime_magic negotiation rewrite setenvif speling
status unique_id userdir usertrack vhost_alias" ELIBC="glibc"
INPUT_DEVICES="keyboard mouse" KERNEL="linux" LCD_DEVICES="bayrad
cfontz cfontz633 glk hd44780 lb216 lcdm001 mtxorb ncurses text"
LINGUAS="en nl fr de" USERLAND="GNU" VIDEO_CARDS="i810 i915 vesa"
```

To obtain some information on a USE flag, you can use the **euse** command:

```
$ euse --info mp3
global use flags (searching: mp3)
**********************************************************
[+  D ] mp3 - Add support for reading mp3 files

local use flags (searching: mp3)
**********************************************************
no matching entries found
```

If you want to manage USE flags on a per-package level (say you want to include MP3 support in mplayer but not in vlc) you can set per-package USE flags in `/etc/portage/package.use`. This location can either be a simple text file, but you can also make it a directory which contains many

text files. This latter feature is very interesting if you want to manage the USE definitions more easily (say by aggregating media-related USE flag settings). Gentoo's Portage will read all files inside this directory.

```
# mkdir /etc/portage/package.use
# nano -w /etc/portage/package.use/media-flags
media-video/vlc mp3
```

Maintaining your Software Stack

With software stack maintenance, I mean installing and deinstalling software, updating your system and keeping track of the security patches for your system.

Obtaining the Latest Portage Tree

As stated before, the Portage tree is the collection of ebuilds (which we will call "packages" from now onwards) that you can install on your system. Whether you want to update your system, install new software or obtain the latest security fixes, you will first need to update your Portage tree.

The command to update the Portage tree is **emerge --sync**. The sync comes from synchronisation and gives you an idea on what **emerge** (Portage' main tool for software management) will do: it will synchronise your local Portage tree with the Portage tree available on one of Gentoo's mirrors.

```
# emerge --sync
```

You can easily configure what mirror **emerge** should use. For this, open up `/etc/portage/make.conf` (Gentoo Portage's main configuration file) and locate the SYNC setting:

```
# nano -w /etc/portage/make.conf
(...)
SYNC="rsync://rsync.us.gentoo.org/gentoo-portage"
```

The SYNC setting is set to a public Portage mirror. In the example, SYNC is set to a round-robin address which covers all mirrors in the United States of America. You can substitute the "us" country code with your country ("be" for Belgium, for instance) to use the round-robin address for the mirrors in your country.

You notice that the setting references the rsync protocol. Indeed, rsync is used to synchronise your local Portage tree. The tool (and protocol) has been selected because it only transfers the differences between two files rather than downloading the entire new tree.

If you cannot use rsync (for instance because of a firewall setting) you can use **emerge-webrsync** (without any options). This tool will fetch an archive which contains the entire Portage tree using the HTTP or FTP protocol and synchronise this archive locally. Of course, this will take more network traffic as an entire tree is downloaded.

Also, Gentoo rsync mirrors update their Portage tree hourly. The snapshots retrieved using **emerge-webrsync** are updated daily.

Getting Gentoo News

With an up to date Portage tree, you will also receive Gentoo's latest announcements for its users. Gentoo's news system allows Gentoo to inform its users about changes that the end user should read before blindly starting with an upgrade. This is also why I mention it before we start doing anything with software.

When a news item is added, you will be notified of this at the end of a Portage update session:

```
* IMPORTANT: 21news items need reading for repository 'gentoo'.
```

```
* Use eselect news to read news items.
```

When you find such a notice at the end of your update, you can use Gentoo's **eselect** tool to find out more:

```
# eselect news list
News items:
  [1]      2009-04-18  Generation 1 Java Setup Deprecated
  [2]      2009-07-02  (2009-07-02-kdeprefix+monolithics - removed?)
  [3]      2010-03-25  Python 3.1
  [4]      2010-08-01  (2010-08-01-as-needed-default - removed?)
  [5]      2010-10-22  Perl 5.12 upgrade procedure
  [6]      2011-04-27  Upgrade to GLIB 2.28
  [7]      2011-05-01  Baselayout update
  [8]      2011-05-22  (2011-05-22-kdeprefix - removed?)
  [9]      2011-08-28  Mesa r600 driver now defaults to gallium
  [10]     2011-10-15  Upgrade to libpng15
  [11]     2012-03-16  (2012-03-16-udev-181-unmasking - removed?)
  [12]     2012-04-24  The default JPEG implementation
  [13]     2012-05-21  Portage config-protect-if-modified default
  [14]     2012-09-09  make.conf and make.profile move
  [15]     2012-09-09  Catalyst updates
  [16]     2012-11-06  PYTHON_TARGETS deployment
  [17]     2013-01-23  (2013-01-23-udev-upgrade - removed?)
  [18]  N  2013-03-29  Upgrading udev to version >=200
```

You can then read the news items one by one (**eselect news read <number>**) but I prefer to just tell eselect to display the content of the unread news items:

```
# eselect news read new
```

Querying for Software

With tens of thousands of software titles at your fingertips, how do you know what software is available (or what software you currently have installed)? Or what about querying information about software itself (like installed or installable files, dependencies ...), even when that software is or is not installed on your system? There are quite a few tools that help you with this...

Searching for Software Using emerge

With emerge, you can query for software using **--search** or **--searchdesc**.

With **emerge --search**, you can look for software whose title (or part of) matches the selection you passed:

```
$ emerge --search acroread
Searching...
[ Results for search key : acroread ]
[ Applications found : 2 ]

*  app-text/acroread
      Latest version available: 9.4.2
      Latest version installed: [ Not Installed ]
      Size of files: 121,848 kB
      Homepage:      http://www.adobe.com/products/acrobat/
      Description:   Adobe's PDF reader
      License:       Adobe

*  media-fonts/acroread-asianfonts
```

```
      Latest version available: 9.1
      Latest version installed: [ Not Installed ]
      Size of files: 5,767 kB
      Homepage:      http://www.adobe.com/products/acrobat/acrrasianfontpack.htm
      Description:   Asian font packs for Adobe Acrobat Reader
      License:       Adobe
```

In the resulting output, you can find the (in this case two) packages that match the given selection, together with

- the latest, installable version
- the currently installed version (if applicable)
- the size of the files that the installation needs to download
- the homepage of the application (where you can usually find more information about the tool)
- the description of the software
- the software license

If you use **--searchdesc**, the search also tries to match the description fields:

```
$ emerge --searchdesc acrobat
Searching...
[ Results for search key : acrobat ]
[ Applications found : 3 ]

*  app-admin/watchfolder
      Latest version available: 0.3.3
      Latest version installed: [ Not Installed ]
      Size of files: 93 kB
      Homepage:      http://freshmeat.net/projects/watchd/
      Description:   Watches directories and processes files, similar to the wa
      License:       GPL-2

*  app-text/fdftk
      Latest version available: 6.0-r1
      Latest version installed: [ Not Installed ]
      Size of files: 5,591 kB
      Homepage:      http://www.adobe.com/devnet/acrobat/fdftoolkit.html
      Description:   Acrobat FDF Toolkit
      License:       Adobe
```

Running **--searchdesc** takes quite some time. I advise you to use an indexing tool to speed up searches...

Searching for Software Using eix

The eix tool (not available by default, you will need to install it: **emerge --ask eix**) indexes the searchable application information to allow you to quickly search through the portage tree.

Once installed, you will need to run **update-eix** (which will index all the information) after every **emerge --sync** (as this pulls in new packages which you want to index). You can also use **eix-sync** which will synchronise portage first and then update its indexing tables immediately afterwards.

Once the index is built, you can quickly search through the Portage tree using eix. I frequently use **-HAS** (**--homepage --category-name --description**) as this also matches the homepage URI next to the application name and description:

```
$ eix -HAS acrobat
```

Searching Installed Software Based on File(s)

The **qfile** tool (part of app-portage/portage-utils) allows you to find out which package has delivered a certain file. That can be interesting if someone is asking you which package provided that particular file at your system.

```
$ qfile /usr/bin/qfile
app-portage/portage-utils (/usr/bin/qfile)
```

Searching Software (Installed or Not) Based on File(s)

A Gentoo community project, called PortageFileList [http://www.portagefilelist.de], is attempting to store information about files provided by all packages. This isn't as simple as it sounds, since the files in a package depend heavily on the user his system: USE flags, other installed packages etc. The project asks users to run one of their tools to upload the local file information (file + package + USE flags) to the server. Other users can then use the tool **e-file** (part of app-portage/pfl) to query for packages.

The command supports SQL-like wildcards. For instance, `%bin/xdm` matches `/usr/sbin/xdm` and `/bin/xdm`.

```
$ e-file '%bin/xdm'
 * x11-apps/xdm
      Available versions: 1.1.8 1.1.9-r0
      Description:        X.Org xdm application
      Matched files:      /usr/bin/xdm;
```

Okay, this one was an easy guess as well ;-)

Listing Installed Files

To list which files are provided by a package, you can use **qlist** (also part of app-portage/portage-utils):

```
$ qlist portage-utils
/usr/bin/q
/usr/bin/qatom
...
/usr/share/doc/portage-utils-0.2.1/qsync
```

Enabling Installation Logs

When a Gentoo developer wants to inform a package user about certain changes or things to look out for, he often puts informational messages in the package itself which are displayed the moment you've finished the installation of the package. However, for lengthy updates or installations, you often overlook these messages. For this purpose, Gentoo Portage allows you to store these messages in separate log files. By default, Portage only sets up a summary log for the installation. You can stick with those defaults, or set up a more verbose logging:

```
# nano /etc/portage/make.conf

PORT_LOGDIR="/var/log/portage"
PORTAGE_ELOG_SYSTEM="save"
PORTAGE_ELOG_CLASSES="info warn error log"
```

With this defined, you'll find per-package logs in `/var/log/portage/elog`. You can then read those log files with your favorite text reader (like **less** or **view**) or using elog-specific tools like **elogv** (console) or **elogviewer** (graphical).

Installing New Software

You've seen already that emerge is Gentoo's main tool to work with Portage and that it can be used to search for software. The same tool is used to install or remove software. To help users install software, it has a few options that allow you to make better informed decisions on which software to install (and what the effects will be). For instance, the **--pretend** option lets emerge show you what it would do, without actually modifying anything. You can even add the **--verbose** flag so that emerge tells you what additional settings the package supports. It also supports the **--ask** option which first behaves like **--pretend**, and then asks for confirmation to continue or not.

For instance, to install Adobe's Acrobat Reader on your system:

```
# emerge --ask --verbose acroread
These are the packages that would be merged, in order:

Calculating dependencies   ... done!
[ebuild  N    ] app-text/acroread-8.1.2  USE="cups ldap nsplugin"
  LINGUAS="de en fr nl -da -es -fi -it -ja -ko -nb -pt -sv -zh_CN -zh_TW"

Would you like to merge these packages? [Yes/No]
```

Note

Sometimes it might be necessary to use the category/package-name syntax (in this case, **emerge --ask --verbose app-text/acroread**). This could happen when two packages share the same name but are part of different categories.

In this example, emerge tells you that it will install Adobe Acrobat Reader version 8.1.2 and that it doesn't need any other dependencies (otherwise you'll have more lines in the same line as the one you see). Emerge also tells you that the acroread package supports three USE flags:

- cups (support for the Common Unix Printing System),
- ldap (support for the Lightweight Directory Access Protocol) and
- nsplugin (creating Acrobat Reader plugins for the Mozilla-based browsers)

All three flags are in this case enabled on this system.

Also, the acroread package supports various languages: de (German), en (English), fr (French), nl (Dutch), da (Danish), es (Spanish), fi (Finnish), it (Italian), ja (Japanese), ko (Korean), nb (Norwegian Bokmal), pt (Portugese), sv (Swedish), zh_CN (Simplified Chinese) and zn_TW (Taiwanese Mandarin). Of all these languages, Portage will only include support for German, English, French and Dutch (the languages I want supported on my system).

To install the software, accept the merge. This will, for every package:

1. download the source code from the Internet,
2. verify the source code with the checksum stored in the Portage tree (to ensure that the downloaded file isn't corrupt),
3. extract the source code to a temporary location on the system (`/var/tmp/portage`),
4. patch the source code with Gentoo-specific patches,
5. configure the source code based on the USE flags you use,
6. build the source code into a temporary location,
7. register the files it has created in this location, and finally

8. move the files to the system (making the application available to the end user)

This installation process will take some time (compared to distributions who use prebuilt packages), mainly because of the build step. There are tools that show the user how long the emerge phase of a package took in average, but these are only for informational purposes and give no guarantee whatsoever about future emerge phase durations (as this depends on USE flags, source code size, CPU speed, system resource availability ...).

qlop (part of the `portage-utils` package) is such a tool. In the next example, it shows the build duration averaged over 27 merges:

```
$ qlop --time --human mozilla-firefox
mozilla-firefox: 1 hour, 15 minutes, 22 seconds for 27 merges
```

Updating your System

When you are booted inside a Gentoo Linux system, many packages will already be installed. It is wise to regularly update your system by pulling in the latest updates, bugfixes and security fixes.

Updating All Installed Packages

If you want to update all installed packages to their latest, stable version, first update your Portage tree as mentioned before. Then, run the following command to see what packages will be updated, installed or removed:

```
# emerge --update --deep --newuse @world --pretend
```

The options you pass on to emerge are:

- **--update** (or **-u**), asking emerge to update software,
- **--deep** (or **-D**), asking emerge to select not only the software selected, but also all of its dependencies,
- **--newuse** (or **-N**), asking emerge to update software when new dynamic switches are used (USE flags),
- @world, asking emerge to select all software that the user has installed, and
- **--pretend** (or **-p**), asking emerge to display what it would install, update or remove

To really perform the update, use the same command without **--pretend**. The next example does this using the single letter arguments:

```
# emerge -uDN @world
```

Now, if you're tired of continuously running a command with **--pretend** only to remove the **--pretend** after verifying the output, use **--ask** instead as we discussed before.

From now onwards, I will be using the **--ask** argument where appropriate.

You might have noticed that the world argument is prefixed with the at sign (@). This is not mandatory - in the past it was without - but a while ago, Portage started supporting what we call "sets" of packages. The world target is one of such sets, and sets are prefixed with an at sign to distinguish them from regular packages.

Updating All User-Installed Packages

If you don't want to upgrade every single package on your system, but only the packages you've selected to install previously, don't use the **--deep** argument. User installed packages are the packages you've asked to install excluding the dependencies that were pulled in to satisfy the application requirements. The end result is that only a small amount of updates is installed. However, this doesn't

mean that you will quickly have outdated dependencies - every time an update is pulled in, this update might instruct Portage that it can only work with a higher version of a particular dependency, causing Portage to update that dependency as well.

```
# emerge --update --newuse --ask @world
```

Do not Stop Installing after Build Failures

By default, Gentoo Portage stops the installation / upgrade process when one package fails to build properly. Although this definitely needs some attention, you can ask Gentoo Portage to resume building the other packages first. At the end of the final build process, you will be greeted with a small report informing you which packages have failed and why.

To ask Portage to keep going after failure, use **--keep-going**:

```
# emerge -uDN --keep-going @world
```

Pulling Security Fixes

Gentoo also supports building the software affected by security issues alone. In this case, none of your unaffected software is upgraded (unless requires as a dependency of a security upgrade). Gentoo provides GLSA (*Gentoo Linux Security Advisory*) documents to inform its users about possible security issues (except kernel issues). These GLSA documents are parsed by the **glsa-check** command which can apply the necessary fixes.

To verify if your system is affected by one or more security advisories, first update your Portage tree and then run **glsa-check --test all**:

```
# glsa-check --test all
This system is affected by the following GLSAs:
200711-10
```

In the example, one security advisory matches my system. I can obtain more information about the advisory using **glsa-check --dump 200711-10**. To fix the system, I can run glsa-check's --fix:

```
# glsa-check --fix all
(This will iterate over all GLSAs, including the matching one(s))
```

Switching between software versions

Gentoo supports having multiple versions of the same software on the system. Notable examples are gcc, python and java but many more exist. Administrators are able to switch between these installed versions using **eselect**. We have seen this tool already previously for switching between profiles.

For instance, to display the installed Python versions:

```
# eselect python list
Available Python interpreters:
  [1] python2.7
  [2] python3.1 *
```

In the above example, Python 3.1 is the system default. You can switch the system to 2.7 then using **eselect python set 1**.

To get an overview of the supported modules in eselect, run **eselect help**.

Uninstalling Software

Now suppose you've found some software installed on your system that you want to remove. With **emerge --unmerge** (**-C** in short) you can remove installed software from your system.

```
# emerge --unmerge acroread --ask
```

However, if this software is a dependency of an installed software title, your next system update will pull the software back in. Even worse, software that depends on it that is still being used / running might even stop functioning properly or crash. A safer way of pursuing the de-installation of a package is by asking Portage to delete the package but only if no other dependencies point to it. To do so, use **emerge --depclean** (or in short, **emerge -c**):

```
# emerge --depclean acroread
```

With this command, Portage will remove the package if it is no longer in use otherwise. In other words, if no other package depends upon acroread, it will remove acroread. However, if some package still depends upon acroread, Portage will let acroread remain on your system and inform you of the dependencies towards it.

Advanced Software Management

Of course, Gentoo allows for much more advanced software management aspects than described before. If you search the available documentation, you'll quickly notice that Gentoo Portage supports the notion of stable versus "unstable" packages (the so-called ~arch states), supports third-party repositories and more.

Package States

Gentoo packages, which are called ebuilds, are always in a certain state. Although this state is never explicitly defined in the package, it is a result of a set of rules set forth by developers to keep track of a package's stability, security, license rules, ...

When you install packages, those rules are checked to make sure you are not installing a package that shouldn't be installed on your system. These various rules can trigger installation errors, most of the time described as being a *package mask*.

Whenever an installation is triggered, Gentoo Portage tries to install the highest version of a package which matches all possible validity rules. Only when no such version exists, or when you explicitly asked to install a certain version, Gentoo aborts the installation with a failure message.

The following sections describe the various states / rules that are checked when an installation is triggered. If you want to work around such a rule, see the section on "Unmasking Packages".

Architecture Availability

Gentoo is a multi-architecture operating system. Packages available within Gentoo can be installed on several architectures, ranging from the popular x86/amd64 onto sparc, mips, powerpc, ... But of course, most packages can't be installed on any platform. Some tools are only supported on the Intel-like platforms (x86, amd64) or on a specific platform (like the silo boot loader, which is explicitly made for the sparc platform).

Packages are marked as being available for a particular platform using their `KEYWORDS` variable. An example, taken from sys-apps/iproute2 version 2.6.29-r1:

```
KEYWORDS="alpha amd64 arm hppa ia64 m68k ~mips ppc ppc64 s390 sh sparc x86"
```

When you install a package, its platform keywords are checked against the keywords that your system accepts. These keywords are defined in the `ACCEPT_KEYWORDS` variable, which you can query with **emerge --info**:

```
$ emerge --info | grep ACCEPT_KEYWORDS
```

```
ACCEPT_KEYWORDS="amd64"
```

Whenever a mismatch occurs (the package doesn't have the keyword that you have defined in ACCEPT_KEYWORDS) you get an error message like so:

```
!!! All ebuilds that could satisfy "sys-boot/silo" have been masked.
!!! One of the following masked packages is required to complete your request:
- sys-boot/silo-1.4.14 (masked by: missing keyword)
```

Having an error due to an unsupported architecture, it is not advisable to try and work around this (forcing the package to be installable). It will most likely not work anyhow.

Other related error messages that you might receive are:

```
- lm-sensors/lm-sensors-2.8.7 (masked by: -sparc keyword)
- sys-libs/glibc-2.3.4.20040808 (masked by: -* keyword)
```

These error messages are similar to the one above (missing keyword). The differences are:

- "missing keyword" means that the architecture isn't set in the KEYWORDS variable.

  The architecture might be supported by the package, but no developer has tried to install it on this platform yet.

- "-sparc keyword" (or other architecture) means that that particular architecture is explicitly denied in the KEYWORDS variable.

  In this case, the developer has either tried to install it, but it destroyed its system, or the package itself is known not to work on the mentioned platform.

- "-* keyword" means that all architectures are explicitly denied in the KEYWORDS variable except a few.

  This case is mostly used for packages that are meant to be available on a single platform only. The developer marks the package as unavailable for any other architecture, hence the "-*".

Stabilization of Gentoo Packages

When a developer creates or edits a Gentoo package, the software isn't immediately available for the general public. First, it is put through a sort of *staging period*. During this time, testers try out the package, finding possible package definition bugs (like wrong dependencies) and helping the further development until the package is deemed stable.

There is an unwritten rule that says that packages should "linger" in the staging period for 30 bug-free days (no additional bugs found by the architecture testers) before it is released to the general public.

Within Gentoo, this staging period is described by prepending the tilde (~) sign in front of the architecture in the KEYWORDS variable. In the KEYWORDS example above (of the iproute2 package) you notice that this is done for the mips architecture. In other words, the package most likely is supported on the MIPS architecture, but it is currently under review (in staging period). It is well possible that within a month, the package is made available for the general MIPS public as well.

Whenever you try to install a package that is in the staging period, you get an error message as follows:

```
!!! All ebuilds that could satisfy ">=sys-apps/portage-2.1.8" have been masked.
!!! One of the following masked packages is required to complete your request:
- sys-apps/portage-2.2_rc62 (masked by: ~amd64 keyword)
```

If you get such an error message, I recommend to wait until the package is generally available (and let the architecture testers bring out most of the bugs for you).

Critical Bugs or Security Issues

Whenever a package has a critical bug (critical = breaks your system severely), a security issue (like having an exploitable vulnerability that might grant malicious users access to your system) or just isn't maintained any more, the Gentoo developer might mask the package explicitly.

Explicit masks do not only disallow the installation of the software, they usually also contain a comment as to why the installation is prohibited. These explicit masks (and their comments) are stored in a file called `package.mask`. By default, this file is available in `/usr/portage/profiles`.

An example would be:

```
!!! All ebuilds that could satisfy "games-roguelike/slashem" have been masked.
!!! One of the following masked packages is required to complete your request:
- games-roguelike/slashem-0.0.772 (masked by: package.mask)
/usr/portage/profiles/package.mask:
# Tavis Ormandy <taviso@gentoo.org> (21 Mar 2006)
# masked pending unresolved security issues #127167
```

In this example, you find out who masked the package, when the mask occurs, and why (including a bug number which can be checked at https://bugs.gentoo.org).

License Approval

Gentoo packages always keep track of the license under which the software is distributed / allowed to be used. Like the architecture available cases, this is done through a variable called `LICENSE`. For instance, for the net-im/skype package:

```
LICENSE="skype-eula"
```

And, just like the KEYWORDS case, your system has a variable called `ACCEPT_LICENSE` in which you set which software licenses you want on your system. The default setting on Gentoo systems is quite liberal, but you can set it to your liking. For instance, you can accept only FSF approved licenses:

```
$ emerge --info | grep ACCEPT_LICENSE
ACCEPT_LICENSE="-* @FSF-APPROVED"
```

The use of the @-sign allows Gentoo to group similar licenses together. The list of license groups can be obtained from `/usr/portage/profiles/license_groups`.

If you ever want to install a software which doesn't match the license(s) you allow, you will get an error like so:

```
!!! All ebuilds that could satisfy "net-im/skype" have been masked.
!!! One of the following masked packages is required to complete your request:
- net-im/skype-2.0.0.72 (masked by: skype-eula license)
```

In this example, the package I want to install uses the "skype-eula" license which doesn't match the licenses I allow (which are only the `FSF-APPROVED` ones).

Profile Supported Software

Gentoo supports several profiles. A *profile* should be seen as a collection of predefined settings for your Gentoo Linux environment. These settings include default USE flags, default license acceptance, default package choices, etc. Such profiles can also have masks against certain packages because they don't fit in the spirit of the profile.

An example is the selinux/v2refpolicy/x86/hardened profile, which masks out glibc-2.4 as it doesn't match the security measures that the profile wants to enforce.

Whenever you try to install a package that isn't allowed due to your profile, you get a message like so:

```
!!! All ebuilds that could satisfy "=sys-libs/glibc-2.4*" have been masked.
!!! One of the following masked packages is required to complete your request:
- sys-libs/glibc-2.4-r2 (masked by: profile)
```

Unmasking Packages

When you get an installation failure because the package you want to install is considered masked, you *can* opt to install it anyway. I recommend that you consider this step carefully - masking of packages is done with a reason.

- If it is because of a critical bug or security issue, I seriously discourage you to install it (for obvious reasons).
- If the package is masked due to the stabilization track, I urge you to wait until it is generally available.
- If the package is masked due to architecture unavailability, I suggest that you investigate if the package supports your architecture. If it doesn't, there's no point in trying to install it anyway. If it does, you might want to ask the architecture testing team of Gentoo to investigate the package.
- If the package is masked due to profile mismatch, you're either running the wrong profile or have made a mistake in trying to install the package and will now continue with your life as it is.
- If the package is masked due to license mismatch, then your license setting is incorrect.

Now, the last reason (license settings) is (in my opinion) the only one that warrants a change in the configuration, but the way on dealing with these blocks is similar.

System-wide or Package Specific?

Whenever you want to resolve a masking issue, you need to ask yourself if you want to resolve this for this particular package only (or a set of packages) or if you want to alter the settings for the entire system. In the majority of cases (except for licenses) it is for a particular package (or set of packages).

Portage has a specific configuration directory (`/etc/portage`) in which deviation settings can be stored. Each type of deviation has its own file or directory:

- `/etc/portage/package.license` allows you to set license-deviations (approve or deny a license on a per-package basis)
- `/etc/portage/package.mask` allows you to mask packages (deny packages on a per-package basis)
- `/etc/portage/package.unmask` allows you to unmask packages (allow packages that are disapproved due to critical bugs, security issues or profile mismatches, on a per-package basis)
- `/etc/portage/package.accept_keywords` allows you to set keyword deviations (inform Portage that for this package, other keywords are accepted as well)

Note

Previous versions of Portage use `package.keywords` instead of `package.accept_keywords`. Recent Portage versions still support this old location, but it is recommended to rename them to package.accept_keywords to remain functional in the future.

Now, I did say "file or directory" in the previous paragraph. That's right: Portage allows you to either put the information in files that are called as mentioned above, or in files that reside in a directory called as mentioned above.

The most proper way, in my opinion, is to create directories. Within each directory, you can then group deviations based on the reason.

If you want to make a system-wide deviation, you need to edit the `/etc/portage/make.conf` file.

Example: Allow the Skype-EULA License for the net-im/skype package

As an example, let's allow the skype EULA license for the skype package (and only for the skype package).

1. Create a directory called `package.license` within `/etc/portage`.

   ```
   # mkdir -p /etc/portage/package.license
   ```

2. Create a file called "skype" inside this directory, and enter "net-im/skype skype-eula".

   ```
   # nano -w /etc/portage/package.license/skype
   net-im/skype skype-eula
   ```

That's it. If you now would launch the installation of the skype package, it will not abort saying that the license is masked because you explicitly allowed the license on this package. However, if for some reason another package uses the same license, it will still fail installing because that particular package isn't mentioned as a valid deviation.

Example: Allow the Skype-EULA License System-wide

The other way is to allow the skype EULA system-wide.

1. Find out what your current license acceptance rule is.

   ```
   $ emerge --info | grep ACCEPT_LICENSE
   ACCEPT_LICENSE="-* @FSF-APPROVED"
   ```

2. Open `/etc/portage/make.conf` in your favourite text editor:

   ```
   # nano -w /etc/portage/make.conf
   ```

3. Navigate to the place where ACCEPT_LICENSE is defined or, if it isn't defined, create a new line and make it so that it looks like the current setting as obtained from the first step:

   ```
   ACCEPT_LICENSE="-* @FSF-APPROVED"
   ```

4. Now add the skype-eula license:

   ```
   ACCEPT_LICENSE="-* @FSF-APPROVED skype-eula"
   ```

5. Save the file and exit.

If you now install the skype package, it will not fail due to the license, as you now state that you accept this license. If any other package also works with the skype EULA license, it will also be installable now.

Example: Allow bonnie++ Version 1.96 (Staging State) to be Installable

If you want to install app-benchmarks/bonnie++-1.96 but find out that the package is masked (say ~x86 keyword mask), you can inform Portage to allow the installation of this particular package regardless of it being in the staging state or not:

1. Create the `/etc/portage/package.accept_keywords` directory

   ```
   # mkdir -p /etc/portage/package.accept_keywords
   ```

2. Create a file (say benchmarks) and add "app-benchmarks/bonnie++" to it:

   ```
   # nano -w /etc/portage/package.accept_keywords/benchmarks
   app-benchmarks/bonnie++
   ```

With this setting in place, Portage will allow you to install bonnie++ (any version) regardless if it is still in the staging state or not.

If you only want to allow this for the 1.96 version, use the following line instead:

```
=app-benchmarks/bonnie++-1.96
```

Example: Allow All Staging State Packages to be Installable

Warning

Setting this is greatly discouraged. You do not get support from Gentoo developers if you do this!

Oh, and if you do it, your next system update will pull in a lot of packages, and chances are slim that you can safely downgrade (so this step is difficult to undo).

To allow the staging packages to be installable on your system (any package in the staging state), edit `/etc/portage/make.conf` accordingly:

1. Find out your current ACCEPT_KEYWORDS value.

   ```
   $ emerge --info | grep ACCEPT_KEYWORDS
   ACCEPT_KEYWORDS="x86"
   ```

2. Open `/etc/portage/make.conf` in your favourite editor

   ```
   # nano -w /etc/portage/make.conf
   ```

3. Navigate to the ACCEPT_KEYWORDS variable or, if it doesn't exist, create it with the value obtained in the first step:

   ```
   ACCEPT_KEYWORDS="x86"
   ```

4. Now add the staging sign to it:

   ```
   ACCEPT_KEYWORDS="~x86"
   ```

5. Save the file and exit.

From this point onwards, Portage will install the highest available version of a package regardless of it being in the staging state or not.

Other Examples...

From the previous examples, you should get a grip on how to handle masked packages:

- either create a file in a `/etc/portage` subdirectory, or
- edit the make.conf file to make things globally

You can find more information on the syntax used within the `/etc/portage` subdirectories through the portage manpage:

```
$ man portage
(...)
(Type in "/" followed by the file name you want to know more about)
/package.mask
(Type "n" (next) until you get the section information)

package.mask
  A list of package atoms to mask. Useful if specific ver-
  sions of packages do not work well for you. For example,
  if you swear ...
```

Automatically unmasking packages

More than often, a staging-state masked package has dependencies upon other masked packages, who have dependencies upon other masked packages, etc. To manually enter these dependencies in /etc/portage/package.accept_keywords and, more importantly, finding out about these packages, you would need to retry installing over and over again until you have met and resolved all errors.

A more automated way of dealing with this is to use **autounmask**. **autounmask** allows you to select the main package you want, and it will automatically create a file inside /etc/portage/package.accept_keywords that unmasks all depending packages as well.

```
# emerge autounmask
# autounmask x11-misc/googleearth
```

Switching System Profiles

Gentoo Linux uses profiles to define defaults for your system environment. You should have gotten a first glance at profiles while installing Gentoo (if you didn't, that's okay). As a quick recap: profiles are a set of predefined default values made by the Gentoo project to destinguish between particular uses of the distribution. Whereas other distributions ask you to select a particular type of deployment (say desktop or server) in the beginning of, or during the initial installation, Gentoo allows you to switch profiles whenever you like.

The profile a system uses is defined by the /etc/make.profile symbolic link:

```
# ls -l /etc/make.profile
lrwxrwxrwx 1 root root 57 Dec 2007 make.profile ->
  /usr/portage/profiles/default-linux/x86/20O7.0/desktop
```

A Gentoo profile defines, amongst other things:

- what USE flags are activated by default
- what applications get installed by default when you select a virtual[1]
- what packages are not available for the profile (for instance, a uclibc profile doesn't support glibc) or should have a certain version (for instance, glibc higher than or equal to 2.4).

Users can easily switch between profiles. A useful tool for this is eselect:

```
# eselect profile list
Available profile symlink targets:
  [1]   default/linux/amd64/13.0 *
  [2]   default/linux/amd64/13.0/selinux
  [3]   default/linux/amd64/13.0/desktop
  [4]   default/linux/amd64/13.0/desktop/gnome
  [5]   default/linux/amd64/13.0/desktop/kde
```

```
  [6]   default/linux/amd64/13.0/developer
  [7]   default/linux/amd64/13.0/no-multilib
  [8]   default/linux/amd64/13.0/x32
  [9]   hardened/linux/amd64
  [10]  hardened/linux/amd64/selinux
  [11]  hardened/linux/amd64/no-multilib
  [12]  hardened/linux/amd64/no-multilib/selinux
  [13]  hardened/linux/uclibc/amd64

# eselect profile set 10
```

As you can see from the profile names, one profile can be a child of another; if this is the case (for instance, profile number 4 from the above list has profile number 3 as parent), the profile inherits the settings from its parent and may extend or override particular settings itself.

Using Third Party Software Repositories

Next to Gentoo's main Portage tree, you can also select third party software repositories. Those are trees of developers, users and organisations that offer ebuilds for you to use which aren't available in Gentoo's Portage tree (yet).

A great tool for managing third party software repositories is **layman**. Install **layman** and then see what repositories layman supports:

```
# layman -L
(...)
xen          [Subversion] (source: http://overlays....)
```

The output of this layman command shows you what third party repositories are configured for use with layman. If you want to select one, add it to your system:

```
# layman -a xen
```

Likewise, to stop using a repository, remove it with **layman -d**. If you want to see what repositories you are currently using, use **layman -l**.

When Things Go Wrong

Of course, both Gentoo and the upstream projects that Gentoo relies on are still ran and developed by humans, and humans are programmed to make occasional mistakes, so things can go wrong. If you have installation or build failures, the following sections might help you troubleshoot it.

Portage Refuses To Install A Package

During package installation analysis, a few issues can occur which need intervention. Those are:

- Blocked packages (a package that needs to be installed is blocked by an already installed package or another package on the installation list)
- Masked packages (a package is not available for your profile or environment)
- Download failures (one or more files needed to install the software aren't available or might be corrupt)
- Kernel configuration mismatch (the package requires particular kernel configuration directives to be set)
- Other issues (catch-all)

Blocked Packages

Sometimes a package offers a functionality that another package also offers and where the two packages cannot coexist with each other. When this occurs, you get the following two error:

```
[blocks B     ] mail-mta/ssmtp (is blocking mail-mta/postfix-2.2.2-r1)
```

In this example, the postfix package is on the installation list, but cannot coexist with ssmtp (which is already installed). As a result, the installation fails.

To resolve this, the easiest approach is to unmerge (remove) the ssmtp package from the system and retry.

Masked Packages

A package can fail to install if it is masked. A mask on a package means that the package is in a state that it shouldn't be installed on your system. Package states are described earlier in this chapter. Package masks are one of the more common causes of installation failures. There can be plenty of reasons why a package mask is interfering with an installation cycle:

- the end user was instructed through a document or another user to unmask one package, causing a possible failure on the dependency integrity
- a developer wrongfully made a package available which has dependencies on packages that aren't available in this particular state
- the package just doesn't fit the profile that the users system requires
- the end user was instructed to install a particular package which isn't available for general consumption yet

If you have a masked package, read "Package States" carefully and, if necessary, "Unmasking Packages" earlier in this chapter.

Verification of download results in failure

When Gentoo Portage attempts to install software, it will verify the downloaded files with the checksums and file sizes that the Gentoo developer has recorded when he added the software package to the Portage tree. When the downloaded file and the checksum/file size doesn't match, an error is shown:

```
('Filesize does not match recorded size', 97003L, 952)
!!! Fetched file: genpatches-2.6.33-1.base.tar.bz2 VERIFY FAILED!
!!! Reason: Filesize does not match recorded size
!!! Got: 97003
!!! Expected: 952
...
>>> Failed to emerge sys-kernel/gentoo-sources-2.6.33, Log file:

>>> '/var/tmp/portage/sys-kernel/gentoo-sources-2.6.33/temp/build.log'
```

In the above example, the file size as recorded by the developer (952 bytes) is different from the file size of the downloaded file (97003 bytes). As a result, Gentoo Portage refuses to install the software as it might be corrupted, contain malicious code, etc.

Most of the time this occurs when the developer has made a honest mistake (he uploaded a file, found an issue, resolved it in Portage but forgot to upload the modified file). In this case, the problem will most likely be reported to https://bugs.gentoo.org and quickly resolved. Try synchronising the Portage tree after a few hours and retry.

If it isn't, check if the problem has been reported or not. You can also try synchronising the Portage tree from a different Gentoo mirror (change the value of the *SYNC* variable in `/etc/portage/make.conf`) as it might be possible that the rsync server you are synchronising with is corrupt.

Another occurring problem can be that a particular file is missing:

```
!!! Fetch failed for sys-libs/ncurses-5.4-r5, continuing...
(...)
!!! Some fetch errors were encountered.  Please see above for details.
```

If your network connection is up and running, it is possible that the mirror where Portage wants to download the source code from is temporarily down. You can try again later or search for a different mirror (see the Gentoo Handbook for more information). If the file is really missing (i.e. other mirrors are also not hosting it), check https://bugs.gentoo.org to see if this is a known issue.

Kernel configuration mismatch

Some packages require certain kernel configurations. To ensure you don't install the packages without running a kernel with the right configurations, these packages verify your kernel configuration settings. If they find a mismatch, they will report it and abort the installation:

```
>>> Emerging (1 of 1) net-wireless/broadcom-sta-5.60.48.36
...
 * Determining the location of the kernel source code
 * Found kernel source directory:
 *     /usr/src/linux
 * Found kernel object directory:
 *     /lib/modules/2.6.33-rc8/build
 * Found sources for kernel version:
 *     2.6.33-rc8
 * Checking for suitable kernel configuration options...
 *   CONFIG_WIRELESS_EXT:    is not set when it should be.
 *   CONFIG_WEXT_PRIV:    is not set when it should be.
 * Please check to make sure these options are set correctly.:
 *   Incorrect kernel configuration options
```

When such an error occurs, you will need to reconfigure your kernel to contain the appropriate configuration parameters (quick hint: during the kernel configuration process, search for the settings using the / help search system).

Multiple packages within a single package slot

When software is being installed, Portage verifies if all requirements are met. This includes USE requirements that some packages have on others. For instance, kdelibs requires dbus, built with the USE="X" setting. If for some other reason dbus is being pulled in without X (an ebuild requires dbus without X, or you changed your USE flags to disallow X in USE), you might get the following error message:

```
!!! Multiple package instances within a single package slot have been pulled
!!! into the dependency graph, resulting in a slot conflict:

sys-apps/dbus:0

  ('ebuild', '/', 'sys-apps/dbus-1.2.24', 'merge') pulled in by
    >=sys-apps/dbus-1.0.2 required by ('installed', '/', 'x11-libs/qt-dbus-4.6.
    sys-apps/dbus required by ('installed', '/', 'app-misc/strigi-0.7.1', 'nome
    >=sys-apps/dbus-1.1 required by ('installed', '/', 'dev-libs/dbus-glib-0.86

  ('installed', '/', 'sys-apps/dbus-1.2.24', 'nomerge') pulled in by
```

```
    sys-apps/dbus[X] required by ('installed', '/', 'kde-base/kdelibs-4.4.5', '
    (and 3 more)

  Explanation:

    New USE for 'sys-apps/dbus:0' are incorrectly set. In order to solve
    this, adjust USE to satisfy 'sys-apps/dbus[X]'.
```

If this is due to a change in USE flags, undo the change (in the above case, setting USE to "-X" generally was a bad idea). If you need a particular USE setting, you can use /etc/portage/package.use to change the USE settings of one or more packages without affecting the entire system.

If this due to packages with conflicting dependencies, you'll either need to drop one of the packages, or file a bug at https://bugs.gentoo.org and ask for a proper solution.

Dependency required with different USE flags

When a package has a dependency on another package, but requires this dependency to be build with some USE flags which you currently have not set, Portage will give you an appropriate error:

```
emerge: there are no ebuilds built with USE flags to
satisfy ">=x11-libs/qt-sql-4.5.0:4[mysql]".

!!! One of the following packages is required to complete your request:
- x11-libs/qt-sql-4.5.2 (Change USE: +mysql)

(dependency required by "app-office/akonadi-server-1.2.1" [ebuild])
```

In the above case, the package akonadi-server requires qt-sql (as a dependency) to be built with *`USE="mysql"`*.

The solution: either add the requested USE flag to your global USE flags (in `/etc/portage/make.conf`) or set the USE flag specifically for the selected package (in `/etc/portage/package.use`).

Other Issues

A few other issues can come up:

- When a package depends on another package which isn't available, you will get a warning "*there are no ebuilds to satisfy <dependency>*". This shouldn't occur as this means an ebuild has a missing dependency or dependency which isn't available for you. If it does occur, please file a bug at https://bugs.gentoo.org.
- When two packages exist with the same name (but different category), you will get a warning that the name is ambiguous. In this case, Portage will ask you to specify the full name (category + packagename) and give you a list of candidates.

Resolving Build Failures

One of the most troublesome issue is when a package fails to build properly. There can be a plethora on issues that is causing this, so this section is definitely going to be updated frequently. In any case, chances that you're the only one having the issue is very small, so a build failure now can be resolved if you sync and retry (say) a day later...

Catch-all solution: revdep-rebuild

Before you start investigating possible build failures, run **revdep-rebuild** first. This tool will check and see if all library dependencies are still correct on your system. Really, this is a time-saver in case of most build errors. This tool is described more in-depth at Dynamic Linking Inconsistency.

```
# revdep-rebuild
```

Missing .la file

A .la file is a libtool archive file. Its use is becoming obsolete, so this error is most likely going to disappear in the near future. However, if you have an error similar to the following:

```
/bin/sh ../libtool --tag=CC   --mode=link i686-pc-linux-gnu-gcc  -O2 -march=pen
-fomit-frame-pointer -Wall -version-info 2600:1:2600 -export-symbols-regex "^pa
-Wl,-O1 -o libpangocairo-1.0.la -rpath /usr/lib libpangocairo_1_0_la-pangocairo
libpangocairo_1_0_la-pangocairo-font.lo libpangocairo_1_0_la-pangocairo-fontmap
libpangocairo_1_0_la-pangocairo-render.lo  libpangocairo_1_0_la-pangocairo-fcfo
libpangocairo_1_0_la-pangocairo-fcfontmap.lo  libpango-1.0.la -lgobject-2.0 -lg
-lglib-2.0   -lcairo   -lm  libpangoft2-1.0.la -lfreetype   -lfontconfig

grep: /usr/lib/libGL.la: No such file or directory
/bin/sed: can't read /usr/lib/libGL.la: No such file or directory
libtool: link: `/usr/lib/libGL.la' is not a valid libtool archive
make[4]: *** [libpangocairo-1.0.la] Error 1
```

then running the lafilefixer command might help you out:

```
# emerge lafilefixer
# lafilefixer --justfixit
```

Compiler errors

Compiler errors are almost always due to a bug in the package itself, not your system. An example would be:

```
/var/tmp/portage/app-misc/lirc-0.8.6-r2/work/lirc-0.8.6/drivers/lirc_i2c/lirc_i
error: unknown field 'id' specified in initializer
/var/tmp/portage/app-misc/lirc-0.8.6-r2/work/lirc-0.8.6/drivers/lirc_i2c/lirc_i
warning: initialization makes pointer from integer without a cast
make[5]: ***
[/var/tmp/portage/app-misc/lirc-0.8.6-r2/work/lirc-0.8.6/drivers/lirc_i2c/lirc_
Error 1
make[4]: ***
[_module_/var/tmp/portage/app-misc/lirc-0.8.6-r2/work/lirc-0.8.6/drivers/lirc_i
Error 2
```

The important part here is the error statement prefixed by the file name (usually ending in .c or .cpp) and a number (which is the line number in the file where the compiler found an error). Whenever you get such error messages, chances are that the package itself is broken. You should check Gentoo's Bugzilla [https://bugs.gentoo.org] to see if it is a known error (usually is) and if a fix is already available (or a workaround).

Changing Configuration Files

When Portage installs or upgrades software which provides configuration files, the files themselves aren't automatically updated. This is to prevent Portage from overwriting changes you made to the configuration files. However, package upgrades might require configuration file upgrades as well.

Note

The setting that defines which locations are "protected" from automated updates and which aren't is the CONFIG_PROTECT (list of directories or files that are not automatically updated) and CONFIG_PROTECT_MASK (list of directories or files, inside those defined by CONFIG_PROTECT, which are automatically updated).

Both variables are defined by the profile used by your system and expanded by the environment files offered by various applications and, optionally, the setting you gave inside `/etc/portage/make.conf`.

To be able to identify possible configuration file upgrades, Portage warns you when configuration file updates are available. There are a few tools available that help you "merge" these updates with your existing configuration files. It is important to understand their limitations though.

Configuration File Updates

Whenever a configuration file update is available, the file itself is stored on the file system next to the current configuration file. These files (whose file name starts with `.cfg_`) can then be compared with the current configuration file.

There are no tools available that do this automatically for you. Most functionality that the tools provide is a view on the differences and the possibility to merge these changes automatically. In this book I describe the **dispatch-conf** tool. Another tool is etc-update.

dispatch-conf

The **dispatch-conf** tool allows you to view the differences between a configuration file and its update and merge updates automatically if the only differences are comments.

To view the list of updates, launch **dispatch-conf**. It will first perform the automated updates and then show you the differences between the rest of the available updates:

```
--- /etc/conf.d/bootmisc           2007-12-26 14:41:35.000000000 +0100
+++ /etc/conf.d/._cfg0000_bootmisc 2008-02-29 18:40:02.000000000 +0100
@@ -9,4 +9,4 @@
# Should we completely wipe out /tmp or just selectively remove known
# locks / files / etc... ?

-WIPE_TMP="no"
+WIPE_TMP="yes"

>> (1 of 4) -- /etc/conf.d/bootmisc
>> q quit, h help, n next, e edit-new, z zap-new, u use-new
   m merge, t toggle-merge, l look-merge:
```

In this example, the `/etc/conf.d/bootmisc` file has an update where the `WIPE_TMP` variable is set to "yes". If you want to keep the "no" setting, press **z** (zap-new) to keep the current configuration file. If you want to use the "yes" value, press "**u**" (use-new) to use the new configuration file.

Modifying Build Decisions

I have already covered the use of USE flags to help Portage decide which features you want (or don't want) in a package. However, there are more items that Portage needs to decide upon and you can offer your choice inside `/etc/portage/make.conf`, Portage' main configuration file.

Languages

Some packages support several languages. Portage can help you select what languages you want supported on your system through the `LINGUAS` variable. You set the LINGUAS variable in /etc/portage/make.conf with the languages of your choice. In the next example, I select the English (en), Dutch (nl), French (fr) and German (de) languages.

```
~# nano -w /etc/portage/make.conf
```

```
LINGUAS="en nl fr de"
```

For instance, when enabled, LibreOffice.org's ebuild will pull in the language packs for those languages.

Underlyingly, Gentoo's package manager will expand this into USE flags. For instance, "linguas_en".

Video Cards

Some packages might need video card information. Examples are the X11 server but also some games or rendering engines. In these cases, you should set the VIDEO_CARDS variable to your video card architecture. You might want to include several in case your architecture is quite new (you can then select the earlier architectures as well).

```
VIDEO_CARDS="i915 i810 intel vesa"
```

Compiler Directives

When building packages, Portage offers optimization directives to the compiler, telling the compiler for which architecture it should build and what optimizations the compiler can try to run. These optimization flags are set in the `CFLAGS` (C-code) and `CXXFLAGS` (C++-code) variables, again inside `/etc/portage/make.conf`.

I'll start off with my example CFLAGS (CXXFLAGS is the same):

```
CFLAGS="-O2 -march=native -pipe"
CXXFLAGS="${CFLAGS}"
```

Let's start off with the architecture selection: march. With `-march=native`, the compiler is asked to optimize code for the architecture of the current system (say a pentium 3 architecture). Also, because I select `-march=`, it will only run on this architecture (or architectures that are backwards compatible with mine). If you want to optimize code for a specific architecture but want to be able to run the code on any x86 architecture, use `-mcpu=` instead. You can also select a particular architecture, like "core2" or "nocona". A full list of supported architectures for the current GCC (make sure you are using the current one) is available online [http://gcc.gnu.org/onlinedocs/gcc/i386-and-x86_002d64-Options.html].

With `-O2` (*O*ptimizations, not zero) I ask GCC to use a list of optimization flags (-O2 is shorthand for a whole set of optimizations) which are documented in the GCC info manual (**info gcc**, select *Invoking GCC*, *Optimize Options*).

I also ask the compiler to pass on information between various stages of compilation using pipes (`-pipe`) rather than temporary files.

Portage Behaviour

Portage itself also has a lot of features it supports. You can set them through the *FEATURES* variable.

```
FEATURES="sign buildpkg"
```

You can find all possible features in the `make.conf` manual page (**man make.conf**). A small selection of supported features is given next.

- *buildpkg* creates binary packages of every succesful build. These binary packages can be reused, even on other systems (as long as they have the same build decision parameters).
- *collision-protect* ensures that a package doesn't overwrite files during its installation that it doesn't own. You might want to use *protect-owned* instead (this is the default) as it allows overwriting files if they aren't currently owned by any package.

- *nodoc* to tell Portage not to install doc files (/usr/share/doc), with *noinfo* and *noman* having similar functions
- *parallel-fetch* will have Portage download files in the background while it is building

Specific software choices

Some software items (mainly scripting languages) can have multiple versions installed side by side. In Gentoo, users can not only define what versions they want installed, but also for which versions other software should build. For instance, applications that provide Python bindings can be configured to provide Python bindings for Python version 2.7 and 3.2 (if supported by the application of course).

These software choices are provided by their respective variables:

```
PYTHON_TARGETS="python2_7 python3_2"
RUBY_TARGETS="ruby18 ruby19"
```

Underlyingly, Gentoo's package manager will expand these variables into USE flags (like "python_targets_python2_7" or "ruby_targets_ruby18").

Fixing Postinstall Problems

Even when software installation has finished successfully, you can still get into trouble.

Dynamic Linking Inconsistency

When Portage (prior to 2.2) upgrades a library, it removes the old shared libraries from the system. As a result, some applications who rely on these libraries might fail with a message like the following:

```
error while loading shared libraries: libcom_err.so.2: cannot
  open object file: no such file or directory
```

If this is the case, you will need to run **revdep-rebuild** to fix this. **revdep-rebuild** is a tool that scans your system for applications that rely on (removed) libraries and will rebuild those packages so that they get linked with the new libraries.

Exercises

1. Portage is not the only package manager that works on Gentoo. There are at least two others. Look up what these packages are and where they differ with Portage.
2. USE flags can also be enabled as an environment variable during your emerge process itself, like so:

   ```
   ~# USE="-ssl" emerge irssi
   ```

   Try to find out where Portage reads all possible USE flags from (you know `make.conf` and environment variable now, but there are others) and in what order.
3. Try to find information regarding Gentoo's architecture testing program.

Chapter 10. User Management

Introduction

Linux is a multi-user operating system. Even systems that will be used by a single user are configured as a multi-user system. This has various advantages:

- security-wise, your system is protected from malicious software execution as the software is executed as an unprivileged user rather than the system administrator
- if at any time multiple users are going to work on the system, you just need to add the user to the system (no need to upgrade to a multi-user environment first)
- you can easily back up all files belonging to a particular user as all user files are located inside his home directory
- if you messed up your personal configuration for a particular software title, you can just remove the configuration files (or move them aside) and start the software title up again to start with a clean slate. No configuration changes made by a user are propagated system-wide

How you deal with this multi-user environment depends on your needs...

Adding or Removing Users

If your system is used by various users, you will need to add or remove their user accounts. Before starting with the command syntax, first a few words on how this information is stored on the Linux system.

User Account Information

A user is identified by his user id, which is an ordinary integer number. However, it is much easier to use a user name instead of a number. For this purpose, a Unix/Linux system maps a user name to a user id. By default, this information is stored within the `/etc/passwd` file. However, you can also configure your system to obtain this information from a central repository (like an LDAP service), similar to how Windows can be configured to connect to an Active Directory.

The passwd file

The `passwd` file contains a line for every user. Each line contains 7 fields, separated by colons:

1. Username
2. Password, or "x" if the password is stored elsewhere
3. User ID
4. Primary group ID
5. Comment or description
6. Home directory
7. Default shell

The *password field* on modern systems contains an "x", telling the system that the password is stored inside the `/etc/shadow` file. Storing the passwords elsewhere is needed to improve the security of the system: the `passwd` file should be world readable because many tools rely on it. Storing the

password (even when it is encrypted or hashed) in a publicly readable file is asking for troubles - tools exist that attempt to crack user account passwords given the encrypted / hashed password values.

For this reason, the hashed password is stored inside the `/etc/shadow` file which is only readable by the root user (system administrator). The tools that work with passwords are small in number and highly audited to decrease the chance that they contain any vulnerabilities. More about the shadow file later...

As you will see in the next section, a user can be a member of many groups. However, every user has a single, *primary group*: this is the active group at the moment that the user is logged on. The active group defines the group ownership of the resources the user creates while logged on (remember, resources are assigned three ownership groups: user, group and others).

The users' *home directory* is usually the directory where the user has full write access to (even more, it is most often the *only* directory where the user has write access to). If a user is logged on through the command line (not graphically), it is also the directory where the user starts to work from.

Finally, the default *shell* for this particular user is defined. We've talked about what a shell is before. Unix/Linux has several shells, each shell provides roughly the same functionality, but is manipulated differently. Gentoo Linux by default uses the bash shell (bourne again shell), a powerfull shell with lots of additional functions such as command autocompletion, output colouring and more. Smaller shells also exist, such as csh (c shell) or ksh (korn shell).

More information about shells is available online.

The shadow file

The `shadow` file contains all information regarding a users' password. The most important field for many is the (hashed) password itself, but other information is available as well. The shadow file, like the passwd file, has a single line for every user; fields are separated by colons.

1. Username
2. Hashed password value
3. Date of last password change (counted in days since Jan 1, 1970)
4. Number of days that need to pass before the password can be changed by the user
5. Maximum number of days since the password change that the password can be used; after this amount of days, the password will need to be changed by the user
6. Number of days before expiry date (see field 5) that the user will be warned about the pending password change policy
7. If the password isn't changed after this many days after the forced password change, the account is locked
8. Date when the account is (or will be) locked (counted in days since Jan 1, 1970)
9. Reserved field (not used)

If the last three fields are left empty (which is the default case), their enforcement isn't valid.

The password value is *hashed*, meaning that the password itself is not stored on the disk (nor in any encrypted form). Instead, a mathematical formula is used to create a unique number or string from a password. To verify if a password given by a user matches, the given password is passed through the same mathematical formula and the resulting number or string is matched against the stored string. Such method makes it harder for a user to find out the password even if he has access to the shadow file because he can't deduce the password from the hash value.

Other account storage: nsswitch.conf

Account information can be stored elsewhere - any repository will do, as long as it provides at least the same information as the passwd (and shadow) file. This is important because in enterprise environments, you rather want to keep track of user accounts in a central repository rather than in the files on several hundreds of systems.

The `/etc/nsswitch.conf` file defines where the system can find this information. An excerpt from an nsswitch.conf file is given below. You notice that it defines services on every line followed by the repository (or repositories) that manages the information.

```
passwd: compat
shadow: compat
group: compat
hosts: files dns
```

In the example, the passwd, shadow and group services are managed by the "compat" implementation. Compat is the default implementation provided by glibc (GNU C Library) which offers access to the various `/etc/*` files. The hosts service (used to resolve host names to IP addresses and vice versa) is managed by two implementations:

1. "files", which is the implementation that offers access to the /etc/hosts file (a table containing IP address and host name(s))

2. "dns", which is the implementation that offers queries with DNS servers

Group Information

Group membership is used to group different users who need access to a shared resource: assign the resource to a particular group and add the users to this group.

The /etc/group file

Similar with the /etc/passwd file, group information is stored inside the /etc/group. Again, every line in this text file defines a group; the fields within a group definition are separated by a colon.

1. Group name

2. Group password, or "x" if the password is stored elsewhere

3. Group ID

4. Group members (who don't have the group as a primary group)

It might seem strange to have a password on a group. After all, a user logs on using his user name. However, there is a sane reason for this: you can add users to a different group and password-protect this group. If a user is logged on to the system (but doesn't use the particular group as primary group) and leaves his terminal, malicious users can't change to this particular group without knowing the password even if they have access to the users' terminal (and therefore logged on session).

Group passwords aren't used often though. The cases where group passwords can be used (privileged groups) are usually implemented differently (for instance using privilege escalation tools like sudo).

Creating or Deleting Users

The useradd command

If you want to add a user to the system, you can use the **useradd** command (you'll need to be root to perform this action):

```
# useradd -D thomas
```

In the above example, a user account identified by "thomas" is created using the system default settings (which, for a Gentoo Linux system, means that the default shell is bash, the home directory is /home/thomas, etc) after which his password is set.

You can pass on additional arguments to the useradd command to alter the users' attributes (such as the user id, home directory, primary group ...). I encourage you to read the useradd manual page for more information.

The userdel command

If a user account needs to be removed from the system, you can use the **userdel** command.

```
# userdel -r thomas
```

With the **-r** option, **userdel** not only removes the user account from the system but also cleans and removes the users' home directory. If you omit this option, the users' home directory remains available on the system, allowing you to keep his (private or not) files for future use.

The usermod command

To manipulate an existing account, you can use the **usermod** command. For instance, to modify the primary group of the thomas account to the "localusers" group:

```
# usermod -g localusers thomas
```

Adding or Removing Users to/from Groups

Once a user account is created, you can't use **useradd** to add the user to one or more groups.

Creating or Deleting Groups

First of all, if a group doesn't exist yet, you'll need to create it: the **groupadd** command does this for you. Similarly, to remove a group from the system, you can use **groupdel**.

Warning

You will be able to remove groups even though there are still users member of this group. The only check that groupdel performs is to see if a group is a users' primary group (in which case the operation fails).

```
# groupadd audio
```

Manipulating Group Membership

Suppose you want to add or remove a user from a group, you can use the **usermod** tool (as seen before) or the **gpasswd** tool.

The **gpasswd** tool is the main tool used to manipulate the group file. For instance, to add a user to a particular group (in the example the "audio" group):

```
# gpasswd -a audio thomas
```

Most resources on a Unix system are protected by a particular group: you need to be a member of a particular group in order to access those resources. The following tables gives an overview of interesting groups.

Table 10.1. Incomplete (!) list of system groups

Group name	Description / resources
wheel	Be able to "**su -**" to switch to the root user
audio	Be able to use the sound card on the system
video	Be able to use the graphical card for hardware rendering purposes (not needed for plain 2D operations)
cron	Be able to use the system scheduler (cron)
cdrom	Be able to mount a CD/DVD

Setting and Changing Passwords

The **passwd** command allows you to change or set an accounts' password.

```
# passwd thomas
New UNIX password: (enter thomas' password)
Retype new UNIX password: (re-enter thomas' password)
passwd: password updated succesfully
```

The root user is always able to alter a users' password. If a user wants to change his own password, the passwd command will first ask the user to enter his current password (to make sure it is the user and not someone who took the users' session in the users' absence) before prompting to enter the new password.

With the tool, you can also immediately expire the users' password (**-e**), lock or unlock the account (**-l** or **-u**) and more. In effect, this tool allows you to manipulate the `/etc/shadow` file.

Elevating User Privileges

On any system, a regular user has little to no rights to perform administrative tasks. However, on a home workstation you'd probably want to be able to shut down the system. You can log on as the root user on a different (virtual) terminal, but you can also elevate your own privileges...

Switching User

With the **su** command you can switch your user identity in the selected session.

```
$ su -
Password: (Enter the root password)
#
```

In the above example, a regular user has switched his session to become a root session. The "**-**" argument informs the su command that not only the users' privileges should be switched, but also that the root users' environment should be loaded. Without the "**-**" option, the regular users' environment would be used.

This environment defines the shell behaviour; its most important setting is the PATH variable which defines where the binaries are located for the commands that this user might summon.

With su, you can also switch to a different user:

```
$ su thomas -
Password: (Enter thomas' password)
$
```

If you just want to execute a single command as a different user, you can use the "-c" argument:

```
$ su -c "shutdown -h now"
```

Assigning Specific Privileged Commands

The su-based methods require the user to know the password of the other accounts. On many systems, you might not want this. There are two ways of dealing with such situations: marking a command so that it always runs as a privileged user, or use a tool that elevates privileges without requiring the password for the elevated privilege...

Marking Commands for Elevated Execution

Executable binaries (not shell scripts) can be marked so that the Unix/Linux kernel executes that command as a specific user, regardless of who started the command. This mark is the *setuid* bit. Once set (using the chmod command), the tool is always executed with the rights of the owner and not the rights of the executor:

```
# chmod +s /path/to/command
```

Warning

Using setuid tools is generally considered a security risk. It is better to avoid setuid tools when possible and use tools such as sudo, as explained later.

For instance, if the shutdown command is marked setuid, then every user is able to run the shutdown command as root (which is the commands' owner) and thus be able to shut down or reboot the system.

Using sudo

If you mark an executable using the setuid bit, every user can execute the command as the application owner (root). You usually don't want to allow this but rather assign the necessary rights on a per-user, per-command basis. Enter sudo.

The **sudo** tool allows the system administrator to grant a set of users (individually or through groups) the rights to execute one or more commands as a different user (such as root), with or without requiring their password (for the same reason as the passwd command which asks the users' password before continuing).

Once available, the system administrator can run the **visudo** command to edit the configuration file. In the next example, the following definitions are set:

- All users in the wheel group are allowed to execute any command as root
- All users in the operator group are allowed to shut down the system
- The test user is allowed to run a script called webctl.ksh without a password
- All users in the httpd group are allowed to edit the /etc/apache2/conf/httpd.conf file through sudoedit

```
%wheel    ALL=(ALL) ALL
%operator ALL=/sbin/shutdown
test      ALL=NOPASSWD: /usr/local/bin/webctl.ksh
%httpd    ALL=(ALL) sudoedit /etc/apache2/conf/httpd.conf
```

If sudo is set up, users can execute commands by prepending **sudo** to it. If allowed, some users can even obtain a root shell through the **sudo -i** command.

```
(Execute a single command as root)
$ sudo mount /media/usb
Enter password: (unless configured with NOPASSWD)
```

```
(Obtain a root shell)
$ sudo -i
Enter password: (unless configured with NOPASSWD)
#
```

Exercises

1. When invoking commands using sudo, sudo logs every attempt (including user name, working directory and command itself). Where is this log?

Chapter 11. Network Management

Introduction

An important aspect of system management is networking configuration. Linux is a very powerful operating system with major networking capabilities. Even more, many network appliances are in fact Linux-based.

There are two configurations you'll most likely get in contact with: wired network configuration (of which I'll discuss the Ethernet connection) and wireless (IEEE 802.11* standards).

Supporting your Network Card

Native Driver Support

PCI Cards

First of all, check how many interfaces you would expect on your system. Verify this with the PCI devices found by Linux. For instance, to find out about a wired network controller ("Ethernet" controller):

```
# lspci | grep Ethernet
06:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd.
        RTL-8169 Gigabit Ethernet (rev 10)
```

In this case, one network card was found that offered Ethernet capabilities. The card uses the Realtek 8169 chip set.

USB Network Cards

There are a few USB devices which offer networking capabilities (most of them wireless) which have native Linux support. An example are the USB devices with the Intel 4965agn chip set. If your Linux kernel supports it, the moment you plug it in, a network interface should be made available. For instance, for wireless devices you could use **iwconfig**, for regular Ethernet cards **ifconfig**:

```
# iwconfig
lo        no wireless extensions.

dummy0    no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11g  ESSID:"default"  Nickname:"default"
          Mode:Managed  Frequency:2.412 GHz  Access Point: 00:1D:6A:A2:CD:29
          Bit Rate:54 Mb/s   Tx-Power=20 dBm   Sensitivity=8/0
          Retry limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=89/100  Signal level=-37 dBm  Noise level=-89 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:7
```

Support through Windows Drivers

It is possible to support your (wireless or not) network card using the Windows drivers. The tool you need to install for that is called **ndiswrapper**. First, install ndiswrapper:

```
# emerge ndiswrapper
```

Next, either download the windows drivers for the network card or mount the driver CD that was provided with the card. In the drivers, you should find an .inf file. This file contains information regarding the driver(s) for the card and is used by ndiswrapper to create a wrapper.

Install the driver using **ndiswrapper -i** from within the location where the driver is unpacked:

```
# ndiswrapper -i net8191se.inf
```

To verify if the driver installation succeeded, get an overview of the installed drivers using **ndiswrapper -l**:

```
# ndiswrapper -l

net8191se: driver installed, hardware present
```

As you can see, the driver got installed and detected compatible hardware.

Now have ndiswrapper create the necessary modprobe information (modprobe is used by the system to load kernel modules with the correct information; ndiswrapper creates modprobe information that ensures that, when the ndiswrapper kernel module is loaded, the installed wrapper drivers are enabled as well) and make sure that the ndiswrapper kernel module is started when you boot your system:

```
# ndiswrapper -m
# nano -w /etc/modules.autoload.d/kernel-2.6
(Add "ndiswrapper" on a new line)
```

You can manually load the ndiswrapper kernel module as well:

```
# modprobe ndiswrapper
```

You can now check if the network interface is available (**iwconfig** or **ifconfig**).

Verify your Networking Abilities

To find out if Linux has recognized this interface, run the **ip link** command. It will show you the interfaces that it has recognized on your system:

```
# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:c0:9f:94:6b:f5 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
      qlen 1000
    link/ether 00:12:f0:57:99:37 brd ff:ff:ff:ff:ff:ff
```

Now, to find out which interface maps to the Ethernet controller shown before you'll need to check the Linux kernel output when it detected the interfaces. You can either use **dmesg** (which displays the last few thousands of lines produced by the Linux kernel) or `/var/log/dmesg` (depending on your system logger) which is the log file where all Linux kernel output is stored for the duration of the systems' session (i.e. until the next reboot).

```
# grep -i eth0 /var/log/dmesg
eth0: RTL8169sb/8110sb at 0xf8826000, 00:c0:9f:94:6b:f5, XID 10000000
  IRQ 11
```

In this case, the eth0 interface indeed maps to the Ethernet controller found before.

If Linux does not recognize your device, you'll need to reconfigure your Linux kernel to include support for your network driver. The Linux kernel configuration has been discussed before as part of the device management chapter.

Wired Network Configuration

Most systems have support for the popular Ethernet network connection. I assume that you are already familiar with the term Ethernet and the TCP/IP basics.

Before you configure Gentoo Linux to support your Ethernet connection, you'll first need to make sure that your network card is supported. Once available, you'll configure your interface to either use a manually set IP address or automatically obtain an IP address.

Configuring the Wired Network

There are two methods you can use to configure your wired network: a manual approach (which works on all Linux systems) or the Gentoo Linux specific approach.

Manual Configuration

The quickest method for configuring your network is to tell Linux what you want - a static IP address for your interface, or automatically obtain the IP address information from a DHCP server which is running on your network (most Internet sharing tools or appliances include DHCP functionality).

To set the static IP address 192.168.0.100 to the eth0 interface, telling Linux that the gateway on the network is reachable through 192.168.0.1 (the IP address that shares access to outside networks):

```
# ifconfig eth0 192.168.0.100 netmask 255.255.255.0
 broadcast 192.168.0.255 up
# ip route add default via 192.168.0.1
```

In the example, I used the **ifconfig** command to tell Linux to assign the IP address 192.168.0.100 to the eth0 interface, setting the netmask (part of the IP address that denotes the network) to 255.255.255.0 and broadcast (IP address which addresses all IP addresses in the local network) to 192.168.0.255. This is the same as assigning the IP address on a 192.168.0.1/24 network (for those who understand the CIDR notation).

If you need static IP addresses but don't know the netmask (and broadcast), please ask your network administrator - these are quite basic settings necessary for an IP configuration.

You'll most likely also receive a set of IP addresses which correspond to the DNS servers (name servers) for your network. You'll need to set those IP addresses inside the `/etc/resolv.conf` file:

```
# nano /etc/resolv.conf
```

```
search lan
nameserver 10.2.3.4
nameserver 10.2.3.5
```

With this configuration file you tell Linux that a host name can be resolved through the DNS services at the corresponding IP addresses (the name servers) if it does not know the IP address itself.

If you want to configure eth0 to automatically obtain its IP address (and default gateway and even DNS servers), which is the most popular method for local network configurations, you can use a DHCP client such as **dhcpcd**:

```
# dhcpcd eth0
```

That's all there is to it (unless the command fails of course ;-)

Gentoo Linux Network Configuration

If you want to have Gentoo Linux configure your network device, you'll need to edit the /etc/conf.d/net file.

```
# nano /etc/conf.d/net
```

If you need to set the IP address yourself (static IP address), you'll need to set the following (suppose the static IP address is 192.168.0.100, gateway 192.168.0.1 and netmask 255.255.255.0 and the name servers are 10.2.3.4 and 10.2.3.5):

```
config_eth0="192.168.0.100 netmask 255.255.255.0"
dns_servers_eth0="10.2.3.4 10.2.3.5"
```

If you want to configure the interface to use DHCP (automatically obtain IP address):

```
config_eth0="dhcp"
```

For more examples on the Gentoo Linux network configuration (with more advanced features), check out the `/usr/share/doc/openrc-*/net.example` file.

To enable this support, you need to add the net.eth0 service to the default runlevel and start the net.eth0 service.

```
# rc-update add net.eth0 default
# /etc/init.d/net.eth0 start
```

If a command tells you that net.eth0 doesn't exist, create it as a symbolic link to the net.lo service script:

```
# cd /etc/init.d; ln -s net.lo net.eth0
```

More about services later.

Wireless Network Configuration

Wireless networking support is actively being developed on Linux. Sadly, it is also one of the regions where a fully automated out-of-the-box solution isn't available yet. Linux is lacking this because the card providers themselves do not follow standards or refuse to help out with (free software) driver development. As a result, wireless card support (drivers) can be triggered through free software drivers (if you're lucky), proprietary Linux drivers (if you're somewhat lucky) or proprietary Windows drivers (if you're not lucky, but will still be able to get your card working). A fourth state can be that you just ... won't ... get ... it ... working. Yet.

However, development of wireless card support is - like I said - actively being developed. Chances are that an unsupported card (or chip set) now will be supported within 6 months.

Generally speaking though, 80% to 90% of the wireless cards/chip sets are supported under Linux.

Supporting your Network Card

If you have configured your kernel with support for your wireless network card, you should be able to find the interface in the **ifconfig -a** output:

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr c8:0a:a9:42:9d:76
          inet addr:192.168.20.2  Bcast:192.168.20.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:30 Base address:0x6000

eth1      Link encap:Ethernet  HWaddr f0:7b:cb:0f:5a:3b
          inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:510358 errors:0 dropped:0 overruns:0 frame:13407
          TX packets:300167 errors:5 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:732540912 (698.6 MiB)  TX bytes:26679459 (25.4 MiB)
          Interrupt:16
```

In the above example, two Ethernet interfaces are detected: eth0 (which in my case is a regular Ethernet interface) and eth1 (which, since I only have a single wired interface on my system, is most likely the wireless card). To be absolutely sure about the wireless capabilities, you'll need to install wireless-tools or iw.

Using Wireless Extensions Support (wireless-tools)

The (old, yet still working) wireless extensions support tool set is slowly being deprecated in favour of the new tool set. However, you might be required to use the old set as the switch requires the wireless card drivers to be rewritten as well. Especially with proprietary drivers this might take a while, so support for wireless-tools is not going to go away soon.

The information in this section will help you configure a wireless card/network using command-line tools. For a more user-friendly approach, please read User-friendly Network Configuration Tools.

Verifying Wireless Capabilities

To verify if a particular Ethernet interface really has wireless capabilities, first install wireless-tools and then run **iwconfig**:

```
# emerge wireless-tools
# iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

eth1      IEEE 802.11bgn  ESSID:"1de_verdiep"  Nickname:""
          Mode:Managed  Frequency:2.462 GHz  Access Point: 02:26:5A:4B:E4:6A
          Bit Rate=54 Mb/s   Tx-Power:24 dBm
          Retry min limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Managementmode:All packets received
          Link Quality=5/5  Signal level=-48 dBm  Noise level=-94 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:32  Invalid misc:0   Missed beacon:0
```

As I already suspected, eth1 is indeed the wireless interface.

Accessing a Wireless Network

To access an existing wireless network, you need a few settings. Some of them can be obtained quickly, others might require information from your network administrator.

Let's first start with the wireless network name, called the ESSID. With **iwlist** you can obtain a list of detected wireless networks and their accompanying ESSIDs:

```
# iwlist eth1 scan
eth1      Scan completed :
          Cell 01 - Address: 00:11:0A:2A:73:03
                    ESSID:"aaa"
                    Protocol:IEEE 802.11bg
                    Mode:Master
                    Frequency:2.417 GHz (Channel 2)
                    Encryption key:off
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 9 Mb/s; 11 Mb/s
                              6 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                              48 Mb/s; 54 Mb/s
                    Quality=82/100  Signal level=-48 dBm
                    Extra: Last beacon: 37ms ago
          Cell 02 - Address: 00:C0:49:B0:37:43
                    ESSID:"USR8022"
                    Protocol:IEEE 802.11b
                    Mode:Master
                    Frequency:2.462 GHz (Channel 11)
                    Encryption key:on
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 22 Mb/s
                    Quality=41/100  Signal level=-76 dBm
                    Extra: Last beacon: 7665ms ago
```

In this case, two wireless networks are found. The first one has ESSID "aaa" and does not require any encryption (so you don't need to know any password or passphraze to access this network) - notice the "Encryption key:off" setting. The second one has ESSID USR8022 and requires an encryption key. However, the second network's signal is also less powerful (lower quality and signal level).

To configure your card to use a particular ESSID, you can use the iwconfig command:

```
# iwconfig eth1 essid aaa
```

Suppose that you need to enter an encryption key as well, you can add the key either in its hexadecimal form, or through the ASCII representation.

```
# iwconfig eth1 essid USR8022 key FF83-D9B3-58C4-200F-ADEA-DBEE-F3
# iwconfig eth1 essid USR8022 key s:MyPassPhraze
```

Once you have attached your wireless interface to a particular network, you can configure it as if it was a fixed Ethernet interface.

Now, Gentoo Linux allows you to configure your wireless network card through `/etc/conf.d/net` as well.

In the next example, the wireless configuration is set so that the two networks (aaa and USR8022) are supported where aaa is the preferred network.

```
modules="iwconfig"
key_aaa="key off"
key_USR8022="s:MyPassPhraze enc open"
preferred_aps="'aaa' 'USR8022'"
```

Once your wireless interface is connected to a wireless network, you can use the IP configuration commands as shown earlier for wired networks.

Again, you'll need to add the net.eth1 service to the default runlevel and then fire up the net.eth1 service:

```
# rc-update add net.eth1 default
# /etc/init.d/net.eth1 start
```

Using the New Wireless Extensions Support (iw)

The new wireless extensions support requires kernel drivers that use the (new) nl80211 netlink interface. Almost all free software wireless drivers have been ported towards this interface, so if your wireless card is by default supported by the Linux kernel, you will most likely want to use the iw tool set.

Verifying Wireless Capabilities

To verify if a particular Ethernet interface really has wireless capabilities, first install iw and then run **iw list**:

```
# emerge iw
# iw list
lWiphy phy0
        Band 1:
                Frequencies:
                        * 2412 MHz [1] (20.0 dBm)
                        * 2417 MHz [2] (20.0 dBm)
...
                        * 2484 MHz [14] (20.0 dBm) (passive scanning, no IBSS)
                Bitrates (non-HT):
                        * 1.0 Mbps
                        * 2.0 Mbps (short preamble supported)
...
                         * 54.0 Mbps
        max # scan SSIDs: 1
        Supported interface modes:
                 * IBSS
                 * managed
```

Unlike wireless-tools, iw lists the device as being phy0 (so no immediate relation with eth0/eth1). The relation can be found using **iw dev**:

```
# iw dev
phy#0
        Interface eth1
                ifindex 4
                type managed
```

Accessing a Wireless Network

To access an existing wireless network, you need a few settings. Some of them can be obtained quickly, others might require information from your network administrator.

Let's first start with the wireless network name, called the ESSID. With **iw scan** you can obtain a list of detected wireless networks and their accompanying ESSIDs:

```
# iw dev eth1 scan

BSS 02:87:11:26:39:f9 (on eth1)
        TSF: 130175283584 usec (1d, 12:09:35)
        freq: 2432
        beacon interval: 100
        capability: ESS Privacy ShortSlotTime (0x0411)
        signal: 61.00 dBm
        last seen: 930 ms ago
```

```
        SSID: TM2300
        Supported rates: 1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0
        DS Parameter set: channel 5
        ERP: Barker_Preamble_Mode
        Extended supported rates: 24.0 36.0 48.0 54.0
        RSN:     * Version: 1
                 * Group cipher: CCMP
                 * Pairwise ciphers: CCMP
                 * Authentication suites: PSK
                 * Capabilities: (0x0000)
BSS 00:1a:70:eb:ae:f4 (on eth1)
        TSF: 606247219588 usec (7d, 00:24:07)
        freq: 2437
        beacon interval: 100
        capability: ESS ShortSlotTime (0x0401)
        signal: 72.00 dBm
        last seen: 870 ms ago
        SSID: linksys
        Supported rates: 1.0* 2.0* 5.5* 11.0* 18.0 24.0 36.0 54.0
        DS Parameter set: channel 6
        ERP: <no flags>
        Extended supported rates: 6.0 9.0 12.0 48.0
```

In this case, two wireless networks are found. The first one has ESSID "TM2300" and requires WPA encryption (this can be deduced from the RSN information). The second network has SSID "linksys" and does not require encryption.

To configure your card to use a particular *non-WPA encrypted* ESSID, you can use the **iw connect** command:

```
# iw eth1 connect linksys
```

Suppose that you need to enter a WEP encryption key as well, you can add the key either in its hexadecimal form, or through the ASCII representation.

```
# iw eth1 connect myssid keys d:0:FF83D9B358C4200FE8343033
# iw eth1 connect myssid keys 0:MyPrivatePassword
```

To verify that the connection succeeded, request the link status using **iw link**:

```
# iw dev eth1 link
Connected to 68:7f:74:3b:b0:01 (on eth1)
        SSID: linksys
        freq: 5745
        RX: 30206 bytes (201 packets)
        TX: 4084 bytes (23 packets)
        signal: -31 dBm
        tx bitrate: 300.0 MBit/s MCS 15 40Mhz short GI
```

Once you have attached your wireless interface to a particular network, you can use the IP configuration commands as shown earlier for wired networks.

Using wpa_supplicant for WPA Encrypted Networks

The wpa_supplicant tool is a software component which controls the wireless connection between your system and an access point. A major advantage of **wpa_supplicant** over the previously described wireless tools is its support for WPA/WPA2.

Before you can use wpa_supplicant, you first need to install it:

```
# emerge -a wpa_supplicant
```

Accessing a Wireless Network

You need to configure your wpa_supplicant to support the wireless network(s) you want to access. Suppose that your home network is called "home" and is a secured (WPA) environment with key "myHomeKey" and at your work there is a wireless network called "CompanyGuests", secured (WPA) environment with key "myCompanyKey" and a third network at your local computer club called "hobby", not secured, then the following `wpa_supplicant.conf` configuration could work:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel

network={
  ssid="home"
  psk="myHomeKey"
}

network={
  ssid="CompanyGuests"
  psk="myCompanyKey"
}

network={
  ssid="hobby"
  key_mgmt=NONE
}
```

The **wpa_supplicant** tool also supports WPA2. For instance:

```
network={
  ssid="akkerdjie"
  proto=WPA2
  psk="highly private key"
}
```

If you do not like to see your private key in plain text, use **wpa_passphraze** to encrypt your key:

```
$ wpa_passphraze akkerdjie "highly private key"
network={
  ssid="akkerdjie"
  #psk="highly private key"   <-- Plain comment, can be removed!
  psk=cbcb52ca4577c8c05b05e84bdd2ef72f313d3c83da18c9da388570ae3a2a0921
}
```

You can copy/paste the resulting information in `wpa_supplicant.conf` and remove the (commented) plain-text key information.

If your wireless card is found by Linux (and its powered on), then running the following command will activate the wpa_supplicant on top of it (assume the wireless interface is called wlan0):

```
# wpa_supplicant -Dwext -iwlan0 -c/etc/wpa_supplicant.conf
```

One interesting option is the -D option: with this you select the wireless driver to use. With -Dwext, we use the Linux wireless extensions (which is quite generic). In certain cases you might need to use a different driver - the Internet has many resources on how to configure your specific wireless network card with Linux if the Linux wireless extensions don't work.

Of course, once the configuration file is finished, you can use Gentoo's networking scripts as well. First, edit `/etc/conf.d/net` to use wpa_supplicant:

```
modules="wpa_supplicant"
wpa_supplicant_wlan0="-Dwext"
```

To have the wireless support active when you boot up your system, enable the net.wlan0 init script. If /etc/init.d/net.wlan0 doesn't exist yet, first create it:

```
# cd /etc/init.d
# ln -s net.lo net.wlan0
```

Next, add the net.wlan0 init script to the default runlevel:

```
# rc-update add net.wlan0 default
```

User-friendly Network Configuration Tools

The above information should allow you to work with any possible Linux installation. However, the commands might look a bit tricky and, especially with the wireless configuration, might even require you to hop between various commands or windows before you get the connection working.

Luckily, there are other tools around which rely on the same tools as mentioned before, but offer the user a saner interface from which they can configure their network. Note that these do require that the network card is already detected by Linux (so the kernel configuration part should have succeeded).

Wicd

My personal favourite is Wicd, installable through net-misc/wicd. The tool exists out of two parts: a daemon and an end-user configuration interface.

```
# emerge wicd
```

Once installed, add the wicd service to the boot or default runlevel:

```
# rc-update add wicd default
```

Next, make sure Gentoo doesn't start its own network configuration by editing `/etc/rc.conf`, setting the following:

```
rc_hotplug="!net.*"
```

Now, start the wicd service (and shut down the services you are currently using):

```
# /etc/init.d/net.eth1 stop
# /etc/init.d/wicd start
```

If you run inside a graphical environment that supports applets (most desktop environments do), run **wicd-client** (from a "Run Program..." prompt or so). From within a command-line interface, you can use **wicd-curses**. This client will connect with the service and allow you to configure your networks (both wired and wireless) more easily.

I refer you to the Wicd homepage [http://wicd.sourceforge.net] for more information / documentation on the tool.

Firewall Configuration

When your system is going to use the Internet often, using a firewall is encouraged. People generally believe that their operating system is secure out of the box if they don't click on "weird" links inside e-mails or Internet sites. Sadly, this isn't true. Also, Linux should never be seen as a secure operating system - security of a system is completely defined by the competence of the system administrator.

A firewall will not fully protect your system from malicious users on the (Inter)net, but it will filter many - of course, depending on the strength of the firewall.

There are many firewalls available for Linux; on Gentoo Linux alone more than a dozen tools exist (just check out the content of the net-firewall category). Most firewall tools use **iptables** as underlying tool. The iptables tool is an administration tool for manipulating IPv4 packets and is a very known and popular tool.

Firewall tools will often generate iptables rules to create filters (the actual firewall).

Because writing firewall rules is quite custom (it depends on what services your system offers and what tools you often use) I suggest using firewall tools first. Later, when you want to customize them further, you can write your own iptables rules.

Sharing your Internet Connection

We have seen the iptables command previously, as part of the firewall configuration. iptables however is not Linux' firewall tool: its purpose is to create rules on how to deal with network packets on your computer. As such, iptables can also be used to create a NAT gateway through which clients can access the Internet.

In the following examples, we suppose that Internet is available at the wlan0 interface while all clients access through the eth0 interface. Also, we will be assigning IP addresses in the range of 192.168.20.200-192.168.20.250 to our clients...

Forwarding Requests

This is the simplest step: we ask iptables to enable masquerading on the Internet interface. Masquerading keeps track of connections packets going out on this interface with their original source IP address; the packets on the connection are altered so it seems as if the local system has created the connection rather than a client:

```
iptables -A POSTROUTING -t nat -o wlan0 -j MASQUERADE
```

The only remaining tasks here is to enable forwarding packets from the clients to the Internet and back:

```
# iptables -A FORWARD -i eth0 -o wlan0 -s 192.168.20.1/24
  ! -d 192.168.20.1/24 -j ACCEPT
# iptables -A FORWARD -o eth0 -i wlan0 -d 192.168.20.1/24
  ! -s 192.168.20.1/24 -j ACCEPT
```

More information about iptables and masquerading can be found on the Internet...

Distributing IP Addresses

Now, if eth0 is accessible then all clients with a correct IP address attached to the eth0 interface can access the Internet; however, they will manually need to mark the local system as the default gateway as well as defining the necessary DNS servers. Luckily, we can automate this by installing a DHCP server so that clients can automatically obtain their IP address and necessary settings.

There are plenty of DHCP servers around. For local, small use, I myself use dhcp:

```
# emerge dhcp
```

Next, I configure dhcp to distribute the necessary IP address and other settings:

```
# nano -w /etc/dhcp/dhcpd.conf
```

```
option domain-name "siphos.be";
```

```
option domain-name-servers 192.168.2.1;
default-lease-time 600;
max-lease-time 7200;
ddns-update-style none ;

option option-150 code 150 = text ;

subnet 192.168.20.0 netmask 255.255.255.0 {
  range 192.168.20.100 192.168.20.200;
  option routers 192.168.20.1;
}
```

Now that dhcpd is configured, we only need to start it when we need it:

```
# /etc/init.d/dhcpd start
```

Again, if you want to have the script started automatically, add it to the default runlevel.

Allowing Remote Access

If you need to allow remote access to your machine, there are a few tools available. As this book isn't focusing on graphical environments much, I'll stick with SSH access, or *Secure SHell*.

Warning

Allowing remote access to a system is never without security risks. If your security software is not up to date, or your password is easy to guess, or ... you risk being the target for more maliciously minded people. This is especially true if the IP address you have is immediately reachable from the Internet (either directly or because you use port forwarding on your routers).

Secure Shell

By enabling secure shell access to your machine, people on your network who have an account on your system (or know the credentials of an account) can access your system. The tool, which is called **ssh**, encrypts the data that is sent on the network so no-one can eavesdrop on the network and see user names, passwords or even more confidential information flow by.

To enable SSH access to your system, first install the net-misc/openssh package:

```
# emerge openssh
```

Of course, this doesn't automatically enable remote access: you still need to tell your system to start the SSH daemon. You can do this manually using **/etc/init.d/sshd**, but also ask Gentoo to automatically do this for you every time the system boots using **rc-update**.

```
# /etc/init.d/sshd start
# rc-update add sshd default
```

Now that that is accomplished, you (or other users on your network) can access your system using any SSH client (on Windows, I seriously recommend PuTTY [http://www.chiark.greenend.org.uk/~sgtatham/putty/]). For instance, to access your system from another Linux system, the command could look like so (assuming that your IP address is 192.168.2.100 and your user name is "captain"):

```
$ ssh -l captain 192.168.2.100
```

You will be asked to enter captain's password, and then you get a shell just like you would when you log on to the system physically.

Secure File Transfer

By installing and enabling SSH access to your system, you can now also perform secure file transfers.

There are two methods for doing secure file transfer using standard openssh tools: scp and sftp.

Secure Copy

With **scp** (secure copy) you can copy files between systems. If your source or destination (or both) are on a remote system, prepend the source/destination folder with the host name or IP address followed by a colon, like so:

```
$ scp thesis.tar.gz 192.168.2.1:/mnt/usb-stick
```

If the copy also needs to change to a different user (say that you are currently logged on as "bunny" but on the remote side, you only have an account "wolf"):

```
$ scp wolf@192.168.2.2:/usr/portage/distfiles/YAML-0.71.tar.gz .
```

Secure FTP

With **sftp** (secure FTP) you have an ftp-alike tool (which supports the same commands) but which uses the SSH protocol for all data (and command) transfers.

```
$ sftp wolf@192.168.2.2
Connecting to 192.168.2.2...
Password: (enter wolf's password)
sftp> cd /usr/portage/distfiles
sftp> pwd
Remote working directory: /usr/portage/distfiles
sftp> lpwd
Local working directory: /home/bunny
sftp> get YAML-*
Fetching /usr/portage/distfiles/YAML-0.71.tar.gz to YAML-0.71.tar.gz
/usr/portage/distfiles/YAML-0.71.tar.gz  100% 110KB 110.3KB/s 00:00
sftp>
```

Further Resources

- NdisWrapper: The Ultimate Guide [http://www.linuxquestions.org/linux/answers/Networking/NdisWrapper_The_Ultimate_Guide/] on www.linuxquestions.org

Chapter 12. Service Management

Introduction

A *service* is a generic term which can be used in many contexts. Here, a service is a tool that runs in the background (also known as a *daemon*) which offers a certain functionality to the system or to the users. It is also possible that the tool just performs a single set of tasks and then quits.

Examples of services on a Linux system are:

- the logger service, allowing programs on the system to send logging notifications to a global location which is then parsed and processed by a logger tool (example: syslog-ng).
- the clock service, which sets the necessary environmental definitions (like time zone information)
- the SSH service, allowing users to log on to your system remotely (through the secure shell)
- ...

The scripts that manipulate the services are called *init scripts* (initialization scripts) and reside inside `/etc/init.d`. Although this is quite generic for all Linux distributions, Gentoo offers a somewhat different way of working with services, so not all activities mentioned in this chapter can be used for other distributions.

Services at System Boot / Shutdown

When your system boots, the Linux kernel starts a process called **init**. This tool executes a set of tasks defined by the various init levels on the system. Each init level defines a set of services to start (or stop) at this stage.

Within Gentoo, init levels are mapped onto named runlevels.

When init is launched, it will first run the sysinit and bootwait init levels. On Gentoo, the associated runlevels are also called sysinit and boot (sysinit is not configurable). Then, it will start the services for the runlevel it is configured to boot into (by default, init level 3). This init level at Gentoo is mapped onto the "default" runlevel.

For instance, the following services are launched when I start my laptop (sysinit not shown, but sysinit is always launched).

```
# rc-status boot
Runlevel: boot
 alsasound        [ started ]
 bootmisc         [ started ]
 checkfs          [ started ]
 checkroot        [ started ]
 clock            [ started ]
 consolefont      [ started ]
 hostname         [ started ]
 keymaps          [ started ]
 localmount       [ started ]
 modules          [ started ]
 net.lo           [ started ]
 rmnologin        [ started ]
 urandom          [ started ]
# rc-status default
Runlevel: default
```

```
 hald                [ started  ]
 local               [ started  ]
 net.eth0            [ started  ]
 net.eth1            [ stopped  ]
 sshd                [ started  ]
 syslog-ng           [ started  ]
 udev-postmount      [ started  ]
 xdm                 [ started  ]
```

As you can see, all configured services for the two runlevels (boot and default) are launched but one: net.eth1 isn't started (because it is my wireless interface and I'm currently on a cabled network which uses net.eth0).

The init configuration file is called `/etc/inittab`. The next excerpt is not a full `inittab` but explains most important settings:

```
id:3:initdefault:            # The default init level is 3
si::sysinit:/sbin/rc sysinit # sysinit       > run the Gentoo "sysinit" runleve
rc::bootwait:/sbin/rc boot   # bootwait      > run the Gentoo "boot" runlevel
l0:0:wait:/sbin/rc shutdown  # init level 0  > run the Gentoo "shutdown" runlev
l1:S1:wait:/sbin/rc single   # init level S1 > run the Gentoo "single" runlevel
l3:3:wait:/sbin/rc default   # init level 3  > run the Gentoo "default" runleve
l6:6:wait:/sbin/rc reboot    # init level 6  > run the Gentoo "reboot" runlevel
```

Okay, so in the end, init uses Gentoo's runlevels. How do you configure those?

Init Scripts

An init script is a script that manipulates a particular service. It should support the "start" and "stop" arguments as these are used by the **init** tool (actually the **rc** tool which is called by **init**). For instance:

```
# /etc/init.d/udhcp start
# /etc/init.d/syslog-ng stop
```

As you can see, the scripts reside in the `/etc/init.d` directory. These scripts are usually provided by the tools themselves (udhcp and syslog-ng in our examples) but sometimes you might need to write one yourself. Luckily, this is less and less the case.

Next to directly executing the scripts, you can also use the **rc-service** tool:

```
# rc-service syslog-ng start
```

Gentoo Runlevels

Inside `/etc/runlevels`, Gentoo keeps track of the various scripts that need to be started when init starts a specific init level (which maps onto a Gentoo runlevel):

```
# ls /etc/runlevels
boot   default   nonetwork   single
```

Inside the directories you get an overview of the services that should be started when the runlevel is active. For instance, inside the default runlevel one could see:

```
# ls /etc/runlevels/default
local  net.eth0   net.wlan0   syslog-ng     xdm
```

The files found inside these directories are symbolic links, pointing to the associated init script found inside /etc/init.d:

```
# ls -l /etc/runlevels/default/local
```

```
lrwxrwxrwx 1 root root 17 Jul 12   2004
  /etc/runlevels/default/local -> /etc/init.d/local
```

To manipulate the Gentoo runlevels, you can manipulate the symbolic links inside these directories directly, but you can also use the tools rc-update, rc-config and rc-status.

With **rc-update**, you can add or delete links from a particular runlevel. For instance, to remove the xdm init script from the default runlevel:

```
# rc-update del xdm default
```

With **rc-status**, you can see what scripts should be started in the selected runlevel and the current state. The next example shows that the net.eth0 runlevel is not started currently even though it is a service for the default runlevel (the reason is simple: I deactivated it as I don't need the interface currently):

```
# rc-status default
Runlevel: default
 local                                                [started]
 net.eth0                                             [stopped]
 net.wlan0                                            [started]
 syslog-ng                                            [started]
 xdm                                                  [started]
```

With **rc-config**, you can manipulate the runlevels (just like with **rc-update**), show the current status of a particular runlevel (just like with **rc-status**) and view all currently available init scripts and the runlevels in which they are available (actually, **rc-update** can also do this using **rc-update show**):

```
# rc-config list
(...)
```

List of Default Services

When a pristine Gentoo install has finished, you will already have quite a few services available. The following sections give a quick overview of those services and what they stand for.

alsasound

The alsasound service is responsible for loading the appropriate sound kernel modules (if they are known as modules) and saving/restoring the sound configuration at boot-up / shut down.

When the service is started, you might see kernel modules being loaded in memory. However, no other processes are started as part of this service.

bootmisc

The bootmisc service is responsible for various boot-level activities, such as:

- loading the kernel parameters from `/etc/sysctl.conf`.
- cleaning up directories to ensure they don't contain rogue information that might hinder the bootup
- create, if they don't exist, system files with the correct permissions

Once the service has finished starting, no additional processes will be running.

checkfs

The checkfs service is responsible for verifying the integrity of your systems' file systems. By default, it will verify the integrity of the file systems whose last digit in `/etc/fstab` isn't zero. You can force a root file system check by adding the *`forcefsck`* boot parameter or force a full file system

check for all partitions (listed in the fstab file) by creating an empty "/forcefsck" file. This file will be automatically removed once the check has been finished.

```
# touch /forcefsck
```

On the other hand, if you want to ignore the file system checks, create the `/fastboot` file. It too will be automatically removed, this time when the system has booted.

Once the service has finished starting, no additional processes will be running.

checkroot

The checkroot service is responsible for checking the consistency of the root file system. This service uses the same boot parameters (forcefsck or fastboot) as the checkfs service.

The service is also responsible for remounting the root file system read-write (by default it gets mounted read-only by the Linux kernel).

Once the service has finished starting, no additional processes will be running.

clock

The clock service is responsible for setting the system time based on the BIOS clock and the settings defined in `/etc/conf.d/clock`. It will also synchronise the system clock with your hardware clock during shut down.

Once the service has finished starting, no additional processes will be running.

consolefont

The consolefont service is responsible for setting the console font.

Once the service has finished starting, no additional processes will be running.

hald

The hald service is responsible for starting the hardware abstraction layer daemon (see HAL).

Once the service has finished starting, you will find the hald process running as the haldaemon user.

host name

The host name service is responsible for setting your systems' host name based on the input of `/etc/conf.d/hostname`.

Once the service has finished starting, no additional processes will be running.

keymaps

The keymaps service is responsible for setting your keyboard mapping (qwerty, azerty, dvorak, ...) based on the `/etc/conf.d/keymaps` file.

Once the service has finished starting, no additional processes will be running.

local

The local service is responsible for handling the scripts defined in `/etc/local.d`. The scripts, with suffix `.start` or `.stop`, are executed in lexical order (of course during start or stop of the system, depending on the suffix).

The local service is started as the last service before you can log on to your system. It is also the first one to be stopped when the system is shutting down.

As you completely manage what this service does, I can't tell you what will happen when the service has finished starting. By default however, it doesn't do anything.

localmount

The localmount service is responsible for mounting all local file systems (mentioned in `/etc/fstab`). It also initiates the necessary support for USB file systems, specific binary format file systems, security file systems and enabling the swap file system.

Once the service has finished starting, no additional processes will be running.

modules

The modules service is responsible for automatically loading the kernel modules listed in `/etc/modules.autoload`.

Once the service has finished starting, no additional processes will be running.

net.lo (net.*)

The net.lo service is responsible for loading networking support for a specific interface. Although the name suggests that it only supports the lo (loopback) interface, the service actually supports any interface. Other interface scripts are just symbolic links to this script.

Once the service has finished starting, no additional processes will be running.

rmnologin

The rmnologin service is responsible for changing the state of your system from a non-logon-capable system (set by the bootmisc service) to a logon-capable one. This is needed to ensure no-one can log on to your system while important services are being loaded.

Once the service has finished starting, no additional processes will be running.

sshd

The sshd service is responsible for launching the secure shell daemon, which allows you to access your system from a remote location (as long as the network / firewalls permit it) in a secure manner.

Once the service has finished starting, you will find the sshd process running.

syslog-ng (or any other system logger service)

The syslog-ng service is responsible for starting the syslog-ng daemon, which is responsible for watching the `/dev/log` socket for log events and managing those events by dispatching them towards the right log file (or other log server).

Once the service has finished starting, you will find the syslog-ng process running.

udev-postmount

The udev-postmount service is responsible for re-evaluating udev events between the moment udev was started and the moment udev-postmount is started which might have failed for any reason (for instance because not everything was up and running yet).

Once the service has finished starting, no additional processes will be running.

urandom

The urandom service is responsible for initializing the random number generator in a somewhat more secure manner (using a random seed obtained during the last shut down of the system). Without this, the random number generator would be a bit more predictable.

Once the service has finished starting, no additional processes will be running.

Service Configurations

General Service Configuration

Gentoo's general configuration file for the start-up service behaviour is `/etc/rc.conf`.

/etc/rc.conf

Inside the `rc.conf` file, generic settings which are (or might be) needed by several services can be configured. The syntax is, as usual, "key=value". Since openrc (new init system for Gentoo), all settings are bundled in this file. Earlier systems spread the configuration across `/etc/rc.conf` and `/etc/conf.d/rc`. The latter is now deprecated.

The file is pretty well documented.

Specific Service Configuration

Each system service within Gentoo can be configured using a file in `/etc/conf.d` which is named the same as the service itself (except in a few specific cases like network configurations, which use the `/etc/conf.d/net` configuration file). All these files use a key=value syntax for configuration purposes.

For instance, the **/etc/init.d/clock** init script can be configured using the `/etc/conf.d/clock` configuration file.

Softlevel States

Gentoo supports softlevels, which are specific configurations of one or more services. The need exists, because you might create different runlevels (say "work" and "home" instead of just "default") in which services need to be configured differently. As the services would only use their general configuration file, this wouldn't work.

To initiate softlevels, you need to specify "softlevel=<yoursoftlevel>" at the kernel option line (for instance, in GRUB, this means you add it to grub.conf's kernel line). Once set, Gentoo will try to start the softlevel given instead of the default runlevel (coincidentally named "default") and first look for configurations of this softlevel for each service. If it cannot find specific configurations, it will use the default one.

An example use of softlevels would be to define a softlevel "work" and a softlevel "home". Both initiate different settings, such as different networking settings, different clock settings, different crypto-loop settings, etc. This could result in the following two GRUB configuration entries:

```
title=Gentoo Linux @Home
kernel /kernel-3.8.5 root=/dev/sda2 softlevel=home

title=Gentoo Linux @Work
kernel /kernel-3.8.5 root=/dev/sda2 softlevel=work
```

Whenever a service is started (or stopped), it will look for its configuration file called `/etc/conf.d/<servicename>.<softlevel>` (for instance, `/etc/conf.d/clock.work`) and if that doesn't exist, use the default one (for instance, `/etc/conf.d/clock`).

To finish the softlevel, create a new runlevel with the softlevels' name:

```
# mkdir /etc/runlevels/work
```

Finish up by adding the services you need to this runlevel.

Bootlevel States

The idea behind bootlevel is the same as softlevel, but instead of changing the default runlevel "default", you change the default boot runlevel "boot".

Chapter 13. Storage Management

Introduction

Within Linux, the file system the user sees acts regardless of the underlying storage used (hard disk partitions, CD/DVD, removable USB disk, remote file systems). You assign a location of the file system to the storage medium and you're up and running.

I've already covered the `/etc/fstab` file, where you can map these locations and media. In this chapter, I'll cover the various media, how you can create and manage them as well as assign them in the `fstab` file.

Hard Disk Partitions

The most common storage is a hard disk partition. I'll only cover the x86 partitions as those are most used at this moment.

Partition Layout

The x86 partition layout only allows for at most 4 partitions called *primary partitions*. At the time the x86 partition layout was designed, this was thought to be sufficient. However, time has proven the assumption wrong. Luckily, the x86 partition layout has a way of working around this restriction: you can designate one partition as a "container" partition which holds other partitions. This container partition is called the *extended partition*. The partitions inside this extended partition are called *logical partitions*.

Linux gives a number to a partition based on their type: primary partitions (including the extended partition) are numbered 1 to 4; logical partitions start at 5 and can, theoretically, sum up to infinity (if you have an infinite amount of disk space, that is). On Linux, you can create up to 63 partitions per disk.

If you do not need more than 4 partitions, you can stick with the primary partitions. However, if you need more than 4 partitions, make sure you create one extended partition (which holds all disk space) and then create additional logical partitions inside this extended partition.

Partitioning a Disk

To partition a disk, you can use tools such as **fdisk**, **cfdisk**, **sfdisk**, **parted**, ... Of course, there are graphical tools as well (such as **qtparted**). In this section, I'll cover the use of **fdisk** as it is a simple command-line tool (making it easy to add the input/output in this book) which is well supported and fully featured.

In the next paragraphs, I'm going to partition the `/dev/sda` disk (`/dev/sda` is Linux' representation of the first (`a`) SCSI/SATA (`sd`) disk). The second SCSI/SATA disk would be called `/dev/sdb`, third `/dev/sdc`, ...

IDE disks are called `/dev/hda` (b, c, d, ...). Unlike SCSI/SATA disks, IDE disks are labelled based upon their position in the computer. Every IDE controller (which is where the disks are attached to) has a certain position in the computer. The first controller manages the disks a, b, c and d; the second controller manages the disks e, f, g and h, etc. Every controller can manage four disks: the first two are called the primary disks, the last two are called the secondary disks. Every pair of disks has a master (first disk) and slave (second disk). So, the primary master disk of the first IDE controller is `/dev/hda`, the secondary slave disk of the second IDE controller would be `/dev/hdh`.

To find out which disk devices are detected on your system, you can list the contents of `/dev/disk/by-path`:

```
# ls -l /dev/disk/by-path
lrwxrwxrwx 1 root root  9 2009-12-05 23:35 pci-0000:00:1f.1 -> ../../hda
lrwxrwxrwx 1 root root  9 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0 -> ../
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part1
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part2
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part5
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part6
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part7
lrwxrwxrwx 1 root root 10 2009-12-05 23:35 pci-0000:00:1f.2-scsi-0:0:0:0-part8
```

Or, if this location doesn't exist (it is created by Gentoo's udev rules, but that doesn't necessarily mean it is available on other distributions), you can check the content of `/sys/block`:

```
# ls /sys/block
hda  loop0   loop1   loop2   loop3    sda
```

Now, fire up fdisk to edit the partitions on /dev/sda:

```
# fdisk /dev/sda
Command (m for help):
```

To view the current partition layout, enter p:

```
Command (m for help): p

Disk /dev/sda: 240 heads, 63 sectors, 2184 cylinders
Units = cylinders of 15120 * 512 bytes

Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1        14    105808+  83  Linux
/dev/sda2            15        49    264600   82  Linux swap
/dev/sda3            50        70    158760   83  Linux
/dev/sda4            71      2184  15981840    5  Extended
/dev/sda5            71       209   1050808+  83  Linux
/dev/sda6           210       348   1050808+  83  Linux
/dev/sda7           349       626   2101648+  83  Linux
/dev/sda8           627       904   2101648+  83  Linux
/dev/sda9           905      2184   9676768+  83  Linux

Command (m for help):
```

Now, before continuing, let's first think about what partition layout you need...

Partition Layout Scheme

On Linux, as I've told before, the file system can be contained inside one partition, but it can also be made up of several partitions. The reason to use a single partition or multiple partitions depends a bit on how flexible you want to assign your available disk space and what other requirements you have for your file systems.

For instance, if you do not want your users to be able to fill up your root partition you might want to put their home directories on a different partition. In this case, if they end up filling their home directory, only the partition used for the home directories is filled up and the other partitions remain untouched.

Note

This is not the only possible way of limiting users' disk space usage. You can also implement quotas so that users are only allowed to use a particular amount of disk space.

You might want to store certain data on a partition which shouldn't be always visible. In this case, you can opt not to mount the partition automatically.

One final note is that, if your system is not blessed with a huge amount of memory, you need to set up some swap space. Although you can use a swap file for this, most distributions (still) prefer that you use a swap partition. This is a (relatively) small partition which will be used by the Linux kernel to store memory pages when physical memory is full.

An example partitioning scheme is given in Table 13.1, “Example partitioning scheme for a Linux desktop”.

Table 13.1. Example partitioning scheme for a Linux desktop

Partition	Size	Description
/dev/sda1	100Mbyte	Will be used to house the /boot partition in which the bootloader configuration and kernel images are stored. Separate partition as none of the files on this partition are needed during regular operations (thus partition is not automatically mounted)
/dev/sda2	12Gbyte	Main (root) partition which will host the Linux operating system
/dev/sda3	27Gbyte	Partition which will hold the /home files (files and folders of the users of the system).
/dev/sda4	900Mbyte	swap partition. The desktop system will have enough physical memory and the system will not use software suspend (write memory content to disk and hibernate the system) so the swap does not need to be as large as the physical memory.

Cleaning Existing Partitions

Once inside fdisk, you can view the current partitions using the **p** command. You can then remove the partitions you do not want any more with the d command, followed by the partition identifier. For instance, suppose you want to remove the third partition (/dev/sda3):

```
Command (m for help): p
Disk /dev/sda: 240 heads, 63 sectors, 2184 cylinders
Units = cylinders of 15120 * 512 bytes

Device Boot    Start       End    Blocks   Id  System
/dev/sda1             1        14    105808+  83  Linux
/dev/sda2            15        49    264600   82  Linux swap
/dev/sda3            50        70    158760   83  Linux
/dev/sda4            71      2184  15981840    5  Extended
/dev/sda5            71       209   1050808+  83  Linux
/dev/sda6           210       348   1050808+  83  Linux
/dev/sda7           349       626   2101648+  83  Linux
/dev/sda8           627       904   2101648+  83  Linux
/dev/sda9           905      2184   9676768+  83  Linux
```

```
Command (m for help): d
Partition number (1-9): 3
```

Repeat this step for every partition you want to delete. When you're finished, type the w command to write the changes to disk and exit fdisk (of course, you probably want to wait until you've created the new partitions as well).

```
Command (m for help): w
```

Adding Partitions

Now to add new partitions, I will work from the given partition layout example given previously. Also, I assume that the disk has no partitions. So first, let's create /dev/sda1:

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-12621, default 1): (Press return to use the default "1")
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-621, default 621):
+100M
```

In the above command sequence, I asked **fdisk** to create a new partition which will be a primary one (remember, if I would select extended then I would be creating a container partition which can then host logical partitions). **fdisk** then asks for the first cylinder and proposes as default the first cylinder (which means as much as "start at the beginning of the disk"). Then **fdisk** asks where the partition should end. As I'm not going to calculate which cylinder is around the 100Mbyte limit, I just tell fdisk to create a partition of 100Mbyte size.

Next, I create /dev/sda2, 3 and 4 in one go:

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (97-12621, default 97): (Return again)
Using default value 97
Last cylinder or +size or +sizeM or +sizeK (97-12621, default 12621):
+12288M

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 3
First cylinder (4041-12621, default 4041): (Return again)
Using default value 4041
Last cylinder or +size or +sizeM or +sizeK (4041-12621, default
12621): +27648M

Command (m for help): n
Command action
   e   extended
```

```
   p   primary partition (1-4)
p
Partition number (1-4): 4
First cylinder (12021-12621, default 12021): (Return again)
Using default value 12021
Last cylinder or +size or +sizeM or +sizeK (12021-12621, default
12621): (Return)
```

In the last sequence we let the final partition be as large as the remaining amount of disk space left, so we accept the default last cylinder proposal.

Right now, the partition scheme is set, but the partitions aren't ready yet. I now need to mark each partition with a particular partition type. A *partition type* is a small label assigned to a partition that allows an operating system to know if it can read the partition (i.e. understand its content) or not.

For instance, the partition type "Linux" (label 83) allows Linux operating systems to identify the partition as a partition it understands. Windows on the other hand will not show this partition as it does not support Linux partitions. Likewise, a FAT32 Windows partition has label 0B (labels are hexadecimal).

In our example, we need to use two partition types: one to identify a Linux partition (83) and one to identify the Linux swap partition (82). To mark partitions as such, use fdisk's **t** command:

```
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): 83
Changed system type of partition 1 to 83 (Linux)

Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 83
Changed system type of partition 2 to 83 (Linux)

Command (m for help): t
Partition number (1-4): 3
Hex code (type L to list codes): 83
Changed system type of partition 3 to 83 (Linux)

Command (m for help): t
Partition number (1-4): 4
Hex code (type L to list codes): 82
Changed system type of partition 4 to 82 (Linux swap)
```

Now that our partitions are created and marked, write the changes to disk and exit fdisk with the **w** command.

Placing a File System on a Partition

With a partition alone you cannot do much: the partition is available as an empty space, but has no file system on it. File systems are mandatory to use as they structure the partition and allow the operating system to manage files, directories and more on the partition. We covered this already in the section called “File Systems”.

To place a file system on a partition you need to use the **mkfs.<type>** command. For instance, to create an ext2 or ext3 partition, use **mkfs.ext2** or **mkfs.ext3**. With the example partition scheme I would use ext2 for the /boot partition (`/dev/sda1`), and ext3 for the two other Linux partitions:

```
# mkfs.ext2 /dev/sda1
# mkfs.ext3 /dev/sda2
```

```
# mkfs.ext3 /dev/sda3
```

A good idea is to label each partition. A *volume label* is a simple string of limited length (16 bytes, so 16 characters as I don't suspect unicode is supported here) which can allow you to find the partition more easily. Say that you label your partitions based on their use:

```
# mkfs.ext2 -L boot /dev/sda1
# mkfs.ext3 -L root /dev/sda2
# mkfs.ext3 -L home /home/sda3
```

With labels set, you can use label-based device file names instead of the (sometimes difficult to remember) standard ones: `/dev/disk/by-label/root` instead of `/dev/sda2`.

Finally, I need to mark the swap partition as a swap partition using the **mkswap** command. This command also supports the **-L <label>** option if you want to use it.

```
# mkswap /dev/sda4
```

Using the Partitions

With the selected partitions created, you can now start using them.

Enabling a Swap Partition

To enable a swap partition, use the swapon command. This will inform the Linux kernel that the selected partition can be used as a swap partition:

```
# swapon /dev/sda4
```

Because you do not want to enter this command every time you boot your system, add the following line to /etc/fstab. This will automatically enable the selected partition as a swap partition:

```
/dev/sda4  none   swap   sw   0 0
```

Enabling a File System

To enable a file system on a partition, you need to mount it on your file system. This has already been covered in the section called “The 'mount' Command and the fstab file”.

Fixing Corrupt File Systems

If a file system is corrupt, you will notice this when you want to mount the file system (or when the system tries to automatically mount the file system for you):

```
/dev/hda4:
The superblock could not be read or does not describe a correct ext2
filesystem. If the device is valid and it really contains an ext2
filesystem (and not swap or ufs or something else), the the superblock
is corrupt, and you might try running e2fsck with an alternate superblock:
         e2fsck -b 8193 <device>

* Filesystem couldn't be fixed :(
```

Now, before you run into the night, screaming for help, sit down and try executing the command that the output gave:

```
(The proposed command will vary depending on file system used)
~# e2fsck -b 8193 /dev/hda4
```

If the **e2fsck** check reports that there is corruption found, it might ask you to confirm every correction it wants to do. As a file system check can easily report hundreds to thousands of corruptions (not that that means that there are thousands of files corrupt), it might be easier to tell e2fsck to just acknowledge them all for you:

```
~# e2fsck -y /dev/hda4
```

Using File System Labels or IDs

Most, if not all file systems allow you to give them an appropriate label. Such labels can then later be used to identify a file system without having to mount it (and look at it). Linux even supports the use of these labels for the various mount and file system operations. The use of labels (or UUIDs, as we will see later) also allows one to use configurations (such as in the `fstab` file) which do not need to be modified when your system gets changed (for instance, new partitions created, new disks added, reshuffling of disks and more).

Labels versus UUIDs

There are two identifiers commonly used when dealing with file systems: LABEL and UUID.

- a *LABEL* is a user-provided name for a file system. An example could be "ROOT", "HOME" or "DATA".
- a *UUID* (Universally Unique Identifier) is a system-generated identifier for a file system. Examples are "bae98338-ec29-4beb-aacf-107e44599b2e" and "31f8eb0d-612b-4805-835e-0e6d8b8c5591"

As you can imagine, a given label is much more user friendly than a UUID. So, how do you set a label for a file system? Well, this heavily depends on the file system you use. For ext2, ext3 or ext4 file systems, you can use the **e2label** command:

```
~# e2label /dev/sda2 ROOT
```

For an XFS file system, the command would be given with **xfs_admin**:

```
~# xfs_admin -L ROOT /dev/sda2
```

You can even set labels for swap file systems (**mkswap -L <labelname> <device>**), FAT file systems (**mlabel -i <device> ::<labelname>**) and JFS file systems (**jfs_tune -L <labelname> <device>**).

The easiest method to read the label and UUID of a file system is to use the **blkid** command:

```
~# blkid /dev/sda3
/dev/sda3: UUID="2bc32022-27a8-47d5-8d33-83c86e23618c" LABEL="ROOT" TYPE="ext4"
```

Using Labels/UUIDs in fstab

If you have set a label for your file system(s) (or use UUIDs) you can use this information in the `/etc/fstab` file. Just substitute the value in the first column (where the device is located) with the correct LABEL= or UUID= setting:

```
/dev/sda2     /  ext4     defaults,noatime   0 0
```

could then become one of the following:

```
LABEL="ROOT"  /  ext4     defaults,noatime   0 0
```

or

```
UUID="bc32022-27a8-47d5-8d33-83c86e23618c"  /  ext4   defaults,noatime   0 0
```

(Not) Using Labels/UUIDs as Kernel Options

Some people hope to use the same information as kernel option (for instance, to change the *root=/dev/sda2* kernel parameter to *root=LABEL=ROOT*). This is possible, but only if you use an initramfs (so use this for the *real_root=* parameter). The Linux kernel itself does not support calling devices through their UUID or LABEL information.

Removable Media

Removable media differs from partitions in the fact that they are... removable. Some removable media cannot be written to, others can. If you can write to it, you most likely can partition (and put a file system on it) just as if it was a hard disk.

The most important difference is that they are not always available for the system: you can plug (put) them in or pull them out, so you should mount and unmount the file system. Luckily, there are tools that automatically mount / umount such devices.

Mounting Removable Media

As seen before, media mount points can be defined in `/etc/fstab` to ease the mount process. Two examples (one for a CD-ROM device and one for a USB storage device) could be:

```
/dev/cdrom    /media/cdrom    auto   defaults,user,noauto   0 0
/dev/sdb1     /media/usb      auto   defaults,user,noauto   0 0
```

As you can see, the mounts are defined with auto as file system type (meaning that the mount process attempts to automatically find the file system to use) and has user (users have the ability to mount this location) and noauto (do not attempt to mount this when the system boots) as options.

But in order to successfully edit the fstab file, you need to know what device will be used and you also need to make sure that the destination directories exist.

Device Files and udev

The **udev** device manager creates device files for your partitions, including removable media when it is attached to the system. One of the advantages of using udev is that it also creates various symlinks that identify the same device. For instance, a plugged in USB stick can get the following device files created:

```
/dev/sdb1
```

The following links can then be created to this device file:

```
/dev/block/8:17
/dev/disk/by-id/usb-_USB_DISK_2.0_0789E600025-0:0-part1
/dev/disk/by-path/pci-0000:00:1d.7-usb-0:1:1.0-scsi-0:0:0:0-part1
/dev/disk/by-uuid/3466-4C39
/dev/disk/by-label/UDISK-2.0
```

The advantage of having these links is that, when you plug another USB stick first and then this one, it's device file might be different (say `/dev/sdc1`) but the by-id, by-uuid and by-label links will remain the same.

Obtaining the device file

When you plug in a removable media (like a USB stick), the kernel will log that it has detected the hardware. One way to find out which device file is used is to filter the **dmesg** output for text such as "removable media":

```
~# dmesg | grep 'removable'
sd 4:0:0:0: [sdb] Attached SCSI removable disk
```

In this example, the hardware device file is for /dev/sdb. The partitions can then be obtained by listing all files in /dev starting with sdb:

```
~# ls /dev/sdb*
/dev/sdb    /dev/sdb1
```

Now, if you want to find out which files point to (or are hardlinks to) a particular device file (say /dev/sdb1), you can use the following **find** construct:

```
~# find -L /dev -samefile /dev/sdb1
/dev/sdb1
/dev/disk/by-label/UDISK-2.0
/dev/disk/by-uuid/3466-4C39
/dev/disk/by-id/usb-_USB_DISK_2.0_07891E600025-0:0-part1
/dev/disk/by-path/pci-0000:00:1d.7-usb-0:1:1.0-scsi-0:0:0:0-part1
/dev/block/8:17
```

Network File Systems

Although Unix has grown with NFS as the primary network file system, others are available which offer different features you might require...

NFS

NFS, or *Network File Server*, is one of the most popular network file systems used in the Linux/Unix world. With NFS, you can export files and directories to other systems on the network while still using the Unix way of assigning permissions and ownership to files.

NFS Server

If you want to export files to your network yourself, you need to install the necessary NFS server tools which is included in the nfs-utils package:

```
$ emerge nfs-utils
```

Once installed, you need to select what directories you want to export through the `/etc/exports` file. The syntax for the exports file is similar to the SunOS exports file[1] and allows you to select a particular directory as an exportable resource together with specific mount options for the clients. I will give a multi-line example that will get you going:

```
/usr/portage  192.168.1.0/24(ro)
/home 192.168.1.0/24(rw,no_subtree_check)
/media/usb rw,no_root_squash mediacenter ws5 ws6 ws7(root_squash)
```

- The first line gives read-only access to /usr/portage to all systems in the 192.168.1.0/24 network.
- The second line gives read-write access to /home to all systems in the same network. I also added the no_subtree_check (see the exports man page for more information) as it improves reliability for file systems where file changes occur frequently (such as home directories).
- The third line gives read-write access to the /media/usb location to the hosts with hostname mediacenter, ws5, ws6 and ws7. Also, all these hosts except for ws7 have root access to the files as

[1]Some Unix flavors have different syntax rules for the exports file. If you are not running Gentoo Linux or have installed a different NFS server package, check the exports man page through **man exports**.

well (as a security measure, by default the root user of the remote clients do not have root access to the file systems exported by NFS).

As you can see from the syntax, you can either give options specific for a host (between brackets) or give general options to all hosts for the selected directory.

To start the NFS service, you only need to start the nfs runlevel:

```
# /etc/init.d/nfs start
```

All other required services, such as rpc.statd and portmap, are automatically started by the initialisation script.

NFS Client

At the client side, you add in one or more lines in the `/etc/fstab` file to mount a remote NFS file system. Suppose that the NFS server is at 192.168.1.100, you can use the following lines in `/etc/fstab` to mount the file systems defined in the previous section:

```
192.168.1.100:/usr/portage  /usr/portage       nfs  ro,proto=tcp  0 0
192.168.1.100:/home         /home/remoteusers  nfs  rw,proto=tcp  0 0
192.168.1.100:/media/usb    /media/usb         nfs  ro,proto=tcp  0 0
```

You can of course also run an NFS mount without `fstab`. For more information about the NFS mount options, please read the nfs manual page.

Samba

Samba, offered by the net-fs/samba package, is a tool set that offers interoperability with Microsoft Windows networks. Not only can you access files shared on a Microsoft Windows network or share files yourself on such a network, you can also use or manage printers that are exported to the Microsoft Windows network.

In the majority of cases, if you need to do something with a Microsoft Windows network, you'll need to configure Samba on your system. Although the configuration options for Samba can be overwhelming, there is an integrated web administration tool for Samba called SWAT which is included when you install the net-fs/samba package with the swat USE flag (which is enabled by default).

Managing Disk Space

To identify the current file system usage, you can use the **df** tool, which stands for "disk free". I recommend using the -h and -T options to show disk usage in a human readable format (using the M and G notations for megabytes and gigabytes) and display the file system types (ext2, ext3, xfs, ...):

```
$ df -hT
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/sda8     ext3     35G   22G   11G  68% /
udev         tmpfs     10M   88K   10M   1% /dev
/dev/sda7     ext3    9.4G  7.6G  1.5G  85% /home
none         tmpfs    754M     0  754M   0% /dev/shm
/dev/sda1     ext3    9.7G  421M  8.8G   5% /mnt/data
```

In the above example you can see the available disk space on all mounted file systems.

Finding Top Disk Space Consumers

To find out how much disk space a file (or, more interestingly, a directory with all files and subdirectories) you can use the **du** command (disk usage). By default, **du** shows the disk usage in

kilobytes for every file passed on to it. You can have **du** show the total using the -s option (summarize) and even have du show it in a more human-readable manner (so 12.6G instead of 13269313 - kilobytes).

```
$ du -hs /home/raghat
12.6G     /home/raghat
```

If you want to view the largest users (subdirectories or files inside the directory) you can do a summary listing of everything beneath it and sort the resulting output by number. To make it a bit manageable, the next example then only shows the last 5 lines (the largest 5 consumers).

```
$ du -ks /home/raghat/* | sort -n | tail -5
7660    folder.pdf
7672    linux_sea.pdf
8532    docs/
3666665 hmd/
```

However, you'll quickly find that this process is tedious (you'll have to repeat it for every subdirectory if you don't plan on wiping out the entire directory) when you need to free some space. One of the solutions is to find large files, for instance through the find command.

The next example finds all files of at least 50mbyte in size from the current location:

```
$ find . -type f -size +50M
./tmp/download/testvid001.avi
./tmp/download/testvid002.avi
./iso/SysRescCD.iso
```

Another frequently used method for cleaning up files is to find all files of a certain age (modification time). For instance, to find all files of at least 10Mbyte with an age of at least 1 year old (365 days):

```
$ find . -type f -size +10M -mtime +365
```

Cleaning Up Gentoo-related Files

If you work with Gentoo, you have a few commands at your disposal that help you clean out Gentoo-specific files as well.

With **eclean**, which is part of app-portage/gentoolkit, most Gentoo space-consuming directories can be cleaned in a safe manner:

- the distfiles location, where Gentoo Portage stores all downloaded source code (in case it needs to rebuild packages), can be cleaned with **eclean distfiles**:

  ```
  # eclean distfiles
  ```

  As a result, all downloaded source code of packages that are not installed on the system any more is removed from the distfiles location.

- the packages location, wheren Gentoo Portage stores the binary package builds of the packages you install (this is not on by default, but I recommend it if you have sufficient amount of space as it can help you recover from certain disasters quickly), can be cleaned with **eclean packages**:

  ```
  # eclean packages
  ```

Another location that you may clean is `/var/tmp/portage`. At this location, Portage performs the entire build process. If an ebuild build process fails, the files are left at this location (this is deliberate, so that one can debug and perhaps fix the build failure manually). Since most users don't manually fix this, if you are not emerge'ing anything, this entire folder can be made empty.

Resizing Partitions

It is possible to resize partitions, although I recommend that, if you plan on using this a lot, you should take a look at LVM2 (logical volume management) which provides an easier and safer method for resizing filesystems.

Resizing xt2/ext3 file systems

To increase the size of an ext3 file system, you first need to resize the partition. You can do this with **fdisk** by removing the partition and then recreating it, starting from the same point as the original partition started, but now larger. Of course, this means that the partition cannot be mounted at that time. Then, run **resize2fs** against the device. The tool will automatically expand the file system to use the entire partition again. Once the resize is completed, you can mount the file system again.

```
# fdisk /dev/sda
(Edit partition sda3, making it take more space than before)
# resize2fs /dev/sda3
```

If you want to decrease the size of an ext3 file system, you first need to use **resize2fs** to shrink the file system. Then you edit the partition table again, removing the existing partition and recreating it, but with a smaller size, and again starting the partition from the same point as the original partition started:

```
# resize2fs /dev/sda3 10G
# fdisk /dev/sda
```

As you can see, resizing an ext2/ext3 file system suffers from the following draw-backs:

- the file system cannot be mounted at the time of resizing. If you want to resize the root partition, this means you'll need to boot from a LiveCD, LiveUSB or another operating system
- the resize operation can only manipulate the end of the partition. The start of the partition must remain the same. In other words, if your entire disk is used by partitions, you cannot increase any partition without deleting one or more partitions behind it (you cannot shrink the last partition by moving its starting point closer to the end).

Resizing other file systems

Almost every file system technology has a command to resize its file system. Some even support online resizing, provided that the storage on which the file system is put (partition) already has unallocated space available. As most people use partitions, this still requires unmounting the file system and editing the partition table using fdisk.

The XFS file system has **xfs_growfs**, the JFS file system uses mount options to resize:

```
(Resizing a JFS file system)
# mount -o remount,resize /jfs/filesystem/mountpoint
```

Limiting User Files

It is possible to limit the amount of files, or total disk space that a particular user can take on an individual partition. This is especially useful when you have multiple users on your system (and you don't want a single user to take almost all disk space on your /home partition - provided that it already is a separate partition, which I strongly recommend). A second possible reason to do this is when / home is not on a separate partition, and you don't want any user to be able to fill up your root partition.

This kind of support is called *quota* support. Quota support (which requires kernel level support as well) is based upon two simple files that you place in the root folder of the mount point. Inside this file, you specify per user (for the `aquota.user` file) or group (for the `aquota.group` file) how

much blocks (kilobytes) and/or inodes (files/directories) can be used. The soft limit there is when the system will start complaining, the hard limit is a final limit - the user or group cannot go beyond that.

To use quota, first install the quota utilities.

```
# emerge sys-fs/quota
```

Next, edit /etc/fstab and add usrquota,grpquota (or one of the two depending on how you want to configure your quota's) as mount parameters.

```
# nano -w /etc/fstab
(Edit the mount parameters, like so:)
/dev/sda3  /home   ext3   defaults,noatime,usrquota,grpquota  0 0
```

Create, inside the mount point, the quota files. Then remount the file system and initialize the quota files:

```
# touch /home/aquota.user /home/aquota.group
# chmod 600 /home/aquota.*
# mount -o remount /home
# quotacheck -avugm
# quotaon -avug
```

Finally, set the quota's using the **edquota** commands. The next example edits the quota's for the raghat user and for the group "usergroup". The command will open up your standard editor (**vi** or **nano**) and display the current quota limits (0 means unlimited). You can then just edit the file to your liking. Changes will be effective immediately.

```
# edquota -u raghat
# edquota -g usergroup
```

Resources

- Linux Partition HOWTO [http://tldp.org/HOWTO/Partition/], section "Partitioning with fdisk [http://tldp.org/HOWTO/Partition/fdisk_partitioning.html]"
- Multimedia with KDE [http://docs.kde.org/stable/en/kdebase-runtime/userguide/multimedia.html], part of KDE base documentation

Chapter 14. System Management

Introduction

System management is a broad term. It is my attempt to cover the system administration tasks that almost every administrator (or end user) will need to know for his system, such as time management, language management, keyboard settings and more.

Environment Variables

The Linux operating system makes extensive use of environment variables.

An environment variable is a simply a key-value pair which a process can read out. For instance, the environment variable `EDITOR` (with, as an example, value `/bin/nano`) informs the process who reads it that the default text editor is (in this case) nano. These variables are not system-wide: if you alter the value of a variable, the change is only active in the session where you are in (which is your shell and the processes started from the shell).

List of Environment Variables

There are quite a few environment variables you'll come across often.

DISPLAY

The *`DISPLAY`* environment variable is used when you're logged on to a Unix/Linux system graphically. It identifies where X applications should "send" their graphical screens to. When you log on to a system remotely, this variable is set to your local IP address and the screen number you're using on this system. Most of the time, when you're logged on locally, it's content is ":0.0" (the first screen on the system).

Note that "screen" here isn't the hardware device, but a name given to a running X instance.

EDITOR

The *`EDITOR`* variable identifies the default text editor you want to use. Applications that spawn a text editor (for instance, visudo) to edit one or more files, use this variable to know which text editor to launch.

LANG and other locale specific variables

Locales are discussed later in this chapter. Its environment variables (*`LANG`* and the various *`LC_*`* variables) identify the users' language, time zone, currency, number formatting and more.

PATH

The PATH variable identifies the directories where the system should look for executable files (being binaries or shell scripts). If unset or set incorrectly, you cannot execute a command without providing the entire path to this command (except built-in shell commands as those are no executable files).

Below is a small example of a PATH variable:

```
~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/opt/bin:/usr/i686-pc-linux-gnu/gcc-bin/4.1.2:
/opt/blackdown-jdk-1.4.2.03/bin:/opt/blackdown-jdk-1.4.2.03/jre/bin:
/usr/kde/3.5/bin:/usr/qt/3/bin:/usr/games/bin:/home/swift/bin/
```

An example of what happens when PATH is not set:

```
~$ ls
(... listing of current directory ...)
~$ unset PATH
~$ ls
-bash: ls: No such file or directory
~$ /bin/ls
(... listing of current directory ...)
```

TERM

The *TERM* variable allows command-line programs with special characters to identify which terminal you use to run them. Although nowadays the xterm TERM is most used, sometimes you will find yourself logged on to a different system which doesn't know xterm or where the application looks really awkward. In such cases a solution could be to set the TERM variable to, for instance, vt100.

How to Set an Environment Variable

Environment variables are user specific, but can be set on three levels: session (only valid for the current, open session), user (only valid for this user and used as default for all sessions of this user) or system wide (used as a global default).

Session Specific

When you want to set an environment variable for a specific session, you can use the shell **set** or **export** command:

```
~$ ls -z
ls: invalid option -- z
Try `ls --help` for more information.
~$ export LANG="fr"
~$ ls -z
ls: option invalide -- z
Pour en savoir davantage, faites: `ls --help`
```

Which one to use depends on what you actually want to achieve:

- With **set**, you change the environment variable for this session, but not for the subshells you might want to start from the current shell. In other words, set is local to the shell session.
- With **export**, you change the environment variable for this session as well as subshells you might want to start from the current shell from this point onward. In other words, export is global.

User Specific

User specific environment settings are best placed inside the `.bashrc` file. This file is automatically read when a user is logged on (at least when he is using the bash shell). A more shell-agnostic file is `.profile`. Inside the file, define the variables as you would for a specific session:

```
export LANG="fr"
```

System Wide Defaults

To make an environment variable system wide, you must make sure that your environment variable is stored in a file or location that every session reads out when it is launched. By convention, `/etc/profile` is a script in which system wide environment variables can be placed. Gentoo offers a nice interface for this: inside `/etc/env.d` you can manage environment variables in a more structured

approach, and the **env-update.sh** script will then make sure that the environment variables are stored elsewhere so that `/etc/profile` reads them out.

Note

The /etc/profile script does not read out all values inside /etc/env.d itself for (at least) two reasons:

1. The structure used in /etc/env.d uses a specific "appending" logic (i.e. variables that are defined several times do not overwrite each other; instead, their values are appended) which could be too hard to implement in /etc/profile without too much overhead. After all, /etc/profile is read by every newly launched session, so if it took too much time, your system would start up much slower.

2. The system administrator might want to make a set of changes which should be made atomic (for instance, remove a value from one variable and add it to another). If changes are publicized immediately, a session could read in /etc/profile which loads an at that time incorrect environment variable set (especially when a process is launched after the administrators' first change but before the second).

Managing Environment Entries

On Linux, the behaviour of many commands is manipulated by values of environment entries, or environment variables. Within Gentoo Linux, you can manage the system-wide environment variables through the `/etc/env.d` directory.

Environment Files

Inside /etc/env.d, you will find environment files which use a simple key=value syntax. For instance, the /etc/env.d/20java file defines, amongst other environment variables, the PATH and MANPATH variables:

```
# cat /etc/env.d/20java
...
MANPATH=/opt/blackdown-jdk-1.4.2.03/man
PATH=/opt/blackdown-jdk-1.4.2.03/bin:/opt/blackdown-jdk-1.4.2.03/jre/bin
```

With these settings, the value of MANPATH (location where man will search for its manual pages) and PATH (location where the system will look for executable binaries every time you enter a command) is *extended* with the given values (note that the variables are not rewritten: their value is appended to the value previously assigned to the variable).

The order in which variable values are appended is based on the file name inside `/etc/env.d`. This is why most files start with a number (as most people find it easier to deal with order based on numbers, plus that the filenames themselves are still explanatory to what purpose they serve).

Changing Environment Variables

If you want to change a system variable globally, you can either add another file to `/etc/env.d` or manipulate an existing one. In the latter case, you should be aware that application upgrades automatically update their entries inside `/etc/env.d` without warning (this location is not protected, unlike many other configuration locations).

As such, it is advisable to always add your own files rather than manipulate existing ones.

When you have altered an environment file or added a new one, you need to call **env-update** to have Gentoo process the changes for you:

```
# env-update
```

This command will read in all environment files and write the final result in `/etc/profile.env` (which is sourced by `/etc/profile`, which is always sourced when a user logs on).

Location Specific Settings

When I talk about location specific settings, I mean the settings that your neighbour is most likely to need as well: language settings, keyboard settings, time zone / currency settings, ... Within the Linux/Unix environment, these settings are combined in the locale settings and keyboard settings.

Locale Settings

A *locale* is a setting that identifies the language, number format, date/time format, time zone, daylight saving time and currency information for a particular user or system. This locale information is stored inside a variable called *`LANG`*; however, it is possible to switch a particular locale setting to another locale (for instance, use the American English settings for everything, but currency to European Euro).

The following table gives an overview of the most important variables:

Table 14.1. Locale variables supported on a Linux system

`LANG`	A catch-all setting which identifies the locale for all possible features. However, individual topics can be overridden using one of the following variables.
`LC_COLLATE` and *`LC_CTYPE`*	Character handling (which characters are part of the alphabet) and (alphabetical) order
`LC_MESSAGES`	Applications that use message-based output use this setting to identify what language their output should be
`LC_MONETARY`	Currency-related settings
`LC_NUMERIC`	Formatting of numerical values
`LC_TIME`	Time related settings

There is another variable available as well, called *`LC_ALL`*. If this variable is set, none of the above variables is used any more. However, use of this variable is strongly discouraged.

To get an overview of your locale settings (including a full list of supported variables), enter the **locale** command.

The format of a locale variable is as follows:

```
language[_territory][.codeset][@modifier]
```

The settings used in this format are:

Table 14.2. List of settings used in a locale definition

language	Language used. Examples are "en" (English), "nl" (Dutch), "fr" (French), "zh" (Chinese)
territory	Location used. Examples are "US" (United states), "BE" (Belgium), "FR" (France), "CN" (China)
codeset	Codeset used. Examples are "utf-8" and "iso-8859-1"
modifier	Modifier used, which allows a different definition of a locale even when all other settings are the

	same. Examples are "euro" and "preeuro" (which has its consequences on the monetary aspect).

So, a few examples are:

```
LANG="en"
LANG="nl_BE"
LANG="en_US.utf-8"
LANG="nl_NL@euro"
```

These settings are read as environment variables (which were discussed earlier) by the applications. You can mark locales system wide, but it is advised that this is stored on a per-user basis. As such, I recommend that you set something like the following in your `~/.bashrc` file (and in `/etc/skel/.bashrc` so that newly created user accounts have this set automatically as well):

```
$ nano -w ~/.bashrc
...
# Put your fun stuff here
LANG="en_US.utf-8"
```

Keyboard Settings

When you aren't using the default qwerty layout, you'll need to modify the keyboard mapping setting on your system. Gentoo makes this easy for you: edit `/etc/conf.d/keymaps` and set the `keymap` variable to the mapping you need:

```
# nano -w /etc/conf.d/keymaps
...
keymap="be-latin1"
```

A list of supported keymaps can be found in the subdirectories of `/usr/share/keymaps`.

If you want to test and see if a particular keymap is correct, load it manually using the **loadkeys** command:

```
# loadkeys <keymap>
```

Time Settings

To change the system time/date, you can use the **date** command. For instance, to set the date to september 30th, 2008 and time to 17.34h:

```
# date 093017342008
```

If your system has Internet access, it is wise to install ntp-supporting tools such as the net-misc/ntp package. With **ntpdate** (and other similar tools), you can use online time servers to set the time of your system correct to the second.

```
# ntpdate pool.ntp.org
```

To save the current (operating system) time to your hardware clock, you can use the **hwclock** program:

```
# hwclock --systohc
```

System Scheduler

Within Unix/Linux, the default scheduler often used is called *cron*. There are quite a few cron implementations available, such as the popular **cronie**, **fcron**, **bcron** and **anacron**. Once installed, you start the cron service through an init script (which you most likely add to the default runlevel):

```
# rc-update add cronie default
# /etc/init.d/cronie start
```

When the cron service is running, every user can define one or more commands he wants to periodically execute.

To edit your personal scheduling rules, run **crontab -e**:

```
$ crontab -e
```

Your current rule file will be shown in the default editor (nano, vim, ...). A crontab entry has 6 columns:

Table 14.3. Crontab columns

Minute	Minute of the hour (0-59)
Hour	Hour of the day (0-23)
Day	Day of the month (1-31)
Month	Month of the year (1-12 or use names)
Weekday	Day of the week (0-7 or use names. 0/7 are Sunday)
Command	Command to execute

Next to the number representation, you can use ranges (first-last), summation (1,3,5), steps (0-23/2) and wildcards.

For instance, to execute "**ntpdate ntp.pool.org**" every 15 minutes, the line could look like:

```
*/15 * * * * ntpdate ntp.pool.org
```

or

```
0,15,30,45 * * * * ntpdate ntp.pool.org
```

If you just want to view the scheduled commands, run **crontab -l**.

Chapter 15. Installing Gentoo Linux

Introduction

I've waited a few chapters before I started discussing the Gentoo Linux installation because it isn't for the faint of heart. Although Gentoo has tried to offer a graphical installer in the past, its user- and developer base swore by the manual installation approach. As a result, the graphical installer has been deprecated and the installation procedure is once more a manual, step by step guide.

With the previous chapters discussed, you should now be able to install a Gentoo Linux yourself with the following simple set of instructions. However, if you want to do it the official way, do not hesitate to read the Gentoo Handbook [http://www.gentoo.org/doc/en/handbook/handbook-x86.xml]. There are also handbooks [http://www.gentoo.org/doc/en/handbook] available for other architectures.

System Requirements

Gentoo Linux can be as heavy as you want, or as small as you want. Yet unless you deviate from the documented approach, Gentoo Linux remains a source-based distribution. Because of that, it has a slightly higher disk space requirement than other distributions. A minimal (base) Gentoo Linux installation takes a little less than 2Gbyte of diskspace (including Linux kernel source code and the Gentoo Portage tree, which take about 640 Mbyte in total). But with 2Gbyte, you don't have room for much more than a base installation.

To have a comfortable installation, yet with room to spare for additional installations, you should consider a total diskspace of at least 20 Gbyte for the applications alone. With current disk sizes, this should not be a problem. If you don't install a full-blown KDE or GNOME, you should have enough with 10Gbyte or less.

Disk space aside, Gentoo Linux can run and install with almost any system specification. Of course, the lower the specs, the higher the duration of an installation. Installing Gentoo Linux on a i486 with 32 Mbyte of memory is doable (but not recommended).

Booting a Linux Environment

A Gentoo Linux installation starts from a Linux environment. You can use any Linux environment you want, but most people suggest to use a LiveCD.

A popular LiveCD to install Gentoo from is System Rescue CD [http://www.sysresccd.org]. All necessary documentation about booting the CD, including setting up networking (which you definitely need to do in order to install Gentoo) is available on the site.

Disk Setup

Once your environment is set up, you'll need to setup your disks by partitioning them and then putting a file system on them. Partitioning and file system management has been discussed beforehand. If you want, assign labels to the file systems to make it easier to create your `fstab` file later:

```
# fdisk /dev/sda
(Partition the disk)
# mkfs.ext2 -L BOOT /dev/sda1
# mkfs.ext3 -L ROOT /dev/sda2
# mkfs.ext3 -L HOME /dev/sda3
# mkswap -L SWAP /dev/sda4
```

Once that your partitions are created and a file system is put on it, it is time to really start the Gentoo Linux installation.

First, mount all the necessary partitions onto your Linux environment. In the rest of this chapter I will assume the partitioning layout as described inthe following table:

Table 15.1. Example partition layout

Device	Partition	Description
/dev/sda1	/boot	Small boot partition to hold the Linux kernel and bootloader information. Can be ext2
/dev/sda2	/	Root partition; should be fairly large in this example. Suggested is ext3
/dev/sda3	/home	Home partition where all users' files are stored. Best to always have a separate partition for the home directories so that future reinstallations can reuse the home structure.
/dev/sda4	<none>	Swap partition, roughly 1.5 times the amount of physical memory nowadays (still this large because I want to use hibernate-to-disk).

```
~# mkdir /mnt/gentoo
~# mount /dev/sda2 /mnt/gentoo
~# mkdir /mnt/gentoo/boot
~# mount /dev/sda1 /mnt/gentoo/boot
~# mkdir /mnt/gentoo/home
~# mount /dev/sda3 /mnt/gentoo/home
~# swapon /dev/sda4
```

With the above commands executed, the various file systems we will use for the Gentoo installation are now available at `/mnt/gentoo`. Every file or directory we put beneath `/mnt/gentoo` will show up on our final Gentoo installation. For instance, `/mnt/gentoo/boot` = `/boot`.

Installing Gentoo Base

First, set your system time correct so that the files you're going to create do not have a weird time stamp:

```
~# ntpdate pool.ntp.org
```

Next, surf to the Gentoo mirror list [http://www.gentoo.org/main/en/mirrors2.xml] and pick a mirror close to you. On most LiveCDs browsers are available. On the sysresccd you can use links or lynx (command-line browsers). Navigate to releases, select your architecture, autobuilds, the latest date directory to find a listing of stage3 files and install files.

```
~# cd /mnt/gentoo
~# links http://www.gentoo.org/main/en/mirrors2.xml
```

- A stage3 file is an archive of a prebuilt Gentoo environment which we will extract to the installation location (`/mnt/gentoo`)
- An install file is an ISO file (CD image) which contains a minimal Gentoo environment from which you can boot and install Gentoo from.

Download the stage3 file and store it in `/mnt/gentoo`. If you have the full URL at hand, you can also use **wget**:

```
# cd /mnt/gentoo
# wget http://gentoo.osuosl.org/releases/x86/autobuilds/20091201/stage3-i686-20
```

On many forums, you will find the notion of "funtoo" stages. Funtoo [http://www.funtoo.org] is, to say it in the author's own words (who happens to be Daniel Robbins, the founder of Gentoo Linux), a Gentoo Linux variant which offers freshly-built Gentoo Linux stable stages using Gentoo's official stable branch. You can use a funtoo stage instead of a Gentoo official stage if you want. After all, they both contain roughly the same material. However, some caution is still in place: the Funtoo stages continuously evolve and diverge into their own set so I recommend to take a quick stab at the Funtoo installation instructions nevertheless. At the time of writing, the instructions are quite resembling.

Next, go back a few directories until you can select snapshots. Enter this directory and download the latest `portage-<date>.tar.bz2` you can find. Store it in `/mnt/gentoo` as well. Finally, quit your browser and extract the downloaded files on your installation location.

```
~# tar xvjpf stage3-*.tar.bz2
```

Next, edit the `/mnt/gentoo/etc/make.conf` file. As discussed previously, this file contains variables that define Portage' behaviour. Right now I'm focussing on the variables CFLAGS, CXXFLAGS and MAKEOPTS...

- `CFLAGS` (C) and `CXXFLAGS` (C++) inform gcc (GNU's Compiler Collection) what optimizations it should use (see Compiler Directives)
- MAKEOPTS defines how many parallel compilations should occur when you install a package (especially useful for multicore / SMP systems). A good choice is the number of core's in your system plus one (for instance, a dual-core CPU would lead to `MAKEOPTS="-j3"`).

You can edit the `make.conf` file using **nano**, **vim** or any other text editor.

Configuring the System

Our next step is to configure the installation environment.

Preparing the Installation Environment

First, prepare the environment for chrooting. *Chrooting* is the process of altering your sessions' file system root to another location. In our case, `/mnt/gentoo` should become `/` for your running session. In order to chroot successfully, we need to ensure that networking will still function properly and that both kernel data and device drivers are available inside the chroot:

```
~# cp -L /etc/resolv.conf /mnt/gentoo/resolv.conf
~# mount -t proc none /mnt/gentoo/proc
~# mount -o bind /dev /mnt/gentoo/dev
```

Chrooting

Now, chroot into the Gentoo installation environment, update your environment variables and, for safety reasons, change your prompt so that you know you're inside your Gentoo installation environment.

```
~# chroot /mnt/gentoo /bin/bash
~# env-update
~# source /etc/profile
~# export PS1="(chroot) $PS1"
```

Right now, this session (where the prompt starts with "(chroot)") is inside your Gentoo installation environment.

Configuring Portage

Now, update the Portage tree to make sure you have the current set of packages at your disposal:

```
~# mkdir /usr/portage; emerge-webrsync;
```

Next, select a Gentoo profile for your environment. A *Gentoo profile* is a collection of default Portage settings. If you want to know what a particular profile selects of default settings, check out its content at `/usr/portage/profiles` (and don't forget to read up on cascading profiles). Currently, the '13.0' set of profiles is the stable, default one.

```
~# eselect profile list
~# eselect profile set <number>
```

Finally, set the USE flags you want in either `/etc/portage/make.conf` (global USE flags) or `/etc/portage/package.use` (local USE flags).

```
~# nano -w /etc/portage/make.conf
```

For those of you who want to run Gentoo Linux with support for international locales, edit `/etc/locale.gen` and specify the locales you want to support. An example of locales are given below. Once set, generate the locale files for your system.

```
~# nano -w /etc/locale.gen
en_US ISO-8859-1
en_US.UTF-8 UTF-8
de_DE ISO-8859-1
de_DE@euro ISO-8859-15

~# locale-gen
```

If you want to know which locales are supported, view the contents of the /usr/share/i18n/SUPPORTED file:

```
# less /usr/share/i18n/SUPPORTED
```

Configuring the Linux Kernel

First select your time zone file from inside /usr/share/zoneinfo and copy it to /etc/localtime. For instance, to use the GMT time zone:

```
~# cp /usr/share/zoneinfo/GMT /etc/localtime
```

Next, install the kernel sources. Gentoo profiles a few kernel packages like vanilla-sources (bare Linux kernel as delivered by the kernel developers) and gentoo-sources (vanilla Linux kernel with patches managed by Gentoo developers).

```
~# emerge gentoo-sources
```

You will find the kernel sources at `/usr/src/linux`. Now continue with building the Linux kernel as discussed in Configuring a Kernel.

Configuring the System

There are three blocks of information we need to configure now:

- file system information (`/etc/fstab`)

- networking information
- system information

To start with the file system information, you need to edit the `/etc/fstab` file. The structure of this file has been discussed before so this shouldn't be an issue (see The mount command). If you want to use labels instead of device locations, use the *`LABEL="..."`* syntax in the first column.

```
/dev/sda1  /boot    ext2  noauto,noatime    0 0
/dev/sda2  /        ext3  defaults,noatime  0 0
/dev/sda3  /home    ext3  defaults,noatime  0 0
/dev/sda4  none     swap  sw                0 0
none       /dev/shm tmpfs defaults          0 0
```

Next, configure your network settings. Start by setting the system host name in `/etc/conf.d/hostname` and then configure the networking settings in `/etc/conf.d/net`. Finally, add your network interface initialization script to the default run level so that networking is automatically started at boot time.

```
~# nano -w /etc/conf.d/hostname
~# nano -w /etc/conf.d/net
~# rc-update add net.eth0 default
```

Also edit your `/etc/hosts` file to include the IP addresses and host names of other systems you might need. Also add your host name to the 127.0.0.1 entry in `/etc/hosts`.

```
~# nano -w /etc/hosts
```

Now, set your root password

```
~# passwd
```

Next, edit `/etc/rc.conf` which contains your general system configuration settings:

```
~# nano -w /etc/rc.conf
```

Next, edit `/etc/conf.d/keymaps` to set your system-wide keyboard layout settings:

```
~# nano -w /etc/conf.d/keymaps
```

Finally, edit `/etc/conf.d/clock` to set the clock options:

```
~# nano -w /etc/conf.d/clock
```

Installing System Tools

Install a system logger, like syslog-ng:

```
~# emerge syslog-ng
~# rc-update add syslog-ng default
```

Install a system scheduler (cron daemon), like vixie-cron:

```
~# emerge vixie-cron
~# rc-update add vixie-cron default
```

Install the file system tools for the file systems you use:

```
~# emerge xfsprogs
~# emerge reiserfsprogs
~# emerge jfsutils
```

Install the necessary networking tools, like a DHCP client:

```
~# emerge dhcpcd
```

Configuring the Boot Loader

Now, we install the GRUB boot loader (remember to use grub-static if you are using an amd64 (x86_64) system with no-multilib):

```
~# emerge grub
```

Once installed, edit the grub configuration file (`/boot/grub/grub.conf`) as we've seen before. Finally, install GRUB on the master boot record:

```
~# grep -v rootfs /proc/mounts > /etc/mtab
~# grub-install --no-floppy /dev/sda
```

Finishing Up

Now that everything is installed, reboot your system by exiting the chroot, umounting all mounted file systems and reboot:

```
~# exit
~# cd
~# umount /mnt/gentoo/boot /mnt/gentoo/dev /mnt/gentoo/proc
~# umount /mnt/gentoo/home /mnt/gentoo
~# reboot
```

Once rebooted (and hopefully inside your Gentoo Linux environment), log in as root and create a user for daily use:

```
~# useradd -m -G users,wheel,audio -s /bin/bash yournick
~# passwd yournick
```

And to remove the traces from the installation, remove the downloaded tarballs from your / file system:

```
~# rm /stage3-*.tar.bz2
~# rm /portage-*.tar.bz2
```

Chapter 16. Introducing the Graphical Environment

Introduction

Linux is often seen as a command-only operating system. This is far from true: although its command-line is a powerful interface, you can also launch graphical environments on Linux. In this chapter, we briefly cover the graphical environments in Linux.

Graphical environments are the de facto standard for working with a workstation. Many users know the Microsoft Windows family or the Apple MacOS series. However, those two aren't the only providers of a graphical environment. When the Intel-compliant PCs were hardly known to the world, consoles and other personal computers already provided a graphical environment to their users.

It comes to no surprise to hear that the free software community also provides graphical environments. And, just like you have choice amongst distributions, you have choice amongst graphical environments: GNOME, KDE, XFCE4 are popular desktop graphical environments; enlightenment, fluxbox, window maker, icewm, ... are window managers.

Although most readers will be sufficiently fluent in using a graphical environment, this book wouldn't be complete if it didn't cover it. As such, and with the danger of being overly simple on the subject, this chapter will briefly cover the concept of graphical environments.

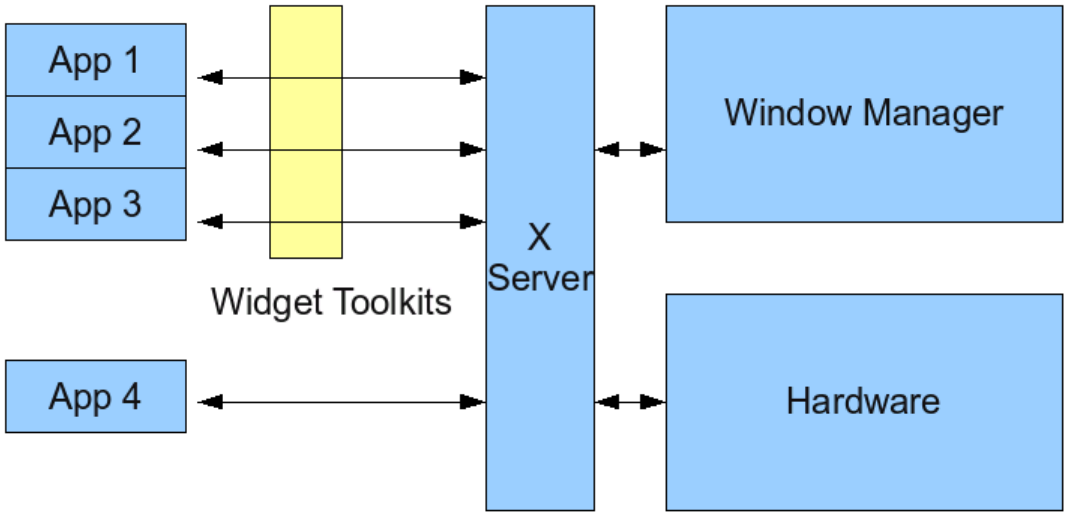
The Structure of X

On Linux, a graphical environment consists of many components:

- Applications
- Widget Tool Kits
- Window Manager
- X Server
- Hardware

Each of those components interacts with others through specific interfaces.

Figure 16.1. A possible representation of how X is structured



An application is able to draw graphic components (buttons, windows, progress bars, labels etc.) through a common API called a *widget tool kit*. Popular widget tool kits on Linux are GTK+ and Qt. However, not all applications require a widget tool kit - they can also talk to the X server immediately. Using such tool kits however facilitates the development of graphical applications.

The widget tool kits communicate with the X server through an interface which basically drives all commands to a *window manager*. A window manager manages the layout of the users' screen: where are the windows positioned, can he drag windows from one location to another, how are buttons rendered, ... Popular window managers are metacity (used by the GNOME desktop environment), KWin (used by the KDE desktop environment), fluxbox, enlightenment, ...

Most window managers are written for specific widget tool kits, but some of their functionality extends beyond one particular window manager: this allows window managers to support not only rendering of applications built with different widget tool kits, but also interoperability between these applications (copy/paste, drag 'n drop ...).

The window manager receives commands from the *X server*. The X server is responsible for turning requests into hardware-specific actions (draw window means to render a window through the graphic card, mouse movements are events coming from the mouse device and directed to the window manager to move the cursor, ...).

In the following sections, we dive a little bit deeper into each of those components...

The X Window System

On a Unix/Linux system, the *X server* is a tool which manages the graphical card on your system and offers services to draw things on your screen. These services are defined in the X11 protocol, an industry open standard. Because the interface is open, many X servers exist, one more powerful than the other. Popular X servers are Xorg and XFree86. However, on Gentoo Linux, Xorg is the only available X server (due to legal restrictions as well as support base).

Installing Xorg

To install Xorg on Gentoo Linux, I suggest to read the X Server Configuration HOWTO [https://wiki.gentoo.org/wiki/Xorg/Configuration] from Gentoo's documentation repository. It describes how to install Xorg, configure it to work with your hardware and more. This chapter only gives a quick introduction to this.

You should understand that the Xorg configuration defines, amongst other things,

- the resolution and refresh rates of your screen(s)
- the language used by your input (keyboard)
- the drivers used to render stuff (i810, vesa, but also closed, proprietary drivers like nVidia and ATIs)
- ...

Once configured to your likings, do not forget to take a backup of your configuration (hint: some people place their X configuration online for others to see - there is nothing personal inside anyway).

Installing Xorg

Before installing Xorg, first make sure that the *VIDEO_CARDS* and *INPUT_DEVICES* variables are set in `/etc/make.conf`:

```
INPUT_DEVICES="evdev keyboard mouse"
VIDEO_CARDS="vesa intel"
```

In the above example, I selected the vesa video driver (a default driver that is supported by most video cards, but with little functionality) and intel video driver (as I have an Intel graphic card).

Next, install x11-base/xorg-server:

```
# emerge x11-base/xorg-server
```

Once finished, it is time to check out the graphical server environment.

Testing Xorg

Try out Xorg without using any configuration file. The Xorg server will try to automatically detect the necessary settings and, to be honest, does a fine job at that. Don't test out things as root though!

```
$ startx
```

If you haven't configured a graphical environment yet, you'll be greeted with a console and an ugly background. However, that alone should suffice to verify if your mouse and keyboard are working as well as do a preliminary verification of the resolution of your screen.

If the graphical server doesn't seem to function properly, make sure to read up on Gentoo's Xorg Server Configuration HOWTO [https://wiki.gentoo.org/wiki/Xorg/Configuration].

Window Managers

Window managers interact with the X server using the X11 interface and manage how your graphical environment looks like, but also how it behaves (for instance, there are window managers that do not support dragging windows).

Certain window managers are accompanied by various other tools that integrate nicely with the window manager. These tools offer services like a panel (from which you can launch commands or

programs immediately), application menus, file manager etc. The aggregation of these tools is often called a *desktop environment* because it offers a complete desktop to the user.

Installing a Window Manager

Gentoo supports many window managers. To install one, simply emerge it.

For fluxbox, a popular, lightweight window manager, Gentoo even has official documentation available: the Fluxbox Configuration HOWTO [http://www.gentoo.org/doc/en/fluxbox-config.xml].

Activating a Window Manager

To activate a window manager for your end user, create a file called `.xinitrc` in your home directory. Inside it, you just add "exec <manager>" where <manager> is the command to launch the window manager.

For instance, for fluxbox:

```
exec fluxbox
```

Desktop Environments

The majority of Linux users use a desktop environment to work with their work station. The two most used desktop environments are KDE and GNOME. The third environment, XFCE4, is gaining momentum as a lightweight yet powerful desktop environment.

GNOME

The GNOME desktop environment is the default desktop environment for many Linux distributions such as Fedora and RHEL. Its desktop is very simple to use: the number of visible options is kept low to not confuse users, and all applications that want to integrate with the GNOME desktop should adhere to various guidelines such as the user interface guideline.

Gentoo has a GNOME Configuration HOWTO [http://www.gentoo.org/doc/en/gnome-config.xml] available as well.

KDE

The KDE desktop is a fully featured desktop environment which offers all the functionality a regular user might expect from his system. KDE comes with many tools, ranging from network related tools (browsers, IM, P2P) to office tools, multimedia tools, authoring and even development environments.

Gentoo provides a KDE Configuration HOWTO [http://wiki.gentoo.org/wiki/KDE].

XFCE4

The XFCE4 desktop is designed to still run smoothly on low memory systems (32 Mbytes and more). Often, power users use XFCE4 even on large memory systems just to reduce the memory overhead of the graphical environment.

Gentoo provides an XFCE Configuration Howto [http://www.gentoo.org/doc/en/xfce-config.xml].

Activating a Desktop Environment

To activate a desktop environment for your end user, create a file called `.xinitrc` in your home directory. Inside it, you just add "exec <environment>" where <environment> is the command to launch the desktop environment.

For instance, for Xfce4:

```
exec xfce4-session
```

Logging on Graphically

If you want to log on to your system using a graphical logon manager, you need to do two things:

- Install a graphical logon manager
- Setup the default graphical environment

Install Graphical Logon Manager

The desktop environments KDE and GNOME provide their own graphical logon manager (which are called kdm and gdm respectively). If you don't have them or want to use a different one, I recommend x11-misc/slim. It is a lightweight graphical logon manager.

```
# emerge x11-misc/slim
```

Once a graphical logon manager is available, configure the xdm service to use it.

In `/etc/conf.d/xdm`:

```
DISPLAYMANAGER="slim"
```

Finally, add the xdm service to the default runlevel.

```
# rc-update add xdm default
```

Setup the Default Graphical Environment

To setup the default graphical environment for a user, you need to create your .xinitrc file as mentioned before (Activating a Window Manager or Desktop Environment).

Supporting 3D Acceleration

The graphical environment can also use 3D acceleration.

Now, 3D acceleration is a tricky subject because there are many implementations that offer 3D services. For instance, you can have 3D services with software rendering (i.e. no delegation of rendering to specific 3D hardware) but this usually isn't seen as 3D acceleration.

When you have a graphic card capable of rendering 3D, you will need to configure the X Window System to hand over 3D rendering tasks to the graphic card. This can happen through either open standards or specifications (such as OpenGL) or through closed, proprietary drivers (such as the nVidia drivers).

Chapter 17. Log File Management

Introduction

If the previous chapters are all read through, you'll most likely have successfully installed Gentoo Linux. However, managing your Linux system is only starting. Within this chapter, we go through how the various Linux services log events on your system and how you can properly manage these log files.

System Logger

The system logger is a service that allows for various software tools to log their events through a unified, standardized interface. For locally running software, these tools can log through the `/dev/log` socket. Services that run remotely can send their events through the network. For a general understanding of logging daemons, having to deal with remotely running software might bring us too far.

/dev/log

The `/dev/log` socket is created by the system logger (which we have very briefly discussed previously) and is made writable for everyone. That is to be expected, as there are many tools that will log through this socket. Every tool that wants to log through the system logger logs a single line at a time inside this socket. The system logger is listening "on the other side" of the socket and will process these log events.

Log Event Meta Information

You will most likely never need to write your own tool that logs through the system logger, yet understanding how a log event is sent is vital if you want to manage your log files properly. The reason for this is because you will want to filter based on criticality and origin of the events, and these are passed on as part of the log event itself.

When a tool wants to log an event, he needs to provide two additional fields:

- the facility where the event is about, and
- the importance level of the event

The *facility* defines the type of the program that is sending out the event. This allows the system logger to properly filter messages, possibly sending them into different log files. Example facilities are authpriv (security/authorization messages), cron (scheduling messages) and kern (kernel messages). A full list of facilities can be obtained through the syslog man page

```
~$ man 3 syslog
```

The *importance level* defines the criticality of the event. The set of supported importance levels is:

Table 17.1. System logger importance levels

DEBUG	Messages used for debugging purposes
INFO	Informational messages
NOTICE	A normal, yet somewhat more important message than informational
WARNING	This message needs your attention
ERROR	Something wicked has occurred and will need further investigation

CRIT	A critical error has occurred, regular operations might be interrupted or run in a degraded mode
ALERT	Immediate action needs to be taken
EMERG	The system is unusable (no, this has nothing to do with **emerge**, Gentoo Portage' installation tool)

Based on these two fields, log messages are then filtered by the system logger into one or more log files.

System Logger Configuration

How a system logger is configured depends on the system logger you use. In this book, I'll focus on the **syslog-ng** logger.

The configuration file for syslog-ng is `/etc/syslog-ng/syslog-ng.conf`. An example configuration is displayed below.

```
@version: 3.0
options {
  stats_freq(43200);
};

source src {
  unix-stream("/dev/log" max-connections(256));
  internal();
  file("/proc/kmsg");
};

destination messages { file("/var/log/messages"); };
destination cron { file("/var/log/cron.log"); };
destination auth { file("/var/log/auth.log"); };

filter f_messages { not facility(cron, auth, authpriv); };
filter f_cron { facility(cron); };
filter f_auth { facility(auth, authpriv); };

filter f_warnplus { level(warn, err, crit, emerg); };

log { source(src); filter(f_cron); filter(f_warnplus); destination(cron); };
log { source(src); filter(f_auth); destination(auth); };
log { source(src); filter(f_messages); destination(messages); };
```

It might be easy to read the configuration file from the bottom up.

- The log entries define where messages come from (source), which filters the system logger applies (filter) and where the resulting messages are stored in (destination)
- The filter entries define what the filters actually do. For instance, the filter f_warnplus only accepts events with an importance level of warn or higher.
- The destination entries define where the events are stored in (the log files)
- The source entry defines where the system logger gets its messages from (which, in this case, is the /dev/log socket, the kernel message interface kmsg and its own internal logging)

This fairly simple example immediately shows how flexible the logs can work. There are many more interesting filters you can apply, such as match() to match regular expressions within the logged event and program() to match log events of a particular tool.

Non-Syslog Log Files

Many tools log through the system logger, but it is not a huge majority. Lots and lots of tools, server software and others have their own logging system. This makes it a bit more difficult to fully manage the log files properly. However, if you know where the log files are, then that's a start.

Xorg Logging

The Xorg server stores it log file at `/var/log/Xorg.0.log`. The trailing 0 denotes that this is of the current/last start. The log file of the start before that is called `Xorg.1.log`, and so on.

Xorg uses the following notations to identify the various criticality levels:

```
Markers: (--) probed, (**) from config file, (==) default setting,
   (++) from command line, (!!) notice, (II) informational,
   (WW) warning, (EE) error, (NI) not implemented, (??) unknown.
```

The Xorg server will automatically rotate the log files, by default 3 times, after which it will erase the oldest log file.

Gentoo Related Logs

Gentoo Portage (and other Gentoo tools) have their own set of logs as well.

Package Installation Logs

The package logs (known as elogs) as defined in the chapter on software management are stored in `/var/log/portage/elog`. The variables in `/etc/make.conf` that define the location and the logging aspects are:

```
# Base directory for Portage logging
PORT_LOGDIR="/var/log/portage"
# Type of logging (save = separate log per installation)
PORTAGE_ELOG_SYSTEM="save"
# Log filter (what to log)
PORTAGE_ELOG_CLASSES="info warn error log"
```

Portage does not clean up old log files though, so we'll need to implement something for ourselves.

Maintaining Log Files with Logrotate

Most tools do not offer log rotation or clean up by default. It is therefore recommended to implement some sort of log rotation for your files. An interesting tool to install for this purpose is app-admin/logrotate. The tool is triggered by your system scheduler (cron) and is best configured by creating separate configuration files for your log files.

Installing Logrotate

First, install logrotate:

```
~# emerge logrotate
```

Next, make sure that it is scheduled to be ran every day. For instance, you can have a script called logrotate.cron inside `/etc/cron.daily` with the following content (Gentoo users will have this as part of the installation):

```
#!/bin/sh
```

```
/usr/sbin/logrotate /etc/logrotate.conf
```

Configuring Logrotate

To configure logrotate, create configuration files inside /etc/logrotate.d. For instance, the following /etc/logrotate.d/syslog-ng helps us rotate the log files that we identified as part of the system logger configuration:

```
/var/log/messages /var/log/cron.log /var/log/auth.log {
  rotate 6
  monthly
  # It is ok if a log file is missing
  missingok
  # Only execute postrotate after all 3 log files are rotated
  # instead of after each log file
  sharedscripts
  # Commands to run after the rotation has occurred
  # This will instruct syslog-ng to reload its configuration
  # and log files
  postrotate
    /etc/init.d/syslog-ng reload > /dev/null 2>&1 || true
  endscript
}
```

The file informs logrotate that the mentioned log files are rotated 6 times on a monthly basis (so you keep 7 months of history). The tool will automatically rename the "old" log files by adding a date stamp to the filename. You can also instruct logrotate to compress the rotated log files or even move them to a different location.

Maintaining Log Files with Cron

Not all log files can be easily managed with logrotate. For instance, you might want to only remove log files that are older than 1 month. This can be easily accomplished with a simple cron entry. For instance, `/etc/cron.weekly/elog-cleanup`:

```
#!/bin/sh

find /var/log/portage/elog -type f -mtime +30 -exec rm '{}' \;
```

Chapter 18. Taking Backups

Introduction

Before we finish this book, lets focus on backups and restores. After all, you don't want to put all that effort in your Linux system, only to see it vanish with the next power surge or daughter-spilled-milk-over-my-laptop incident.

Poor Man's Backup

Not taking any backups is living on the edge. Especially in case of Gentoo Linux, you don't want to lose all the effort you've put in setting up your system. A basic, poor man's backup, is nothing more but an off-line copy of your important files. This can be on a USB stick, an SD card or a CD/DVD. But what to backup?

The following sections describe a possible set of files and directories to back up, without resorting to lots and lots of storage requirements.

System Settings

First of all, take a copy of your `/etc` folder. The `/etc` directory contains all system settings, including your file system table (`fstab`), Portage-specific settings (`/etc/portage`), service configuration files (in `/etc/conf.d` and `/etc/env.d`), ... Really, this location is not to be forgotten.

Next, there are a few non-/etc files or directories that you want to copy as well.

- `/var/lib/portage/world` is your world-file, which contains a list of all software you've installed (well, not their dependencies, but that's not needed anyhow). In case that you ever need to rebuild your system, this file will tell you what needs to be installed
- `/usr/local` contains all non-Portage installed software and settings. Although you might not have anything inside that, those of you that do will most likely take a backup of that as well.
- `/proc/config.gz` (or `/usr/src/linux/.config`) is your kernel configuration. You've probably lost a lot of sweat and oxygen while setting up your kernel configuration, so you definitely don't want to lose it

User Settings

Next to the system settings, you'll find that your end user(s) on your system also have gone through lots of trouble to set up their system as they want it. End users' configuration settings are stored in the end users' home directories. There is no single directory for all configuration settings, so finding which ones to back up and which not might be difficult. However, one way that works for me (and which usually doesn't include the end users' personal files) is to backup all `${HOME}/.[^.]*` files and directories. Most configuration directories (or files) are hidden files (i.e. starting with a period).

Sample Script

The following command will make an archive of the above mentioned settings.

```
~# tar cvzf /var/tmp/poormanbackup.tar.gz /etc /var/lib/portage/world /usr/loca
```

Again, this is just a poor man's backup, because it is not really managed, but it is still better than nothing. Make sure you copy the poormanbackup.tar.gz to a safe(r) place. Note that it will still be fairly large, because of the end user copy (as it contains lots of software caches, like your browsers'

cache). If you drop the end user directories (the `/home/*/.[^.]*`) the file is reduced from many dozens (up to hundreds) of megabytes down to about half a megabyte.

To restore files, get the file on your system (mount the USB stick/CD) and extract the file(s) you need. For instance, to extract all `/etc/portage/*` files:

```
~# tar xvzpf /media/usb/poormanbackup.tar.gz -C / etc/portage
```

Note the space between `/` and `etc/portage`!

More Managed Backups

A slightly more complicated, but probably safer method, is to use tools that provide you with a managed backup solution. Managed means that you tell them what to backup (although they do propose certain backup situations) and they take care of the rest, such as repetitive backups, as well as easy restore activities in case things go wrong.

If you take a look at the packages that Gentoo offers in its app-backup category, you'll find that there are plenty of them. I can't tell you which one is better, because that is a personal taste. But I will provide a few pointers so you can get started.

Backup Ninja

The backupninja application is a tool which is scheduled to run every night (for instance) and that will read in the configuration file(s) in /etc/backup.d. Inside this directory, you define what needs to be backed up, how often, etc. For instance, the same poor man's backup settings of above would result in a file similar to the following:

File `/etc/backup.d/20-basic-system.rdiff` to backup regular files:

```
when = daily
options = --exclude-special-files
nicelevel = 15

[source]
label = myhostname.mydomain
type = local
keep = 60
include = /etc
include = /var/lib/portage/world
include = /usr/local
include = /home/*/.[^.]*
exclude = /home/*/.mozilla/firefox/*.default/Cache

[dest]
type = local
directory = /var/backups/backupninja
```

File `/etc/backup.d/19-kernel-config.sh`:

```
zcat /proc/config.gz > /etc/kernel-config
```

File `/etc/backup.d/21-copy-to-usb.sh`:

```
rsync -avug /var/backups/backupninja /media/usb
```

The result of this is that the kernel configuration file is generated first (into /etc/kernel-config) after which (incremental) backups are taken of the presented locations, and stored in /var/backups/backupninja. Finally, these files are copied (synchronized) to /media/usb (which might be a USB stick

or external disk drive). If you ever want to store that backup outside of your home (for instance, at your parents/kids house), detach the USB stick and store it there. Just make sure another USB stick is attached and mounted for the next backup cycle.

Bare Metal Recovery

When you want to make sure your system is up and running in no time, even when the entire system crashed, you need to take bare metal backups (full system backups) which you can restore quickly. A popular method to accomplish this is to use imaging software tools.

PartImage

If you installed Gentoo using the sysresccd CD, then you already have the tools at your disposal to perform imaging backups. A popular tool is *PartImage*, and is available on the sysresccd.

The tool offers a graphical (or curses) based interface, but you don't need to use that: the tool also supports command-line backups.

An example usage would be the following. Boot from the sysresccd, and attach USB storage to your system on which you want to back up your entire system:

```
~# mount /dev/sdc1 /media/usb
```

Next, take image(s) of your partitions, or take a full disk backup. Don't worry, PartImage is able to detect when disk space isn't used, and will skip those empty blocks. The following takes a full backup of /dev/sda in blocks of 700Mbyte (in case you ever want to store the files on CD or DVD):

```
~# partimage -b -z1 -o V700 save /dev/sda /media/usb/gentoo_full.sda.partimg.gz
```

If you ever need to restore the system, use the following command:

```
~# partimage -e restore /dev/sda /media/usb/gentoo_full.sda.partimg.gz.000
```

Stage4/5/... Installation

Specifically for Gentoo, some users have created enhancements on top of the standard "stage3" installation used by Gentoo Linux. With stage4 or stage5 installations, the extracted tarball is a lot larger (it contains almost an entire system) and might be accompanied with additional script(s) that set up or recover the system. These scripts can, for instance, restore your partition layout, reformat the partitions, reinstall the bootloader, etc.

Compared with the poor man's backup approach described above, this can be seen as a "entire system poor man's backup", with optionally

- Backup & restore of the master boot record and partition table

```
(Backup)
dd if=/dev/sda of=/media/usb/full-stage-backup/mbr.img bs=512 count=1
sfdisk -d /dev/sda > /media/usb/full-stage-backup/sfdisk.dat

(Restore)
dd if=/media/usb/full-stage-backup/mbr.img of=/dev/sda
sfdisk /dev/sda < /media/usb/full-stage-backup/sfdisk.dat
```

- Restoration of the boot loader

```
(Restore: command after restoring /boot content)
grub-install --root-directory=/mnt/gentoo/boot /dev/hda
```

- Reformatting of partitions (be carefull here, this is very error-prone and specific for your system)

```
(Backup)
mount | grep -v 'dev/mapper' | grep -v 'dev/md' | grep 'type ext' | awk -F' '

(Restore)
/media/usb/full-stage-backup/reformat.sh
```

I personally prefer imaging software for such large recovery scenarios. However, such stage4/stage5 installations prove to be useful when you migrate your system from one disk to another. Using imaging tools might fail here, because they often require that the target hard disk is equally sized, whereas you probably want to migrate to a larger disk.

So, if you are interested in a stage4 or stage5 backup/restore, you can take one from a running system or booted from a livecd. I suggest using a livecd (or other live environment) as that will ensure that the restore of the archive will give you a sane situation. If you take such an archive while running inside the system itself, you will also copy state files and other files that should not be there when a system is being booted.

Mount your file systems (say at `/mnt/gentoo`) except for the dynamically created ones (like proc, dev, sys, ...). Then, run the following **tar** command to create the archive:

```
~# tar -cvzf /media/usb/backup/stage4.tar.gz --strip-components=2 /mnt/gentoo
```

By using `--strip-components=2`, all files are stored in the archive without the leading "`/mnt/gentoo`".

If you need to put back the backup, you can recreate your partitions (or create new ones), mount them somewhere (say `/mnt/gentoo` again) and run:

```
~# tar xvzpf /media/usb/backup/stage4.tar.gz -C /mnt/gentoo
```

Don't forget to reinstall the boot loader before you reboot!

Chapter 19. Using A Shell

Introduction

As the Linux basics have been unfolded in the previous chapters, I want to close off this book with an introduction to shell scripting. Shell scripting is one of Linux' (and Unix) powerful features. It allows you to automate various tasks and is often used to help you manage your system. One of Linux' main principles is "keep it simple", and by providing simple tools that do simple jobs (but do them well) you can build powerful systems. The downside of simple tools is that you need plenty of them to do certain tasks, and shell scripts help you glue together these various tools.

Before we launch off our shell scripting, we need to cover a few basic items of running commands in Linux first. Some of them have been described earlier in this book so it might not seem too unfamiliar for most of you.

Chaining Commands

Running Multiple Commands

When you are inside a shell, you can run one or more commands after another. We have seen such use cases previously, for instance to view the content of a directory and then go inside a subdirectory:

```
~$ mkdir Documents
~$ cd Documents
```

In the previous example, multiple commands are ran as single commands one after another. But with a shell, you can execute multiple on a single line. To do so, you separate the commands with a semicolon:

```
~$ mkdir Documents; cd Documents
```

You can add as many commands as you want. The shell will execute each command when the previous one has finished. For instance:

```
~# emerge --sync; layman -S; emerge -uDN @world; revdep-rebuild -p
```

is the equivalent of

```
~# emerge --sync
~# layman -S
~# emerge -uDN @world
~# revdep-rebuild -p
```

Nothing too powerful about that, but wait, there's more. Commands in a Unix environment always finish with a particular return code. This return code tells whoever called the command if the command exited successfully (return code is 0) or not (return code is not 0) and why not. This return code, as you might have guessed, is an integer, usually within the range of 0...255. You can see the return code of a command by showing the special shell variable `$?` (which can be done using **echo**). Let's see a simple example:

```
~$ ls
Documents     tasks.txt     bookmarks.html
~$ echo $?
0
~$ ls -z
ls: invalid option -- 'z'
Try 'ls --help' for more information
```

```
~$ echo $?
2
```

Conditional Execution

The use of return codes allows us to investigate conditional execution. For instance, you want to update Portage and then update your system, but the latter should only start if the update of Portage finished successfully. If you run commands manually, you would wait for the Portage update to finish, look at its output, and then decide to continue with the update or not. By making use of the return code of emerge --sync; we can run the following command only if the synchronization finished with return code 0. Using shell scripting, you can use *&&* as separator between such commands.

```
~# emerge --sync && emerge -uDN @world
```

By using *&&*, you instruct the shell only to execute the following command if the previous one is successful. Now what about when the command finishes with an error?

Well, the shell offers the `||` separator to chain such commands. Its use is less pertinent in regular shell operations, but more in shell scripts (for instance for error handling). Yet, it is still useful for regular shell operations, as the following (perhaps too simple, yet explanatory) example shows:

```
~$ mount /media/usb || sudo mount /media/usb
```

The command sequence tries to mount the /media/usb location. If for any reason this fails (for instance, because the user does not have the rights to mount), retry but using **sudo**.

Input & Output

Now on to another trick that we have seen before: working with output and input. I already explained that we can chain multiple simple tools to create a more useful, complex activity. One of the most often used chaining methods is to use the output of one command as the input of another. This is especially useful for text searching and manipulation. Say you want to know the last 5 applications that were installed on your system. There are many methods possible to do this, one of them is using /var/log/emerge.log (which is where all such activity is logged). You could just open the logfile, scroll to the end and work backwards to get to know the last 5 installed applications, or you can use a command like so:

```
~$ grep 'completed emerge' /var/log/emerge.log | tail -5
1283033552:   ::: completed emerge (1 of 1) app-admin/cvechecker-0.5 to /
1283033552:   ::: completed emerge (1 of 3) dev-perl/HTML-Tagset-3.20 to /
1283033552:   ::: completed emerge (2 of 3) dev-perl/MP3-Info-1.23 to /
1283033552:   ::: completed emerge (3 of 3) app-pda/gnupod-0.99.8 to /
1283033552:   ::: completed emerge (1 of 1) app-admin/cvechecker-0.5 to /
```

What happened is that we first executed grep, to filter out specific string patterns out of /var/log/emerge.log. In this case, grep will show us all lines that have "completed emerge" in them. Yet, the question was not to show all installations, but the last five. So we *piped* the output (that's how this is called) to the **tail** application, whose sole task is to show the last N lines of its input.

Of course, this can be extended to more and more applications or tools. What about the following example:

```
~$ tail -f /var/log/emerge.log | grep 'completed emerge' | awk '{print $8}'
app-admin/cvechecker-9999
dev-perl/libwww-perl-5.836
...
```

In this example, **tail** will "follow" `emerge.log` as it grows. Every line that is added to `emerge.log` is shown by **tail**. This output is piped to **grep**, which filters out all lines but those containing the string 'completed emerge'. The results of the **grep** operation is then piped to the **awk** application which prints

out the 8th field (where white space is a field separator), which is the category/package set. This allows you to follow a lengthy emerge process without having to keep an eye on the entire output of **emerge**.

The | sign passes through the standard output of a process. If you want to have the standard error messages passed through as well, you need to redirect that output (stderr) to the standard output first. This is accomplished using the "2>&1" suffix:

```
~# emerge -uDN @world 2>&1 | grep 'error: '
```

In the above example, all lines (including those on standard error) will be filtered out, except those having the "error: " string in them. But what is that magical suffix?

Well, standard output, standard error and actually also standard input are seen by Unix as files, but special files: you can't see them when the application has finished. When an application has a file open, it gets a *file descriptor* assigned. This is a number that uniquely identifies a file for a specific application. The file descriptors 0, 1 and 2 are reserved for standard input, standard output and standard error output.

The 2>&1 suffix tells Unix/Linux that the file descriptor 2 (standard error) should be redirected (>) to file descriptor 1 (&1).

This also brings us to the redirection sign: >. If you want the output of a command to be saved in a file, you can use the redirect sign (>) to redirect the output into a file:

```
~# emerge -uDN @world > /var/tmp/emerge.log
```

The above command will redirect all standard output to /var/tmp/emerge.log (i.e. save the output into a file). Note that this will not store the error messages on standard error in the file: because we did not redirect the standard error output, it will still be displayed on-screen. If we want to store that output as well, we can either store it in the same file:

```
~# emerge -uDN @world > /var/tmp/emerge.log 2>&1
```

or store it in a different file:

```
~# emerge -uDN @world > /var/tmp/emerge.log 2> /var/tmp/emerge-errors.log
```

Now there is one thing you need to be aware: the redirection sign needs to be directly attached to the output file descriptor, so it is "2>" and not "2 >" (with a space). In case of standard output, this is implied (> is actually the same as 1>).

Grouping Commands

Shells also offer a way to group commands. If you do this, it is said that you create a sub-shell that contains the commands you want to execute. Now why would this be interesting?

Well, suppose that you want to update your system, followed by an update of the file index. You don't want them to be ran simultaneously as that would affect performance too much, but you also don't want this to be done in the foreground (you want the shell free to do other stuff). We have seen that you can background processes by appending a " &" at the end:

```
~# emerge -uDN @world > emerge-output.log 2>&1 &
```

However, chaining two commands together with the background sign is not possible:

```
~# emerge -uDN @world > emerge-output.log 2>&1 &; updatedb &
bash: syntax error near unexpected token ';'
```

If you drop the ";" from the command, both processes will run simultaneously. The grouping syntax comes to the rescue:

```
~# (emerge -uDN @world > emerge-output.log 2>&1; updatedb) &
```

This also works for output redirection:

```
~# (emerge --sync; layman -S) > sync-output.log 2>&1
```

The above example will update the Portage tree and the overlays, and the output of both commands will be redirected to the `sync-output.log` file.

Storing Commands

All the above examples are examples that are run live from your shell. You can however write commands in a text file and execute this text file. The advantage is that the sequence of commands is manageable (if you make a mistake, you edit the file), somewhat documented for the future (how did I do that again?) and easier to use (just a single file to execute rather than a sequence of commands).

A text file containing all this information is called a shell script. As an- example, the following text file will load in the necessary virtualisation modules, configure a special network device that will be used to communicate with virtual runtimes, enable a virtual switch between the virtual runtimes and the host system (the one where the script is ran) and finally enables rerouting of network packages so that the virtual runtimes can access the Internet:

```
modprobe tun
modprobe kvm-intel
tunctl -b -u swift -t tap0
ifconfig tap0 192.168.100.1 up
vde_switch --numports 4 --hub --mod 770 --group users --tap tap0 -d
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables --flush
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
iptables -A FORWARD -i tap0 -o eth0 -s 192.168.100.1/24 ! -d 192.168.100.1/24 -
iptables -A FORWARD -o tap0 -i eth0 -d 192.168.100.1/24 ! -s 192.168.100.1/24 -
```

Can you imagine having to retype all that (let alone on a single command line, separated with ;)

To execute the script, give it an appropriate name (say "enable_virt_internet"), mark it as executable, and execute it:

```
~# chmod +x enable_virt_internet
~# ./enable_virt_internet
```

Note that we start the command using "./", informing Linux that we want to execute the file that is stored in the current working directory.

Now on to the advantages of using shell scripts even more...

Documenting

You can easily document your steps in shell scripts. A comment starts with #, like so:

```
# Load in virtualisation modules
modprobe tun
modprobe kvm-intel
# Setup special networking device for communicating with the virtual environmen
tunctl -b -u swift -t tap0
ifconfig tap0 192.168.100.1 up
vde_switch --numports 4 --hub --mod 770 --group users --tap tap0 -d
# Enable IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
```

```
# Set up "internet sharing" by allowing package forwarding towards the Internet
iptables --flush
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
iptables -A FORWARD -i tap0 -o eth0 -s 192.168.100.1/24 ! -d 192.168.100.1/24 -
iptables -A FORWARD -o tap0 -i eth0 -d 192.168.100.1/24 ! -s 192.168.100.1/24 -
```

Comments can also be added at the end of a line, btw.

Another, special, comment, is one that tells your operating system how the shell script should be called. There are plenty of shells in Linux (bash, bsh, zsh, csh, ...) and there are even scripting languages that also use text files for their content (perl, python, ruby, ...). Linux does not use file extensions to know how to execute or interpret a file. To help Linux determine the correct execution environment, we add a special comment at the beginning of the file (this must be the first line of the script), in this case to inform Linux that this is a bash shell script:

```
#!/bin/sh
# Load in virtualisation modules
modprobe tun
...
```

This line is quite important, and should always start with #! (also called the *shebang*) followed by the interpreter (/bin/bash in our case).

Introduce Loops, Conditionals, ...

A shell script like the above is still quite simple, but also error-prone. If you had your share of issues / incidents with that script, you will most likely start adding error conditions inside it.

First, make sure that we are root. We can verify this by reading the special variable $UID (a read-only variable giving the user id of the user executing the script):

```
#!/bin/sh

if [ $UID -ne 0 ];
then
  echo "Sorry, this file needs to be executed as root!"
  exit 1;
fi

# Load in virtualisation modules
...
```

The [...] set is a test operator. It gives a return code of 0 (statement is true) or non-zero (statement is false). The if ... then ... fi statement is a conditional which is executed when the test operator returns true (0).

Next, we introduce a loop:

```
# Load in virtualisation modules
for MODULE in tun kvm-intel;
do
  modprobe $MODULE
done
```

What the loop does is create a variable called `MODULE`, and give it as content first "tun" and then "kvm-intel". The shell expands the loop as follows:

```
MODULE="tun"
modprobe $MODULE     # So "modprobe tun"
```

```
MODULE="kvm-intel"
modprobe $MODULE     # So "modprobe kvm-intel"
```

Introduce Functions

You can even made the script more readable by introducing special functions.

```
...

# This is a function definition. It is not executed at this point.
load_modules() {
  for MODULE in tun kvm-intel;
  do
    modprobe $MODULE
  done
}

setup_networking() {
  tunctl -b -u swift -t tap0
  ifconfig tap0 192.168.100.1 up
  vde_switch --numports 4 --hub --mod 770 --group users --tap tap0 -d
  echo 1 > /proc/sys/net/ipv4/ip_forward
}

share_internet() {
  iptables --flush
  iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE
  iptables -A FORWARD -i tap0 -o eth0 -s 192.168.100.1/24 ! -d 192.168.100.1/24
  iptables -A FORWARD -o tap0 -i eth0 -d 192.168.100.1/24 ! -s 192.168.100.1/24
}

# Only now are the functions called (and executed)
load_modules
setup_networking
share_internet
```

Functions are often used to help with error handling routines. Say that we want the script to stop the moment an error has occurred, and give a nice error message about it:

```
...

die() {
  echo $* >&2;
  exit 1;
}

load_modules() {
  for MODULE in tun kvm-intel;
  do
    modprobe $MODULE || die "Failed to load in module $MODULE";
  done
}
...
```

If at any time the modprobe fails, a nice error message will occur, like so:

```
~# ./enable_virt_internet
Failed to load in module tun
~#
```

Advanced Shell Scripting

The above is just for starters. Much more advanced shell scripting is possible - shells nowadays have so many features that some scripts are even disclosed as full applications. You will be surprised how many scripts you have on your system. Just for starters, in /usr/bin:

```
~$ file * | grep -c 'ELF '
1326
~$ file * | grep -c script
469
```

So in my /usr/bin folder, I have 1326 binary applications, but also 469 scripts. That is more than one third of the number of binary applications!

Want more?

This is as far as I want to go with a book targeting Linux starters. You should now have enough luggage to make it through various online resources and other books, and still have a reference at your disposal for your day-to-day activities on your (Gentoo) Linux system.

I would like to thank you for reading this document, and if you have any particular questions or feedback, please don't hesitate to contact me at <sven.vermeulen@siphos.be> or on the Freenode and OFTC IRC networks (my nickname is SwifT).

Chapter 20. Tips and Answers to Questions

What is Linux?

1. You can find information about various Linux distributions at DistroWatch [http://www.distrowatch.com]. Also, Wikipedia [http://en.wikipedia.org] has a List of Linux Distributions [http://en.wikipedia.org/wiki/List_of_linux_distributions]. However, you will most likely have to look at the homepage of each distribution to learn how they perform in the fields you find important.

2. A small list of CPU architectures is:

   - For the embedded market, popular architectures are MIPS and ARM
   - For the desktop market, Intel x86 and related architectures (IA-32, x86-64, AMD64) have almost monopolized the market. POWER (formerly known as PowerPC and IBM POWER) has been used on desktop environments (most notably by Apple)
   - For the server market, Sun SPARC and HP PA-RISC are trying to keep their market share, although systems with Intel's Itanium (IA64) are forcefully growing in the server market.
   - For the more specialized market, IBM's Cell architecture (which is actually a mix of POWER4 and *RISC*-based coprocessors) is a nice example (used in Sony's Playstation 3)
   - In the mainframe market (which is almost fully delivered by IBM) the z/Architecture is well known through its use by the IBM zSeries mainframes

3. New kernel releases are made by the kernel maintainers of that particular tree (for instance, the vanilla tree is managed by Linus Torvalds). At this point, the source code for the new kernel is made available tagged with the specific version number (the source code for the Linux kernel is always available, you can even obtain the at that time development version which might be changed the second after you've downloaded it - look for the linux-2.6 git repository).

   Distributions then obtain the source code and start building kernels based on generic kernel configurations, often with additional software code patches applied. The result of the distribution builds are packages containing the Linux kernel together with many kernel modules (dynamically loadable parts of the Linux kernel) which are then tested by many users. This testing is possible because the Linux kernel will not (by default) load a Linux kernel module that isn't needed or where the system doesn't have the necessary hardware.

   When a kernel built has been thoroughly tested, the kernel build is distributed to the distribution users (or, in case of sourcecode based distributions, the patched source code is distributed).

How does Free Software affect Linux?

1. You can find information about GPL at the GNU site [http://www.gnu.org/copyleft/gpl.html].

2. A few examples of operating systems that use the ELF format or where the format is heavily based upon ELF are those used by the Sony PlayStation Portable/2/3 and the Nintendo Wii.

3. Many software projects still support older versions of the software. For instance, at the time of writing, the KDE project still supports version 3.5 even though 4.2 is being developed and 4.1 is considered the latest stable one. All efforts put in the 3.5 series are bugfixes and security fixes but no new features.

Distributions that want to offer a stable software stack tend to use these software versions rather than the latest available ones. Although their users lag behind on features, their software stack is quite stable.

4. Upgrading a distribution means upgrading the packages offered by the distribution. Now, these packages are not heavily depending upon each other: you can upgrade one package without requiring to upgrade all other packages (although perhaps dependencies need to be pulled in). And because this software is freely available on the Internet, there is no license cost attached to it.

   Whenever a distribution makes a new release, it is often accompanied with a list of "new" supported software. Users of that distribution can then just start upgrading their packages to the "new" software without requiring any reinstall.

   Distributions do make new releases often, but this is mostly because the installation media itself (installation CD and tools) are updated to fit the latest hardware available.

The Role of the Community

1. The Gentoo Linux distribution offers the discussion mediums discussed in this chapter:

   - The Gentoo Linux Forums [http://forums.gentoo.org] are heavily used (over a thousand postings a day) web forums
   - Gentoo hosts its mailing lists itself - you can find an overview of the available mailing lists online [http://www.gentoo.org/main/en/lists.xml]
   - On the Freenode IRC network, Gentoo has a few official chat channels, including the generic #gentoo channel

   There is also the official Gentoo Wiki [https://wiki.gentoo.org].

Running Linux

1. Organising your home directory should not be taken lightly. By using a good structure you can easily backup important documents, access files you often need and still keep a good overview.

   For instance, to create the directories as given in the exercise:

   ```
   $ mkdir doc pics work tmp
   ```

   With this simple structure, the most important directory would be doc (personal documents) and perhaps work. You most likely do not want to back up the temporary files (tmp) and the pics folder might require a lower frequency on backups.

   Of course, you should attempt to use a structure you find the most appealing.

2. The **tar** command allows you to group multiple files (and directories) into a single file (archive). It is originally created to allow administrators to back up multiple files on tape (tar is most likely short for "tape archive"). A tar file is not compressed - it is merely a concatenation of the files with additional metadata about the files (such as filename, permissions, ...).

   This is where the **gzip**/**bzip2** compression comes in: these compression methods do not support archiving, so one should first archive the files together in a tar file and then compress the tarfile using **gzip** or **bzip2**. **gzip** is the most used as it offers a fast compression algorithm. **bzip2** is popular too because it has a higher compression rate.

   The combination result of **tar** with **gzip**/**bzip2** creates what is called a *tarball*. These usually have the extension .tar.gz (or .tgz) for gzip, or .tar.bz2 for bzip2.

The fourth compression is provided by the **compress** command. Just like **gzip**/**bzip2** it compresses a single file; its extension is .Z (so a tarball would yield .tar.Z as an extension). **compress** is the oldest method of these four (**compress**, **zip**, **gzip**, **bzip2**) and supported on all Unix and Unix-alike operating systems.

3. Photorec [http://www.cgsecurity.org/wiki/PhotoRec] is a software tool that allows you to recover removed files from a file system. In Gentoo, you can install this tool through app-admin/testdisk.

The Linux File System

1. The command to recursively change a mode can be found in the manual page of chmod:

   ```
   $ man chmod
   ```

   In effect, the command could be:

   ```
   $ chmod -R o-r tmp/test
   ```

   All underlying directories (`test/to`, ...) will be changed as well.

2. The `/tmp` directory is world-writable, but has a specific flag set: the sticky bit. Check out the manual page of chmod to find out why a possible solution to this question would be:

   ```
   $ chmod 1777 tmp
   ```

Working with Processes

1. There are quite a few possibilities to obtain a process id.

   The first one is to obtain an entire listing of all processes and **grep** out those you are interested in. The output's second column then displays the process' ID.

   ```
   $ ps -ef | grep firefox
   ```

   Another method is to use the pidof command. The disadvantage is that you need to know the process name exactly (not just a part of it):

   ```
   $ pidof firefox-bin
   ```

   If you have /proc available, you can search through all /proc/<pid> directories and read out the cmdline file:

   ```
   $ grep firefox /proc/*/cmdline
   ```

2. Although many possibilities exist, two are quite popular: **nohup** and **screen**.

   With **nohup**, you tell the operating system that the process should not be terminated (nohup = no hang-up) when the session is terminated. However, a process launched with nohup can only be put in the foreground as long as the session is running. The moment the session is terminated, the process still remains active but you wont be able to put it back to the foreground. You should see nohup as a means to make a process behave like a daemon.

   With **screen**, you can run processes in named sessions, detach sessions from your terminal and reattach those sessions in a different terminal. The **screen** command is quite popular in command-line environments because you have such flexibility at hand (you can launch a command-line chat on a server inside irssi, detach from your terminal, go elsewhere, log on to the server and reattach to the screen session to continue the chat).

3. A defunct process is a process that cannot communicate with its parent or child for who knows what reason. Unlike zombie processes (who don't really exist), defunct processes still exist but are

just... well... defunct. To remove defunct processes from the system, see if they have running child processes and terminate those first (it happens that a defunct process can recover when the child processes are terminated). If not, terminate its parent process (just like you would for a zombie process).

Configuring a Linux Kernel

1. Many bootloaders support what is called "chaining". When a bootloader is asked to boot an operating system through another bootloader, it hands over the CPU control to the other bootloader. This however requires that the other bootloaders' code is still available (for instance on a certain partition).

   Chaining is frequently used to boot Windows from a Linux boot loader (LILO or GRUB); the Windows boot loaders' code is available on the Windows' partition (so doesn't need to reside in the MBR).

Hardware Support

No exercises for this chapter yet.

Software Management

1. One alternative package manager is called Paludis [http://paludis.pioto.org/], another is pkgcore [http://www.pkgcore.org/].

2. Various locations for USE flag definitions are the system profiles (pointed to by `/etc/make.profile`), `make.conf`, `/etc/portage/package.use` and the environment variable set in the user's session.

3. Gentoo's architecture testing program is brought to life to assist package developers with the time-consuming testing of packages for every supported architecture. Each architecture has its own AT (arch tester) staff. For more information, Google for "gentoo arch testers".

User Management

1. By default, **sudo** logs to the system logger, so you'll find it either on the messages console or general messages file. You can alter this behaviour in **sudo** if you wish (see **man sudoers**).

Network Management

No exercises for this chapter yet.

Service Management

No exercises for this chapter yet.

Storage Management

No exercises for this chapter yet.

System Management

No exercises for this chapter yet.

Introducing the Graphical Environment

No exercises for this chapter yet.

Installing Gentoo Linux

No exercises for this chapter yet.

Glossary

Glossary

RISC
: Reduced Instruction Set Computing, a CPU instruction design that attempts to offer a simple list of instructions to use by software engineers. By limiting the complexity of the instructions, RISC CPUs can be made faster than more complex CPUs clockwize.

CISC
: Complex Instruction Set Computing, a CPU instruction design that attempts to offer a large list of powerful CPU instructions to use by software engineers. The powerful instructions allow engineers to perform complex tasks with a minimum on instructions. The CPU is optimized to perform these instructions faster than most software implementations on RISC CPUs.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z