Slackware Linux Basics

For Slackware Linux 10.1

Daniël de Kok

Slackware Linux Basics: For Slackware Linux 10.1

by Daniël de Kok

Published Tue Jul 19 18:27:09 CEST 2005

Table of Contents

Preface ..... **i**

I. Getting started ..... **i**

- 1. About this book 1
  - Availability 1
  - Conventions 1
    - File names 1
    - Commands 1
    - Screen output 1
    - Notes 1
- 2. An introduction to Slackware Linux 2
  - What is Linux? 2
  - What is GNU/Linux? 2
  - What is Slackware Linux? 2
  - The UNIX philosophy 2
  - Slackware Linux 10.1 features 3
  - Slackware Linux on CD-ROM 3
- 3. Sources of help 4
  - On your system 4
    - Linux HOWTO’s 4
    - Manual pages 4
  - On the internet 5
    - alt.os.linux.slackware 5
- 4. General concepts 6
  - Multitasking 6
    - Introduction 6
    - Processes and threads 6
  - Filesystem hierarchy 7
    - Structure 7
    - Mounting 8
    - Common directories 8
  - Devices 9
    - Introduction 9
    - ATA and SCSI devices 10
- 5. Installing Slackware Linux 11
  - Booting the installation CD-ROM 11
  - Partitioning a hard disk 11
  - Installing Slackware Linux 13
- 6. Custom installation 30
  - Partitioning a hard disk 30
  - Initializing and mounting filesystems 30
  - Installing packages 31
  - Post-install configuration 32
    - fstab 32
    - LILO 32
    - Networking 33
    - Tuning startup scripts 33
    - Configuring the dynamic linker run-time bindings 34
    - Setting the root password 34
    - Setting the timezone 34

Creating the X11 font cache ... 35
Automated installation script ... 35

II. GNU/Linux Basics ... 40

7. The Bash shell ... 41
Introduction ... 41
Starting the shell ... 41
Shell basics ... 41
Executing commands ... 41
Browsing through shell commands ... 41
Completion ... 42
Wildcards ... 42
Matching a string of characters ... 42
Matching single characters ... 43
Matching characters from a set ... 43
Redirections and pipes ... 43
Redirection ... 43
pipes ... 44
8. Files and directories ... 45
Introduction ... 45
The basics ... 45
pwd ... 45
ls ... 45
cd ... 46
mkdir ... 46
cp ... 46
mv ... 47
rm ... 47
Permissions ... 47
A short introduction ... 47
chown ... 48
chmod ... 48
Archives ... 49
Introduction ... 49
Extracting archives ... 49
Creating archives ... 50
Extended attributes ... 51
Introduction ... 51
Installing the necessary utilities ... 51
Showing extended attributes ... 51
Setting extended attributes ... 51
Mounting filesystems ... 52
Introduction ... 52
mount ... 52
umount ... 52
The fstab file ... 53
fs_spec ... 53
fs_file ... 53
fs_vfstype ... 53
fs_mntops ... 53
fs_freq ... 54
fs_passno ... 54

Encrypting and signing files 54
Introduction 54
Generating your private and public keys 54
Exporting your public key 57
Signatures 57
Encryption 58
9. Text Utilities 59
Introduction 59
The basics 59
cat 59
echo 60
wc 60
tr 60
sort 61
uniq 62
10. Process management 63
Introduction 63
Process basics 63
ps 63
kill 64
Advanced process management 64
Background processes 65
Stopping processes 65
Altering priorities 65
III. Editing and typesetting 67
11. LaTeX 68
Introduction 68
Preparing basic LaTeX documents 68
Minimal document structure 68
Generating printable formats 69
Sections 70
Font styles 71
IV. Electronic mail 72
12. Reading and writing e-mail with mutt 73
Introduction 73
Usage 73
Browsing the list of e-mails 73
Reading e-mails 73
Sending e-mails 73
Basic setup 74
Customized headers 74
The sendmail binary 74
Signing/encrypting e-mails 74
Introduction 75
Configuration 75
Usage 75
13. Sendmail 76
Introduction 76
Installation 76
Configuration 76
Working with mc files 76

Using a smarthost ... 77
Alternative host/domain names ... 77
Virtual user table ... 77
14. Spamassassin ... 79
Introduction ... 79
Installing spamassasin ... 79
SpamAssassin ... 79
Starting spamd ... 79
Using spamassassin with procmail ... 80
Marking messages as spam ... 80
Moving spam mail to a separate mailbox ... 80
V. System administration ... 81
15. User management ... 82
Introduction ... 82
Adding and removing users ... 82
useradd ... 82
passwd ... 83
adduser ... 83
userdel ... 84
Avoiding root usage with su ... 84
Restricting su access ... 85
Disk quota ... 85
Introduction ... 85
Enabling quota ... 85
Editing quota ... 86
Getting information about quota ... 87
16. Printer configuration ... 88
Introduction ... 88
Preparations ... 88
Configuration ... 88
Access control ... 89
Ghostscript paper size ... 90
17. X11 ... 91
X Configuration ... 91
Window manager ... 91
18. Package Management ... 93
Pkgtools ... 93
Introduction ... 93
pkgtool ... 93
installpkg ... 93
upgradepkg ... 93
removepkg ... 94
Slackpkg ... 94
Introduction ... 94
Configuration ... 94
Importing the Slackware Linux GPG key ... 94
Updating the package lists ... 95
Upgrading packages ... 95
Installing packages ... 96
Using rsync ... 96
Tagfiles ... 97

Introduction ... 97
Creating tagfiles ... 97
Using tagfiles ... 98
19. Building a kernel ... 100
Introduction ... 100
Configuration ... 100
Compilation ... 101
Installation ... 101
Installing the kernel ... 102
Configuring LILO ... 102
20. System initialization ... 104
The bootloader ... 104
LILO configuration ... 104
LILO installation ... 105
init ... 105
Initialization scripts ... 106
Hotplugging ... 107
Executing scripts on hotplug events ... 107
21. Security ... 110
Introduction ... 110
Closing services ... 110
Introduction ... 110
Finding open ports ... 110
inetd ... 111
22. Miscellaneous ... 112
Scheduling tasks with cron ... 112
Hard disk parameters ... 113
Monitoring memory usage ... 114
vmstat ... 114
VI. Network administration ... 116
23. Networking configuration ... 117
Hardware ... 117
Configuration of interfaces ... 117
Configuration of interfaces (IPv6) ... 118
Introduction ... 118
Slackware Linux IPv6 support ... 119
Adding an IPv6 address to an interface ... 119
Wireless interfaces ... 120
Resolving ... 121
IPv4 Forwarding ... 122
24. IPsec ... 124
Theory ... 124
Kernel configuration ... 124
Installing IPsec-Tools ... 125
Setting up IPsec with manual keying ... 125
Introduction ... 125
Writing the configuration file ... 125
Activating the IPsec configuration ... 127
Setting up IPsec with automatic key exchanging ... 127
Introduction ... 128
Using racoon with a preshared key ... 128

Activating the IPsec configuration ... 130
25. The internet super server ... 131
Introduction ... 131
Configuration ... 131
TCP wrappers ... 131
26. Apache ... 133
Introduction ... 133
Installation ... 133
User directories ... 133
Virtual hosts ... 133
27. BIND ... 135
Introduction ... 135
Delegation ... 135
DNS records ... 135
Masters and slaves ... 135
Making a caching nameserver ... 135

List of Tables

5-1. Alternative kernels ... 11
7-1. Bash wildcards ... 42
8-1. Archive file extensions ... 49
9-1. Special tr character sequences ... 60
11-1. LaTeX document classes ... 69
11-2. LaTeX font styles ... 71
18-1. Tagfile fields ... 97
20-1. Hotplug device classes ... 107
23-1. Important IPv6 Prefixes ... 119
27-1. DNS records ... 135

List of Figures

4-1. Forking a process ... 6
4-2. The filesystem structure ... 7
5-1. The cfdisk parition tool ... 12
5-2. The setup tool ... 13
5-3. Setting up the swap partition ... 13
5-4. Selecting a partition to initialize ... 14
5-5. Formatting the partition ... 14
5-6. Selecting a filesystem type ... 15
5-7. Selecting the source medium ... 16
5-8. Selecting the disk sets ... 16
5-9. Installing the kernel ... 17
5-10. Creating a bootdisk ... 17
5-11. Selecting the default modem ... 17
5-12. Enabling hotplugging ... 18
5-13. Selecting the kind of LILO installation ... 18
5-14. Choosing the framebuffer resolution ... 19
5-15. Adding kernel parameters ... 19
5-16. Choosing where LILO should be installed ... 20
5-17. Configuring a mouse ... 20
5-18. Choosing whether GPM should be started or not ... 21
5-19. Choosing whether you would like to configure network connectivity ... 21
5-20. Setting the host name ... 22
5-21. Setting the domain name ... 22
5-22. Manual or automatic IP address configuration ... 23
5-23. Setting the IP addres ... 23
5-24. Setting the netmask ... 24
5-25. Setting the gateway ... 24
5-26. Choosing whether you want to use a nameserver or not ... 25
5-27. Setting the nameserver(s) ... 25
5-28. Confirming the network settings ... 26
5-29. Enabling/disabling startup services ... 26
5-30. Choosing whether the clock is set to UTC ... 27
5-31. Setting the timezone ... 27
5-32. Choosing the default window manager ... 28
5-33. Setting the root password ... 28

5-34. Finished 29
7-1. Standard input and output........ 43
7-2. A pipeline 44
23-1. The anatomy of an IPv6 address 118
23-2. Router example........ 122

Preface

This book aims to provide an introduction to Slackware Linux. It addresses people who have little or no GNU/Linux experience, and covers the Slackware Linux installation, basic GNU/Linux commands and the configuration of Slackware Linux. After reading this book, you should be prepared to use Slackware Linux for your daily work, and more than that.

Thanks to the rapid development of opensource software, there are now comprehensive desktop environments and applications for GNU/Linux. Most current distributions and books focus on using GNU/Linux with such enviroments. I chose to ignore most of the graphical applications for this book, and tried to focus this book on helping you, as a reader, to learn using GNU/Linux in a more traditional UNIX-like way. I am convinced that this approach is often more powerful, and helps you to learn GNU/Linux well, and not just one distribution or desktop environment. The UNIX philosophy is described in the section called *The UNIX philosophy* in Chapter 2.

I wish everybody a good time with Slackware Linux, and I hope that you will find this book is useful for you.

Daniël de Kok

I. Getting started

Chapter 1. About this book

Availability

This book was written in DocBook/XML, and converted to HTML and PDF with Jade. The latest version of the book is always available from: http://www.slackbasics.org/.

Conventions

This section gives a short summary of the conventions in this book.

File names

File or directory names are printed as: `/path/to/file`. For example: `/etc/fstab`

Commands

Commands are printed as bold text. For example: **ls -l**

Screen output

Screen output is printed like this:

```
Hello world!
```

If commands are being entered in the screen output the commands will be printed as bold text:

```
$ command
Output
```

If a command is executed as root, the shell will be displayed as “#”. If a command is executed as a normal non-privileged user, the shell will be displayed as “$”.

Notes

Some sections contain extra notes. It is not necessary to know the information in notes, but notes may provide valuable information, or pointers to information. Notes are printed like this:

> **Note:** This is a note.

Chapter 2. An introduction to Slackware Linux

What is Linux?

Linux is a UNIX-like kernel which is written by Linus Torvalds and other developers, who communicate using the internet. Linux runs on many different architectures, for example on many IA32, IA64, Alpha, m68k, SPARC and PowerPC machines. The latest kernel and more information can be found at: http://www.kernel.org

Linux is often confused with the GNU/Linux system. Linux is only a kernel, not a complete operating system. GNU/Linux consists of the GNU operating system with the Linux kernel. Please read the following section for a more detailed explanation of GNU/Linux.

What is GNU/Linux?

At the beginning of the eighties Richard Stallman started an ambitious project with the goal to write a free UNIX-like operating system. The name of this system is GNU (GNU is Not UNIX). At the beginning of the nineties most important components of the GNU oparating system were written, except for the kernel, which is still under development under the name HURD. HURD consists of some servers which provide UNIX-like kernel functionality. In turn these servers run under the Mach microkernel. At the beginning at the nineties the HURD team still had to wait till the Mach sources were released as free software. In the meanwhile Linus started filling the gap with the Linux kernel. GNU/Linux thus refers to the GNU system running on the Linux kernel. Right now the HURD kernel is also in a usable state and can be downloaded in the form of the GNU/HURD operating system. The Debian (http://www.debian.org/) project has even developed a version of the GNU operating system which works with the NetBSD (http://www.netbsd.org/) kernel. We should call “Linux distributions” “GNU/Linux distributions”, because GNU is a substantial part of most distributions.

What is Slackware Linux?

Slackware Linux is a GNU/Linux distribution which is maintained and developed by Patrick Volkerding. In contrast to many other distributions Slackware Linux adheres to the so-called KISS (keep it simple stupid) principle. This means that Slackware Linux does not have complex graphical tools for configuring a system. For newbies this can be somwhat harsh, but it provides more transperancy and flexibility. Besides that you will get to learn GNU/Linux to the bones with Slackware Linux.

Another distinguishing aspect of Slackware Linux, that also “complies” with the KISS principe is the Slackware Linux package manager. Slackware Linux does not have complex package manager like RPM. Packages are normal tgz (tar/gzip) files, mostly with an additional installation script and a package description. Tgz is much more powerful than RPM for novice users and avoids dependency problems. Another famous feature of Slackware Linux are the BSD-like initialization scripts of Slackware Linux. Slackware Linux has one initialization script for each runlevel insteas of a script for eacht daemon. It allow you to tweak with your system easily, without the need to write net init scripts yourself.

The packages in Slackware Linux are compiled with as little modifications as possible. This means you can use most general GNU/Linux documentation.

The UNIX philosophy

Traditionally, UNIX had a distict philosophy that made it widely loved. Doug McIlroy summarized the UNIX philosophy in a few simple rules:

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Odds are that you do not intend to write programs for GNU/Linux, though even as a user these basic UNIX rules can mean a lot to you. Once you get to know the basics commands that have been part of UNIX for many years, you will be able to combine simple programs to solve complex problems. It helps if you keep this in mind while you learn Slackware Linux and try to get a feeling of how you can divide complex problems in simple combined operations.

Slackware Linux 10.1 features

- *Linux 2.4.29* - Slackware Linux uses the proven 2.4 Linux kernel as the default kernel. Slackware Linux 10.1 provides Linux 2.6.10 as an option. When a 2.6 kernel is booted, Slackware Linux will automatically use *udev*, which is a daemon that automatically generates `/dev` device entries.
- *X11R6.8.1* - This is the second Slackware Linux to provide the X Window System provided by the X.org Foundation.
- *GCC 3.3.4* - Version 3.3.4 of the GNU Compiler Collection is provided. GCC provides C, C++, Objective-C, Fortran-77, and Ada 95 compilers.
- *The K Desktop Environment (KDE) 3.3.2* - The full KDE environment is provided, which includes KOffice, the Konqueror web browser, multimedia programs, development tools, and many more useful applications.
- *The GNU Network Object Model Environment (GNOME) 2.6.1* - GNOME is a popular GTK2 based desktop environment.

Slackware Linux on CD-ROM

Slackware Linux can be purchased at quite many (internet) shops. It is important to make a distinction between the official CD-ROM set and cheap copies. When you buy the official CD set you are financially supporting the development of Slackware Linux. So, if you would like to see continuing development of Slackware Linux, buy the CD set! The Slackware Store can be found at: http://store.slackware.com/

Chapter 3. Sources of help

On your system

Linux HOWTO's

The famous Linux HOWTOs are a collection of documents which cover specific parts of a GNU/Linux system. Most HOWTOs are distribution independent, and therefore very useful for using them with Slackware Linux. The "linux-howtos" package in from the “f” disk set contains the HOWTO collection. After installing this package the HOWTOs can be found in the `/usr/doc/Linux-HOWTOs/` directory. Slackware Linux also contains a collection of Mini-HOWTOs, which are shorter and cover narrower topics. The Mini-HOWTOs can be found in the `/usr/doc/Linux-mini-HOWTOs/` directory after installing the “linux-mini-howtos” package.

Manual pages

Most UNIX-like commands are covered by the a traditional *nix help system called the manual pages. You can read the manual page of a program by using the **man** command. Executing **man** with the name of the command as a parameter shows the manual page for that command. For example:

```
$ man ls
```

If you do not know the exact name of a manual page or command, you can search a page using the keyword (*-k*) parameter:

```
$ man -k routing
NETLINK_ROUTE [rtnetlink] (7)  - Linux IPv4 routing socket
netstat              (8)  - Print network connections, routing tables, interface statist
route                (8)  - show / manipulate the IP routing table
routed               (8)  - network routing daemon
rtnetlink            (7)  - Linux IPv4 routing socket
```

We have to add that there are also manual pages that cover other things than commands. These sections of manual pages are available:

```
1   Executable programs or shell commands
2   System calls (functions provided by the kernel)
3   Library calls (functions within program libraries)
4   Special files (usually found in /dev)
5   File formats and conventions eg /etc/passwd
6   Games
7   Miscellaneous  (including  macro  packages and conven-
    tions), e.g. man(7), groff(7)
8   System administration commands (usually only for root)
9   Kernel routines [Non standard]
```

If you would like to print a manual page to the default printer that you have set up, you can pipe the output of **man** to the **lpr**. If the *-t* of **man** is used **man** will output the manual page in Postscript format, rather than ASCII. For example, you could use the following command to print the **tar** manual page:

```
$ man -t tar | lpr
```

On the internet

alt.os.linux.slackware

alt.os.linux.slackware is a Slackware Linux newsgroup. You can read newsgroups with a newsreader like tin or knode. Be careful: it is expected that you have read all necessary documentation before posting to this newsgroup. If you have not the chance of getting “flamed” is really big.

Chapter 4. General concepts

This chapter gives an introduction to some general UNIX and GNU/Linux concepts. It is a good idea to read this chapter thoroughly if you do not have any UNIX or GNU/Linux experience. Many concepts covered in this chapter are used in this book and in GNU/Linux.

Multitasking

Introduction

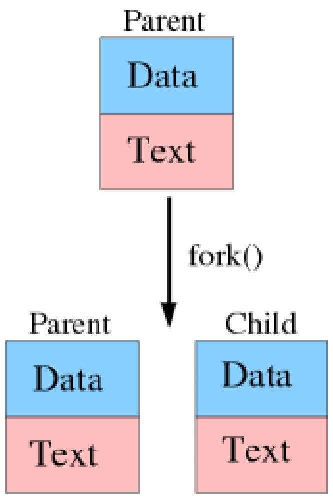
One of Linux' strengths is multi-tasking. Multi-tasking means that multiple processes can run at the same time. You might wonder why this is important to you, because most people are using one application at a time. Besides the more obvious reason that it is just handy to browse while you have a word processor running in the background, multi-tasking is a bare necessity for UNIX-like systems. Even if you have launched no applications there are a bunch of processes running in the background. Some processes might provide network services, others sit there showing a login prompt on other consoles, and there is even a process that executes scheduled tasks. These processes that are running in the background are often called *daemon* (not to be confused with the word demon, a daemon is a protective angel). At a later stage we are going to look at how you can move processes to the background yourself (see Chapter 10).

Note: Note that on single-processor systems processes are not really running simultaneously. In reality a smart scheduler in the kernel is dividing CPU time between processes, giving the illusion that processes are running simultaneously.

Processes and threads

Applications run as one or more processes. To see what a process actually is we need to know what it consists of. Every process basically has two areas; an area that is named *text* and an area that is named *data*. The *text* area is the actual program code; it is used to tell the computer what to do. The *data* area is used to store information that the process has to keep. The operating system makes sure that every process gets its time to execute. New processes can be created by duplicating a running process with a system call named *fork*. Figure 4-1 shows a fork in action schematically. The parent process issues a *fork()* call, and as a result a new process is created.

Figure 4-1. Forking a process



The problem with processes is that they can get quite large, and that is not very efficient for computers with more than one processor. This problem is solved by duplicating the *text* area of a process. So, a threaded process is basically a program that has multiple instances of its executing code running, but these instances share the data area of the process (unlike a fork). These threads can be divided over multiple CPUs, making it possible to run one process on more than one CPU simultaneously.

Filesystem hierarchy

Structure

Operating systems store data in filesystems. A filesystem is basically a collection of directories that hold files, like the operating system, user programs and user data. In GNU/Linux there is only one filesystem hierarchy, this means GNU/Linux doesn’t have multiple drives (e.g. A:, C:, D:), like Windows. The filesystem looks like a tree, with a root directory (`/`) which has no parent directory, branches, and leaves (directories with no subdirectories). Directories are separated using the “/” character.

Figure 4-2. The filesystem structure

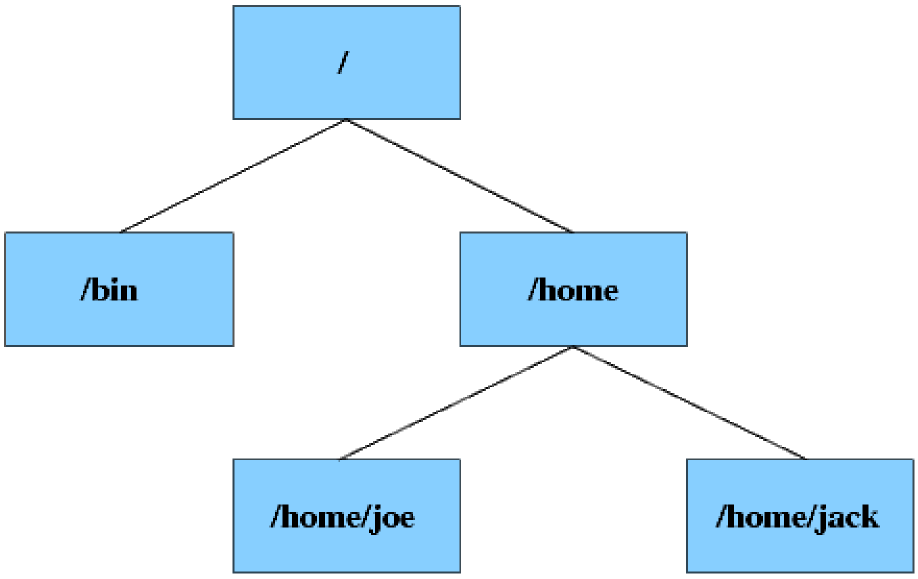


Figure 4-2 shows the structure of a filesystem. You can see that the root directory (`/`) has two child directories; `bin` and `home`. The `home` directory has two child directories, `joe` and `jack`. The diagram shows the full pathname of each directory. The same notation is used for files. Suppose that there is a file named `memo.txt` in the `/home/jack` directory, the full path of the file is `/home/jack/memo.txt`.

Each directory has to special entries, “`.`”, and “`..`”. “`.`” refers to the same directory, “`..`” to the parent directory. These entries can be used for making relative paths. Suppose that you are working in the `jack` directory. From this directory you can reference to the `joe` directory with `../joe`.

Mounting

You might have started to wonder how it is possible to access other devices or partitions than the hard disk partition which holds the root filesystem. Linux uses the same approach as UNIX for accessing other mediums. Linux allows the system administrator to connect a device to any directory in the filesystem structure. This process is named “mounting”. For example, one could mount the CD-ROM drive to the `/cdrom` directory. If the mount was correct, the files on the CD-ROM can be accessed through this directory. The mounting process is described in detail in the section called *Mounting filesystems* in Chapter 8.

Common directories

The Filesystem Hierarchy Standard Group has attempted to create a standard that describes which directories should be available on a GNU/Linux system. Nowadays most major distributions use the Filesystem Hierarchy Standard (FHS) as a guideline. This section describes some mandatory directories on GNU/Linux systems.

Please note that GNU/Linux does not have a separate directory for each application (like Windows), files are ordered by function and type. For example, the binaries for most common user programs are stored in `/usr/bin`, and their libraries in `/usr/lib`. This is a short overview of important directories:

- */bin*: essential user binaries that should still be available in case the `/usr` is not mounted.
- */dev*: device files. These are special files used to access certain devices.
- */etc*: the `/etc` directory contains all important configuration files.
- */home*: contains home directories for individual users.
- */lib*: essential system libraries (like glibc), and kernel modules.
- */root*: home directory for the *root* user.
- */sbin*: essential binaries that are used for system administration.
- */tmp*: a world-writable directory for temporary files.
- */usr/X11R6*: the X Window System.
- */usr/bin*: stores the majority of the user binaries.
- */usr/lib*: libraries that are not essential for the system to boot.
- */usr/sbin*: non-essential system administration binaries.
- */var*: variable data files, like logs.

Devices

Introduction

In UNIX and Linux almost everything is represented as a file, including devices. Each GNU/Linux system has a a directory with special files, named `/dev`. Each file in the `/dev` directory represents a device. You might wonder how this is done; a device file is a special file because it has two special numbers, the *major* and the *minor* number. The kernel knows which device a device file represents by these numbers. The following example shows these numbers for a device:

```
$ ls -l /dev/zero
crw-rw-rw-    1 root     root       1,   5 Apr 22  2003 /dev/zero
```

The **ls** lists files and information about files. In this example information about the `/dev/zero` device is listed. This particular device has *1* as the major device number, and *5* as the minor device number.

> **Note:** If you have the kernel sources unpacked after installing Slackware Linux, you can find a comprehensive list of all major devices with their minor and major numbers in `/usr/src/linux/Documentation/devices.txt`. An up-to-date list is also available online at: ftp://ftp.kernel.org/pub/linux/docs/device-list/

For the Linux kernel there are two types of devices: *character* and *block* devices. Character devices can be read byte by byte, block devices can not. Block devices are read per block (for example 4096 bytes at a time). Whether a device is a character or block device is determined by the nature of the device. For example, most storage media are block devices, and most input devices are character devices. Block devices have one distinctive advantage, namely that they can be cached. This means that commonly read or written blocks are stored in a special area of the system memory, named the cache. Memory is much faster than most storage media, so it is a huge performance benefit to perform read and write operations on commonly used blocks in memory. Of course, eventually changes have to be written to the storage medium.

ATA and SCSI devices

There are two categories of devices that we are going to look into in detail, because understanding the naming of these devices can be crucial for partitioning a hard disk, or for mounting. Almost all modern computers use ATA hard disks and CD-ROMs. Under Linux these devices are named in the following way:

```
/dev/hda - master device on the first ATA channel
/dev/hdb - slave device on the first ATA channel
/dev/hdc - master device on the second ATA channel
/dev/hdd - slave device on the second ATA channel
```

On most computers the hard disk is the master device on the first ATA channel (`/dev/hda`), and the CD-ROM the master device on the second ATA channel. Hard disk partitions have the device name plus a number. For example, `/dev/hda1` is the first partition on the `/dev/hda` disk.

Most average PCs do not have SCSI hard disks or CD-ROM drives, but SCSI is often used for USB drives. Besides that Serial ATA (SATA) drives are also made available as SCSI disks. For SCSI drives the following notation is used:

```
/dev/sda - First SCSI disk
/dev/sdb - Second SCSI disk
/dev/sdc - Third SCSI disk
/dev/scd0 - First CD-ROM
/dev/scd1 - Second CD-ROM
/dev/scd2 - Third CD-ROM
```

Partitions are notated in the same way as ATA disks; `/dev/sda1` is the first partition on the first SCSI disk.

Chapter 5. Installing Slackware Linux

Booting the installation CD-ROM

The easiest method for booting the installation system is by using the installation CD-ROM. The Slackware Linux installation CD-ROM is a bootable CD, which means that the BIOS can boot the CD, just like it can boot, for example, a floppy disk. Most modern systems have a BIOS which supports CD-ROM booting.

If the CD is booted when you have the CD inserted in the CD-ROM drive during the system boot, the boot sequence is probably not correctly configured in the BIOS. Enter the BIOS setup (usually by this can be done by holding the <Del> or <Esc> key when the BIOS screen appears) and make sure the CD-ROM is on the top of the list in the boot sequence. If you are using a SCSI CD-ROM you may have to set the boot sequence in the SCSI BIOS instead of the system BIOS. Consult the SCSI card manual for more information.

When the CD-ROM is booted, a pre-boot screen will appear. Normally you can just press <ENTER> to proceed loading the default (`bare.i`) Linux kernel. If want to install Slackware Linux to a disk that is attached to a SCSI or SATA controller, you have to boot an alternative kernel. You can boot an alternative kernel by entering the kernel name on the prompt, and pressing <ENTER>. The following table lists some of the alternative kernels, for a full list of kernels, refer to the `Slackware-HOWTO` file on the CD-ROM.

Table 5-1. Alternative kernels

Kernel	Description
adaptec.s	This kernel provides support for most Adaptec SCSI controllers, including VLB and EISA models aside from the more modern PCI controllers.
bareacpi.i	This kernel is similar to the *bare.i* kernel, but provides support for the Advanced Configuration and Power Interface (ACPI).
ataraid.i	This kernel provides support for some commonly used ATA RAID controllers.
sata.i	This kernel is similar to the *bare.i* kernel, but provides support for commonly used SATA controllers.
scsi.s/scsi2.s/scsi3.s	These kernels provide support for various SCSI controllers. Refer to the `Slackware-HOWTO` on the CD-ROM for a list of controllers supported by each kernel.

After booting the installation system, you will be asked whether you are using a special (national) keyboard layout or not. If you have a normal US/International keyboard, which are the most common, you can just press <Enter> at this question. After that the login prompt will appear. Log on as “root”, no password will be requested. After logging on the shell is started, and you can start installing Slackware Linux. The installation procedure will be explained briefly in this chapter.

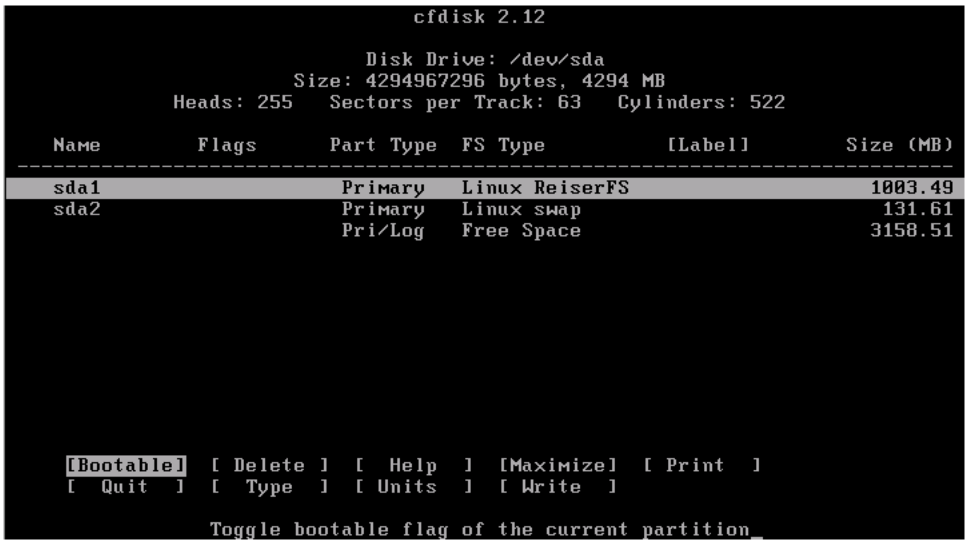
Partitioning a hard disk

Installing Slackware Linux requires at least one Linux partition, creating a swap partition is also recommended. To be able to create a partition there has to be free unpartitioned space on the disk. There are some programs that can resize partitions. For example, FIPS can resize FAT partitions. Commercial programs like Partition Magic can also resize other partition types.

After booting the Slackware Linux CD-ROM and logging on, there are two partitioning programs at your disposal: **fdisk** and **cfdisk**. **cfdisk** is the easiest of both, because it is controlled by a menu interface. This section describes the **cfdisk** program.

To partition the first harddisk you can simply execute **cfdisk**. If you want to partition another disk or a SCSI disk you have to specify which disk you want to partition (**cfdisk /dev/device**). An ATA hard disks have the following device naming: `/dev/hdn`, “n” is replaced by a character. E.g. the “primary master” is named `/dev/hda`, the “secondary slave” is named `/dev/hdd`. SCSI disks are named in the following way: `/dev/sdn`, “n” is replaced by the device character (the first SCSI disk = a, the fourth SCSI disk = d).

Figure 5-1. The cfdisk parition tool



```
                              cfdisk 2.12

                          Disk Drive: /dev/sda
                    Size: 4294967296 bytes, 4294 MB
          Heads: 255   Sectors per Track: 63   Cylinders: 522

    Name        Flags      Part Type  FS Type          [Label]        Size (MB)
 ------------------------------------------------------------------------------
    sda1                    Primary   Linux ReiserFS                    1003.49
    sda2                    Primary   Linux swap                         131.61
                            Pri/Log   Free Space                        3158.51








     [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
     [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]

                 Toggle bootable flag of the current partition_
```

After starting **cfdisk** currently existing partitions are shown, as well as the amount of free space. The list of partitions can be navigated with the “up” and “down” arrow keys. At the bottom of the screen some commands are displayed, which can be browsed with the “left” and “right” arrow keys. A command can be executed with the <Enter> key.

You can create a Linux partition by selecting “Free Space” and executing the “New” command. **cfdisk** will ask you whether you want to create a primary or logical partition. The number of primary partitions is limited to four. Linux can be installed on both primary and logical partitions. If you want to install other operating systems besides Slackware Linux that require primary partitions, it is a good idea to install Slackware Linux onto a logical partition. The type of the new partition is automatically set to “Linux Native”, so it is not necessary to set the partition type.

The creation of a swap partition involves the same steps as a normal Linux partition, but the type of the partition has to be changed to “Linux Swap” after the partition is created. The suggested size of the swap partition depends on your own needs. The swap partition is used to store programs if the main (RAM) memory is full. If you have a harddisk of a reasonable size, it is a good idea to make a 256MB or 512MB swap partition, which should be enough for normal usage. After creating the partition the partition type can be changed to “Linux Swap”, by selecting the “Type” command. The

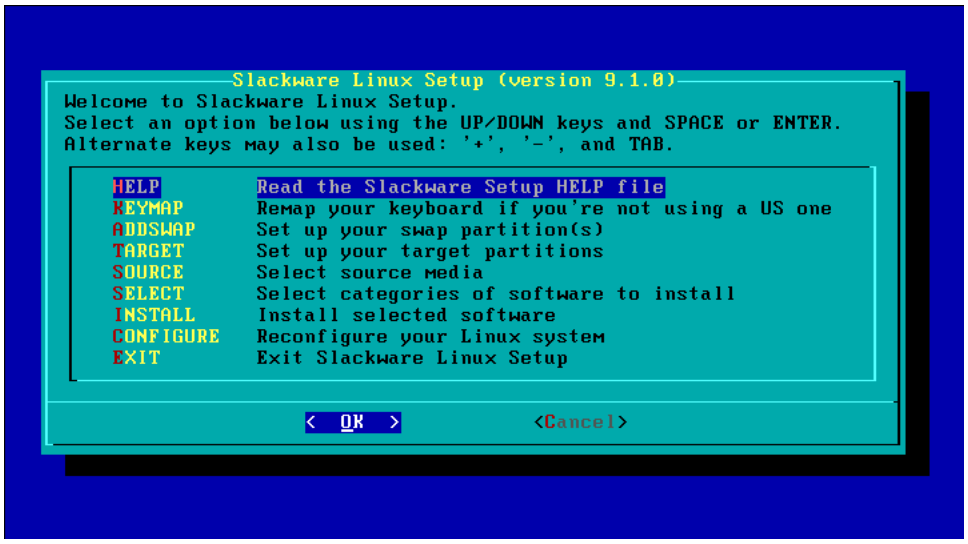
cfdisk program will ask for the type number, “Linux Swap” partitions have type number 82. Normally number 82 is already selected, so you can go ahead by pressing the <Enter> key.

If you are satisfied with the partitioning you can save the changes by executing the “Write” command. This operation has to be confirmed by entering **yes**. After saving the changes you can quite **cfdisk** with the **Quit** command. It is a good idea to reboot the computer before starting the installation, to make sure that the partitioning changes are active. Press <ctrl> + <alt> + <del> to shut Linux down and restart the computer.

Installing Slackware Linux

The Slackware Linux installer is started by executing **setup** in the installation disk shell. Setup will show a menu with several choices, you can see a screenshot of the installer in Figure 5-2. Every option has to be completed to do a complete Slackware Linux installation, but once you start the **setup** guide will guide you through the options.

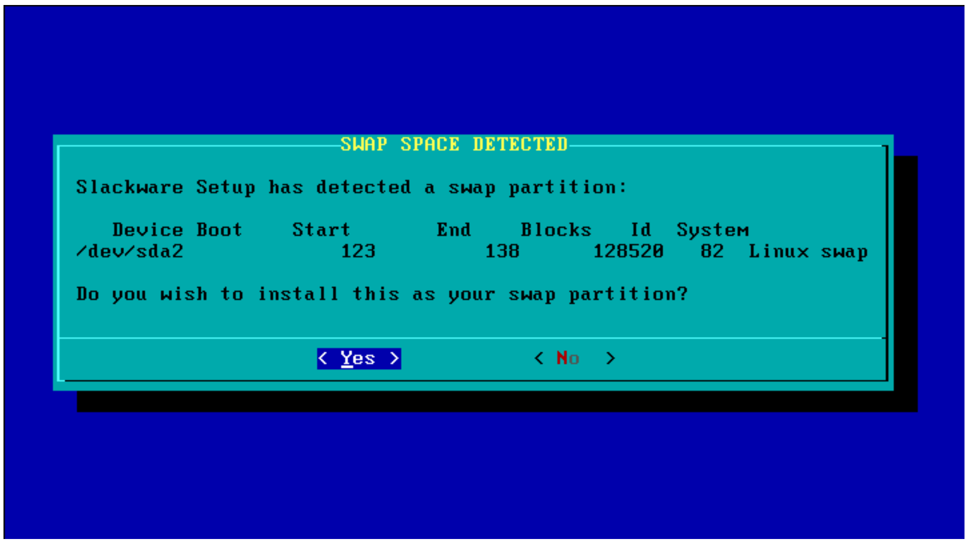
Figure 5-2. The setup tool



```
Slackware Linux Setup (version 9.1.0)
Welcome to Slackware Linux Setup.
Select an option below using the UP/DOWN keys and SPACE or ENTER.
Alternate keys may also be used: '+', '-', and TAB.

    HELP        Read the Slackware Setup HELP file
    KEYMAP      Remap your keyboard if you're not using a US one
    ADDSWAP     Set up your swap partition(s)
    TARGET      Set up your target partitions
    SOURCE      Select source media
    SELECT      Select categories of software to install
    INSTALL     Install selected software
    CONFIGURE   Reconfigure your Linux system
    EXIT        Exit Slackware Linux Setup

                 <  OK  >           <Cancel>
```

The first part of the installation is named “ADDSWAP”. The **setup** tool will look for a partition with the “Linux Swap” type, and ask you if you want to format and activate the swap partition (see figure Figure 5-3). Normally you can just answer “Yes”.

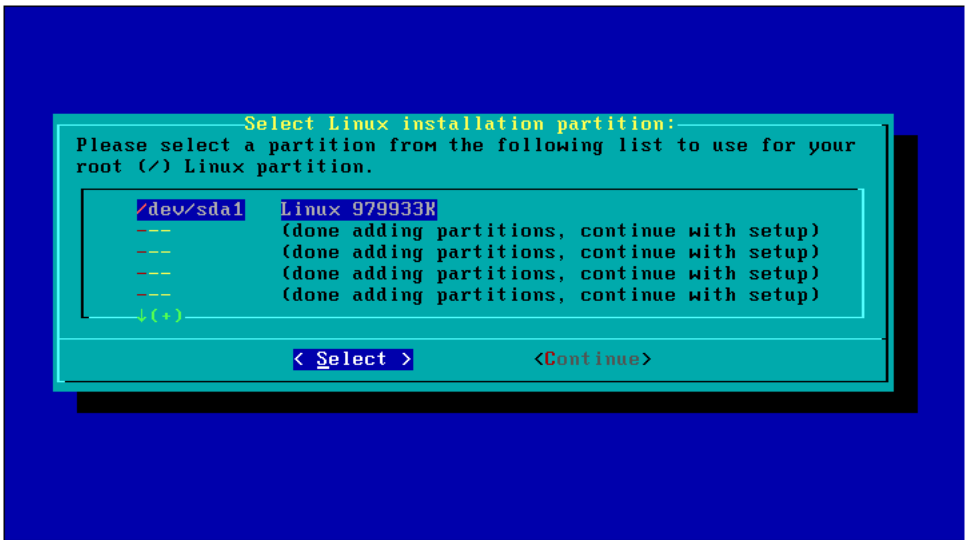
Figure 5-3. Setting up the swap partition



```
SWAP SPACE DETECTED

Slackware Setup has detected a swap partition:

   Device Boot     Start        End     Blocks    Id  System
/dev/sda2              123        138       128520   82  Linux swap

Do you wish to install this as your swap partition?
```

After setting the swap up space the “TARGET” menu is launched, which you can see in Figure 5-4. It is used to initialize the Slackware Linux partitions. Setup will display all partitions with the “Linux native” type.

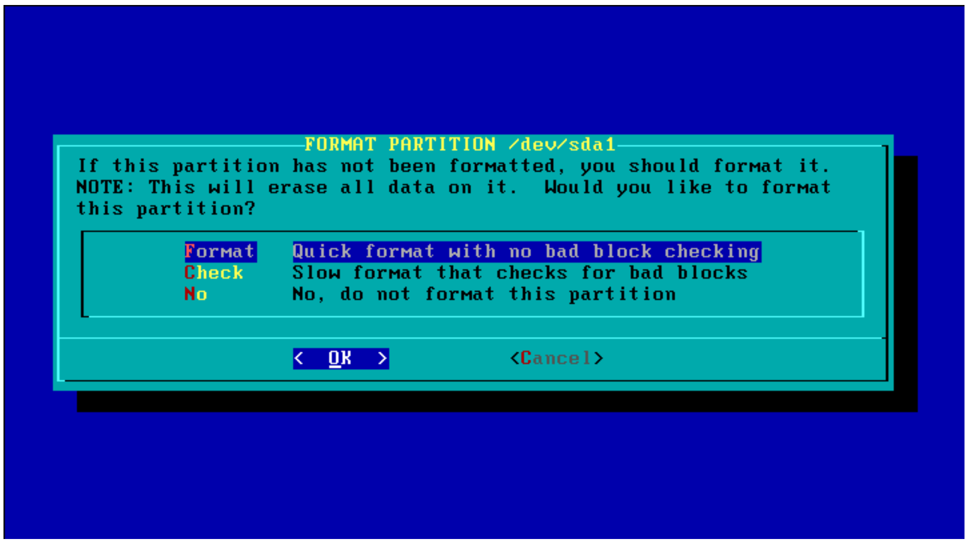
Figure 5-4. Selecting a partition to initialize



```
Select Linux installation partition:
Please select a partition from the following list to use for your
root (/) Linux partition.

    /dev/sda1    Linux 979933K
    ---          (done adding partitions, continue with setup)
    ---          (done adding partitions, continue with setup)
    ---          (done adding partitions, continue with setup)
    ---          (done adding partitions, continue with setup)
    ↓(+)
```

After selecting one partition, the setup tool will ask whether you want to format a partition or not, and if you want to format it, whether you want to check the disk for bad sectors or not (Figure 5-5). Checking the disk can take a lot of time.

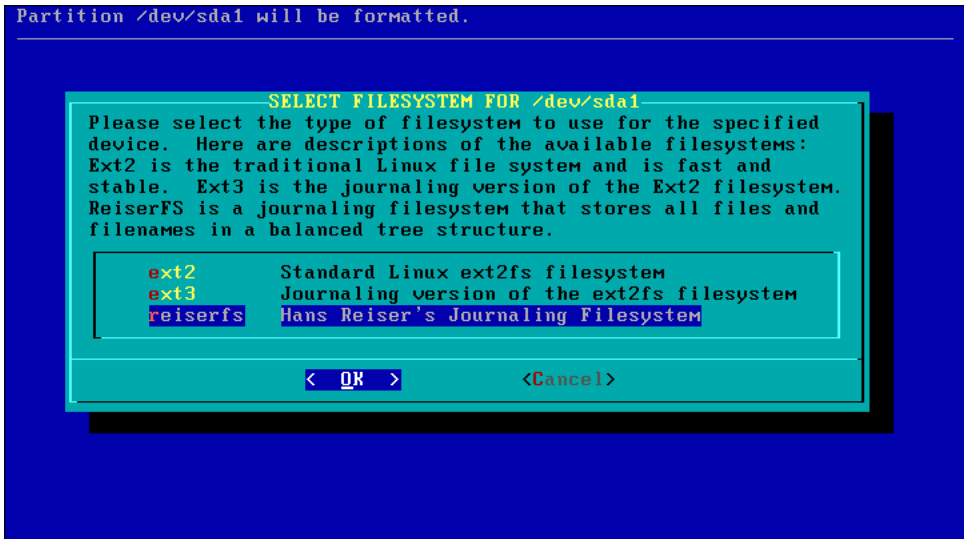
Figure 5-5. Formatting the partition



```
FORMAT PARTITION /dev/sda1
If this partition has not been formatted, you should format it.
NOTE: This will erase all data on it.  Would you like to format
this partition?

         Format    Quick format with no bad block checking
         Check     Slow format that checks for bad blocks
         No        No, do not format this partition

                  <  OK  >          <Cancel>
```

After selecting whether you want to filesystem or not, you can specify which filesystem should be used (Figure 5-6). Normally you can choose the ext2, ext3 and reiserfs filesystems. Ext2 was the standard Linux filesystem for many years, the disadvantage is that Ext2 does not support journaling. A journal is a special file or area of a partition in which all filesystem operations are logged. When the system crashes the filesystem can be repaired rapidly, because the kernel can use the log to see what disk operations were performed. Ext3 is the same filesystem as Ext2, but adds journaling. Reiserfs is a newer filesystem, that also provides journaling. Besides that Reiserfs uses balanced trees, which make many filesystem operations, especially when you are working with many small files, faster than with Ext2 or Ext3. A disadvantage is that Reiserfs is newer, that is why it can be a bit more unstable.

Figure 5-6. Selecting a filesystem type

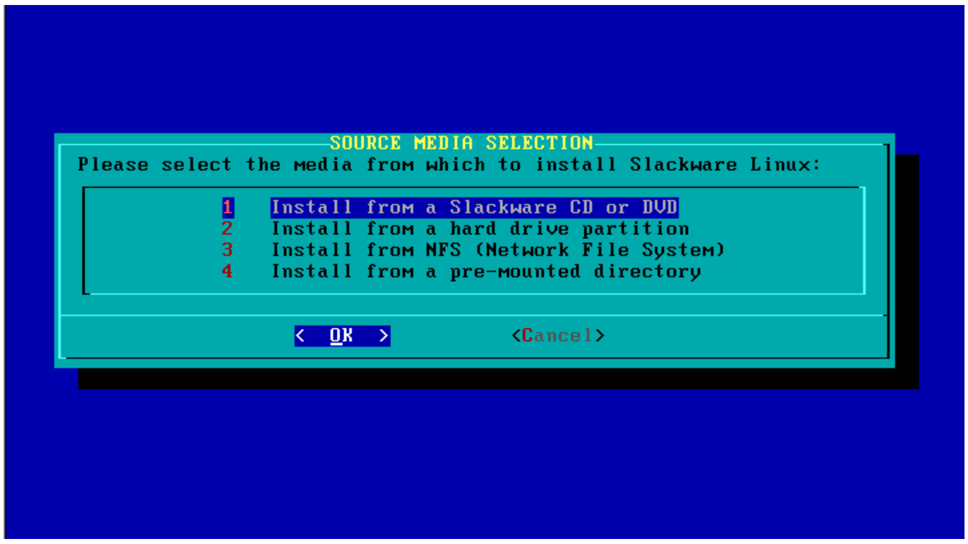


```
Partition /dev/sda1 will be formatted.

SELECT FILESYSTEM FOR /dev/sda1
Please select the type of filesystem to use for the specified
device.  Here are descriptions of the available filesystems:
Ext2 is the traditional Linux file system and is fast and
stable.  Ext3 is the journaling version of the Ext2 filesystem.
ReiserFS is a journaling filesystem that stores all files and
filenames in a balanced tree structure.

     ext2        Standard Linux ext2fs filesystem
     ext3        Journaling version of the ext2fs filesystem
     reiserfs    Hans Reiser's Journaling Filesystem

                  <  OK  >          <Cancel>
```

The first initialized partition is automatically mounted as the root (/) partition. For other partitions the mount point can be selected after the initialization. You could, for example make separate partitions for `/`, `/var`, `/tmp`, `/home` and `/usr`. This provides extra protection against crashes. For example the, `/` partition is barely changed after the installation if you create these partitions. So, on the occasion of a crash, the chance that the / partition was in the middle of a write operation is much smaller.

The next step is to select the source medium (Figure 5-7). This dialog offers several choices, like installing Slackware Linux from a CD-ROM or installing Slackware Linux via NFS. Most of the times Slackware Linux is installed from CD-ROM, so this is what we are going to look at. After selecting “CD-ROM” you will be asked whether you want to let setup look for the CD-ROM itself (“Auto”) or you want to select the CD-ROM device yourself (“Manual”). If you select “Manual” the setup tool will show a list of devices. Select the device holding the Slackware Linux CD-ROM.

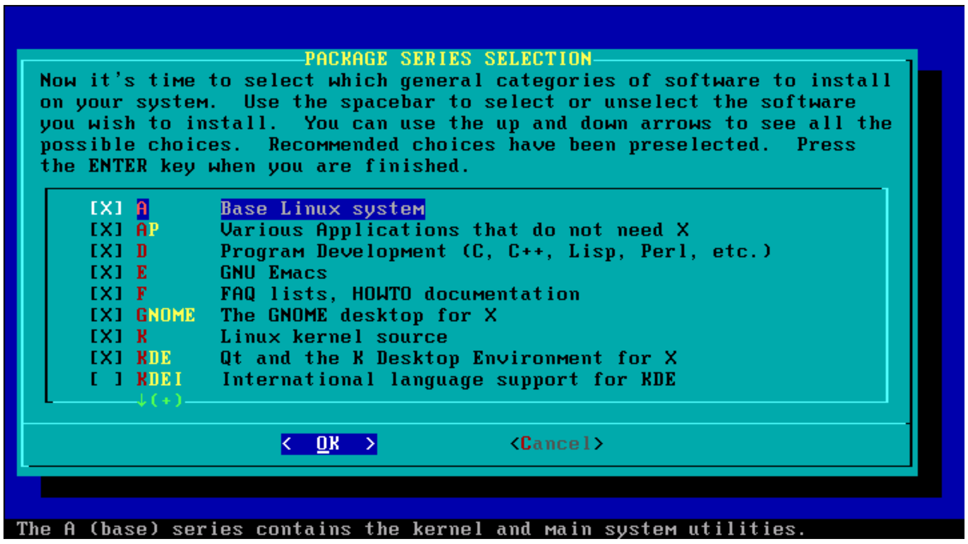
Figure 5-7. Selecting the source medium



```
SOURCE MEDIA SELECTION
Please select the media from which to install Slackware Linux:

            1   Install from a Slackware CD or DVD
            2   Install from a hard drive partition
            3   Install from NFS (Network File System)
            4   Install from a pre-mounted directory

              <  OK  >            <Cancel>
```

After choosing an installation source the setup tool will ask you which disk sets (series) you want to install packages from (Figure 5-8). A short description of each disk set is listed.

Figure 5-8. Selecting the disk sets

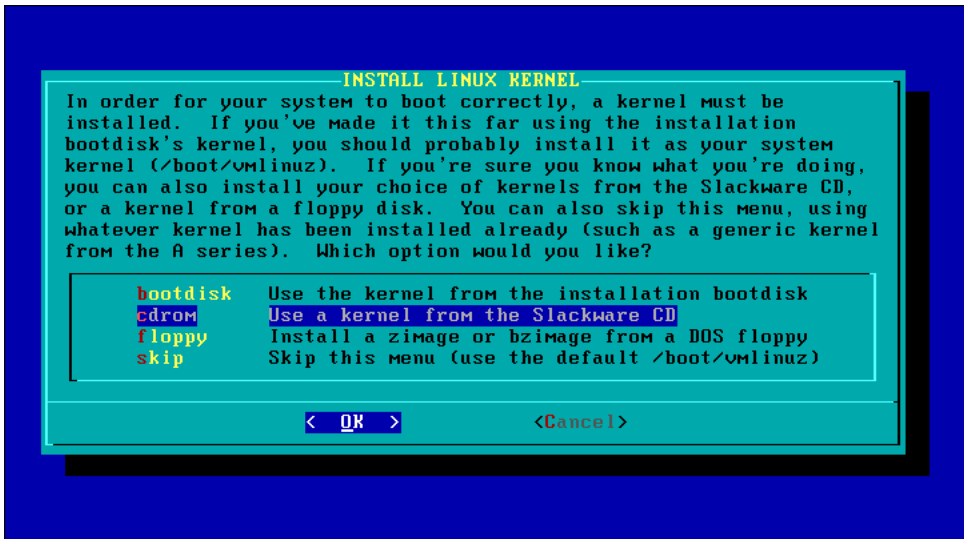


```
PACKAGE SERIES SELECTION
Now it's time to select which general categories of software to install
on your system.  Use the spacebar to select or unselect the software
you wish to install.  You can use the up and down arrows to see all the
possible choices.  Recommended choices have been preselected.  Press
the ENTER key when you are finished.

    [X] A        Base Linux system
    [X] AP       Various Applications that do not need X
    [X] D        Program Development (C, C++, Lisp, Perl, etc.)
    [X] E        GNU Emacs
    [X] F        FAQ lists, HOWTO documentation
    [X] GNOME    The GNOME desktop for X
    [X] K        Linux kernel source
    [X] KDE      Qt and the K Desktop Environment for X
    [ ] KDEI     International language support for KDE
          ↓(+)

              <  OK  >            <Cancel>

The A (base) series contains the kernel and main system utilities.
```

Now it is almost time to start the real installation. The next screen asks how you would like to install. The most obvious choices are “full”, “menu” or “expert”. Selecting “full” will install all packages in the selected disk sets. This is the easiest way of installing Slackware Linux. The disadvantage of this choice is that it can take quite much disk space. The “menu” option will ask you for each disk set which packages you want to install. The “expert” option is almost equal to the “menu” option, but allows you to deselect some very important packages from the “a” disk set.

After the completion of the installation the setup tool will allow you to configure some parts of the system. The first dialog will ask you where you would like to install the kernel from (see Figure 5-9). Normally it is a good idea to install the kernel from the Slackware Linux CD-ROM, this will select the kernel you installed Slackware Linux with. You can confirm this, or select another kernel.

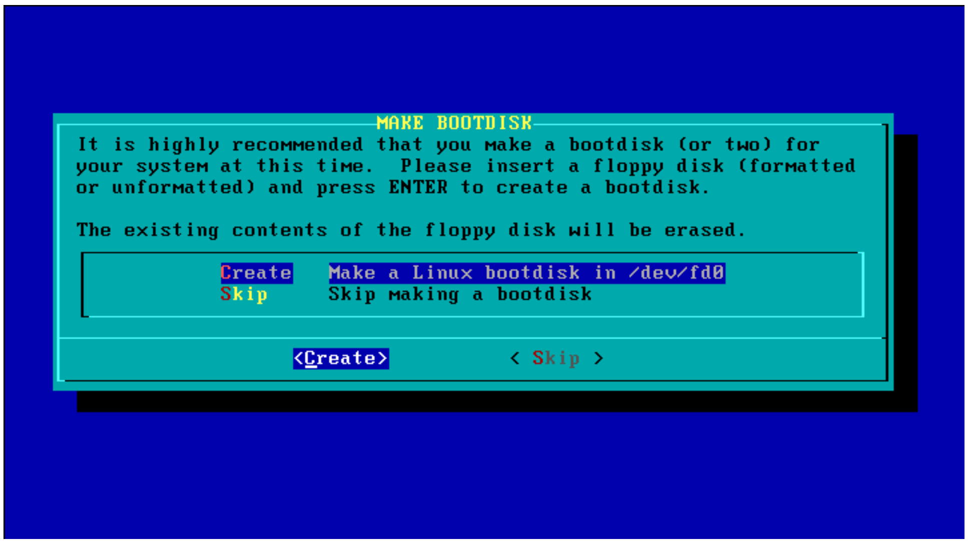
Figure 5-9. Installing the kernel



```
INSTALL LINUX KERNEL
In order for your system to boot correctly, a kernel must be
installed.  If you've made it this far using the installation
bootdisk's kernel, you should probably install it as your system
kernel (/boot/vmlinuz).  If you're sure you know what you're doing,
you can also install your choice of kernels from the Slackware CD,
or a kernel from a floppy disk.  You can also skip this menu, using
whatever kernel has been installed already (such as a generic kernel
from the A series).  Which option would you like?

      bootdisk    Use the kernel from the installation bootdisk
      cdrom       Use a kernel from the Slackware CD
      floppy      Install a zimage or bzimage from a DOS floppy
      skip        Skip this menu (use the default /boot/vmlinuz)

                  <  OK  >            <Cancel>
```

At this point you can choose to make a bootdisk (Figure 5-10). It is a good idea to make a bootdisk, you can use it to boot Slackware Linux if the LILO configuration is botched.

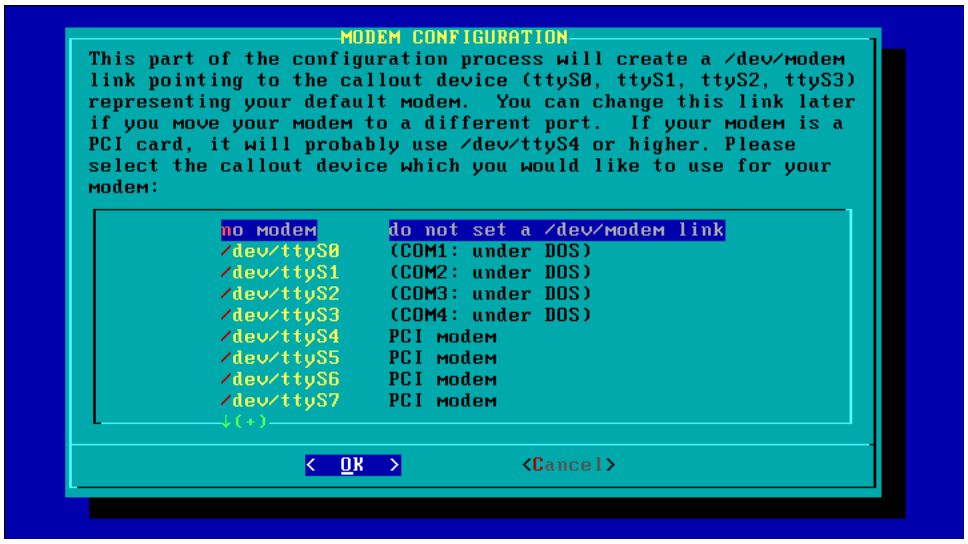
Figure 5-10. Creating a bootdisk



```
MAKE BOOTDISK
It is highly recommended that you make a bootdisk (or two) for
your system at this time.  Please insert a floppy disk (formatted
or unformatted) and press ENTER to create a bootdisk.

The existing contents of the floppy disk will be erased.

             Create    Make a Linux bootdisk in /dev/fd0
             Skip      Skip making a bootdisk

               <Create>          < Skip >
```

The following dialog can be used to make a link, `/dev/modem`, that points to your modem device. (Figure 5-11). If you do not have a modem you can select *no modem*.

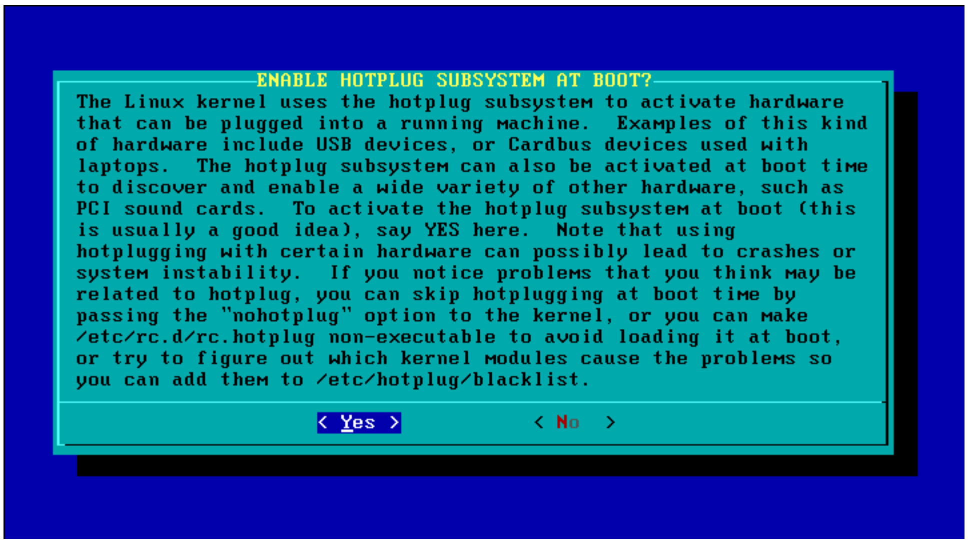
Figure 5-11. Selecting the default modem



```
MODEM CONFIGURATION
This part of the configuration process will create a /dev/modem
link pointing to the callout device (ttyS0, ttyS1, ttyS2, ttyS3)
representing your default modem.  You can change this link later
if you move your modem to a different port.  If your modem is a
PCI card, it will probably use /dev/ttyS4 or higher. Please
select the callout device which you would like to use for your
modem:

            no modem       do not set a /dev/modem link
            /dev/ttyS0     (COM1: under DOS)
            /dev/ttyS1     (COM2: under DOS)
            /dev/ttyS2     (COM3: under DOS)
            /dev/ttyS3     (COM4: under DOS)
            /dev/ttyS4     PCI modem
            /dev/ttyS5     PCI modem
            /dev/ttyS6     PCI modem
            /dev/ttyS7     PCI modem
            ↓(+)

                  <  OK  >             <Cancel>
```

The next step is to select whether you would like to use hotplug (Figure 5-12). Hotplug is used for automatically configuring pluggable USB, PCMCIA and PCI devices. Generally speaking it is a good idea to enable hotplugging, but some systems may have problems with the probing of the hotplug scripts.

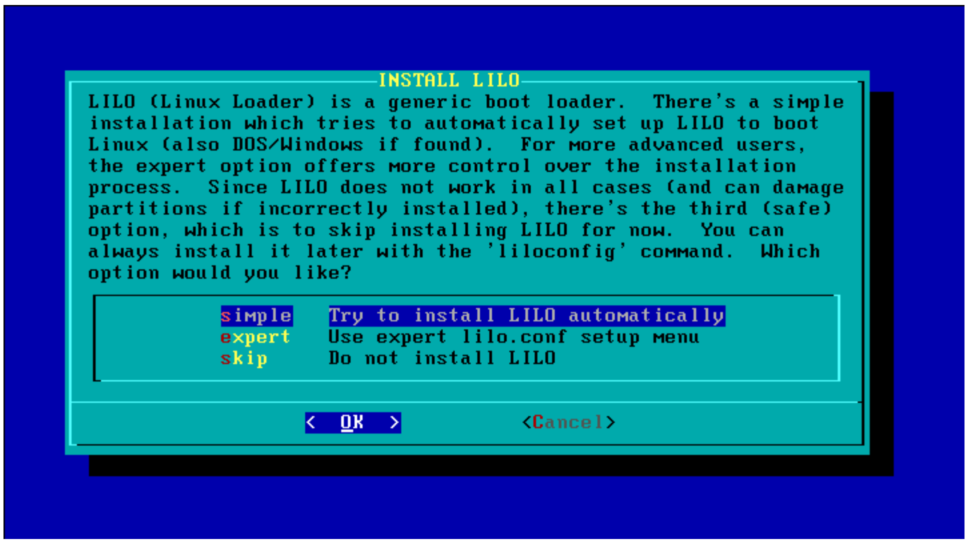
Figure 5-12. Enabling hotplugging



```
ENABLE HOTPLUG SUBSYSTEM AT BOOT?
The Linux kernel uses the hotplug subsystem to activate hardware
that can be plugged into a running machine.  Examples of this kind
of hardware include USB devices, or Cardbus devices used with
laptops.  The hotplug subsystem can also be activated at boot time
to discover and enable a wide variety of other hardware, such as
PCI sound cards.  To activate the hotplug subsystem at boot (this
is usually a good idea), say YES here.  Note that using
hotplugging with certain hardware can possibly lead to crashes or
system instability.  If you notice problems that you think may be
related to hotplug, you can skip hotplugging at boot time by
passing the "nohotplug" option to the kernel, or you can make
/etc/rc.d/rc.hotplug non-executable to avoid loading it at boot,
or try to figure out which kernel modules cause the problems so
you can add them to /etc/hotplug/blacklist.

                  < Yes >            <  No  >
```

The following steps are important, the next dialogs will assist you with installing LILO, the Linux bootloader. Unless you have experience in configuring LILO it is a good idea to choose to use the *simple* option for configuration of LILO, which tries to configure LILO automatically (Figure 5-13).

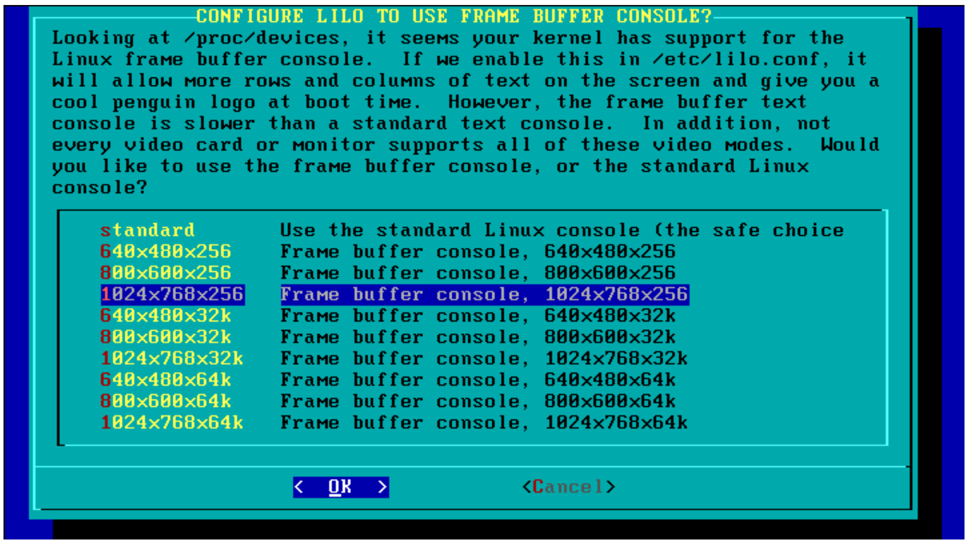
Figure 5-13. Selecting the kind of LILO installation



```
INSTALL LILO
LILO (Linux Loader) is a generic boot loader.  There's a simple
installation which tries to automatically set up LILO to boot
Linux (also DOS/Windows if found).  For more advanced users,
the expert option offers more control over the installation
process.  Since LILO does not work in all cases (and can damage
partitions if incorrectly installed), there's the third (safe)
option, which is to skip installing LILO for now.  You can
always install it later with the 'liloconfig' command.  Which
option would you like?

          simple    Try to install LILO automatically
          expert    Use expert lilo.conf setup menu
          skip      Do not install LILO

            < OK >            <Cancel>
```

After selecting the *simple* option the LILO configuration utility will asks you whether you would like to use a framebuffer or not (Figure 5-14). Using a framebuffer will allow you to use the console in several resolutions, with other dimensions than the usual 80x25 characters. Some people who use the console extensively prefer to use a framebuffer, which allows them to keep more text on the screen. If you do not want a framebuffer console, or if you are not sure, you can choose *standard* here.

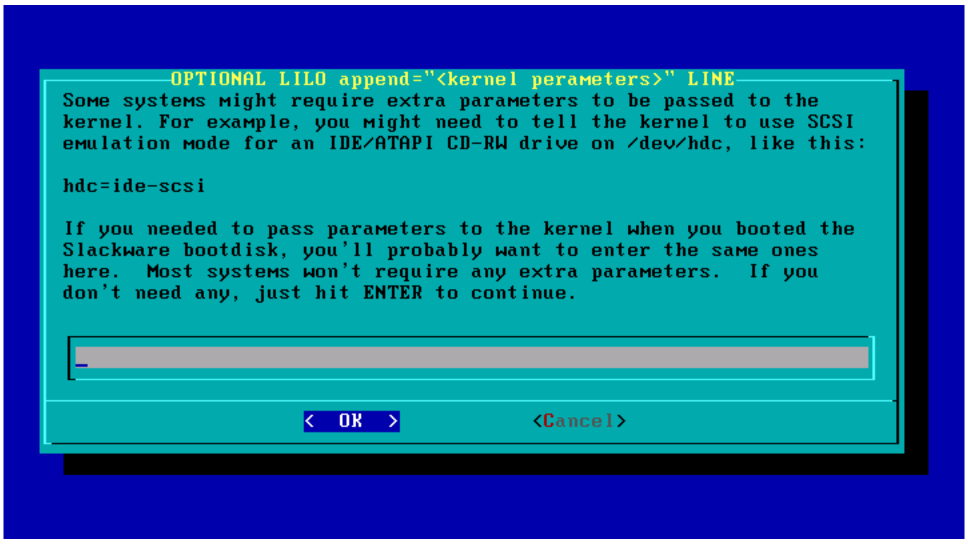
Figure 5-14. Choosing the framebuffer resolution



```
CONFIGURE LILO TO USE FRAME BUFFER CONSOLE?
Looking at /proc/devices, it seems your kernel has support for the
Linux frame buffer console.  If we enable this in /etc/lilo.conf, it
will allow more rows and columns of text on the screen and give you a
cool penguin logo at boot time.  However, the frame buffer text
console is slower than a standard text console.  In addition, not
every video card or monitor supports all of these video modes.  Would
you like to use the frame buffer console, or the standard Linux
console?

    standard        Use the standard Linux console (the safe choice
    640x480x256     Frame buffer console, 640x480x256
    800x600x256     Frame buffer console, 800x600x256
    1024x768x256    Frame buffer console, 1024x768x256
    640x480x32k     Frame buffer console, 640x480x32k
    800x600x32k     Frame buffer console, 800x600x32k
    1024x768x32k    Frame buffer console, 1024x768x32k
    640x480x64k     Frame buffer console, 640x480x64k
    800x600x64k     Frame buffer console, 800x600x64k
    1024x768x64k    Frame buffer console, 1024x768x64k

              < OK >            <Cancel>
```

After setting the framebuffer you can pass extra parameters to the kernel (Figure 5-15). This is normally not necessary, if you do not want to pass extra parameters you can just press the <Enter> key.

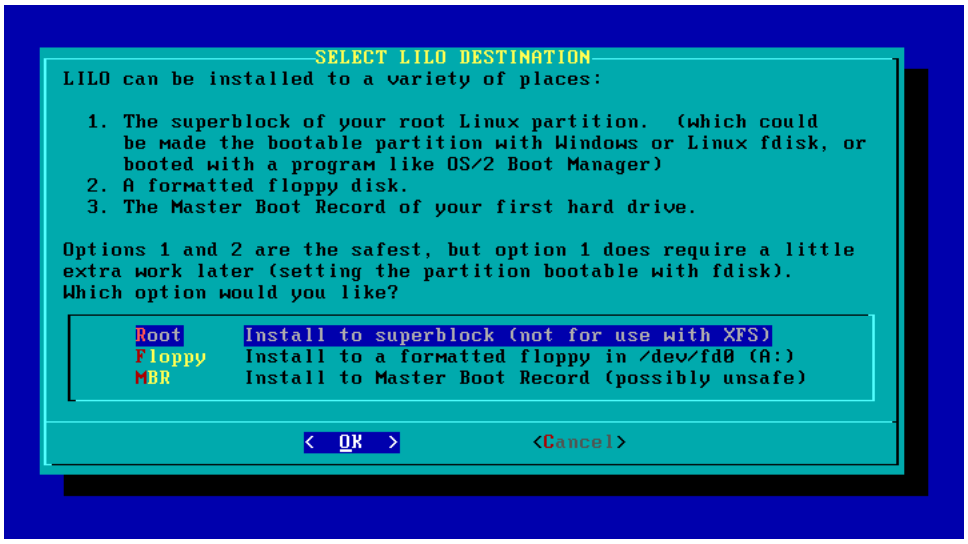
Figure 5-15. Adding kernel parameters



```
OPTIONAL LILO append="<kernel perameters>" LINE
Some systems might require extra parameters to be passed to the
kernel. For example, you might need to tell the kernel to use SCSI
emulation mode for an IDE/ATAPI CD-RW drive on /dev/hdc, like this:

hdc=ide-scsi

If you needed to pass parameters to the kernel when you booted the
Slackware bootdisk, you'll probably want to enter the same ones
here.  Most systems won't require any extra parameters.  If you
don't need any, just hit ENTER to continue.

< OK >        <Cancel>
```

The last step of the LILO configuration is selecting where LILO should be installed (Figure 5-16). *MBR* is the master boot record, the main boot record of PCs. Use this option if you want use Slackware Linux as only OS, or if you want to use LILO the boot other operating systems. The *Root* option will install LILO in the boot record of the Slackware Linux / partition. Use this option if you use another bootloader.

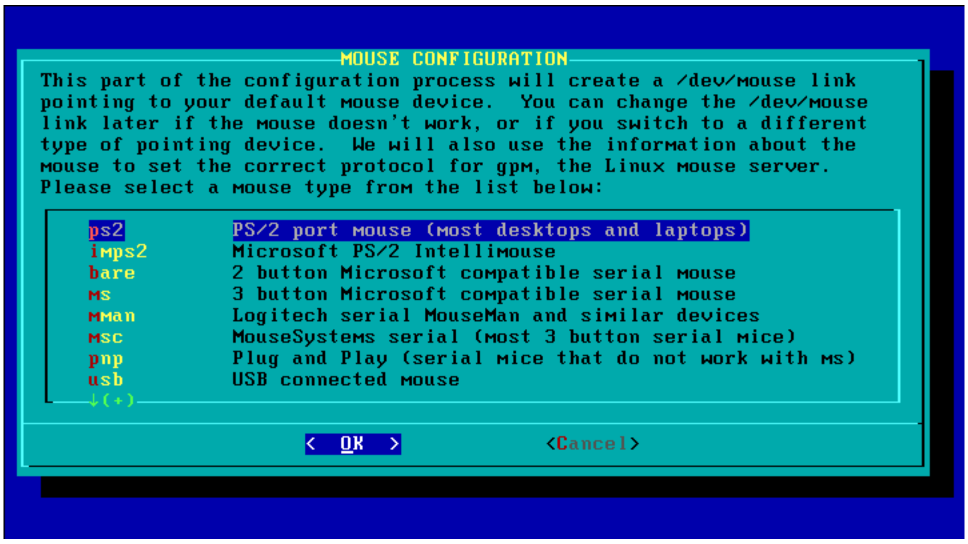
Figure 5-16. Choosing where LILO should be installed



```
SELECT LILO DESTINATION
LILO can be installed to a variety of places:

  1. The superblock of your root Linux partition.  (which could
     be made the bootable partition with Windows or Linux fdisk, or
     booted with a program like OS/2 Boot Manager)
  2. A formatted floppy disk.
  3. The Master Boot Record of your first hard drive.

Options 1 and 2 are the safest, but option 1 does require a little
extra work later (setting the partition bootable with fdisk).
Which option would you like?

      Root      Install to superblock (not for use with XFS)
      Floppy    Install to a formatted floppy in /dev/fd0 (A:)
      MBR       Install to Master Boot Record (possibly unsafe)

< OK >        <Cancel>
```

You will now be asked to configure your mouse. Select the mouse type from the dialog that appears (Figure 5-17).

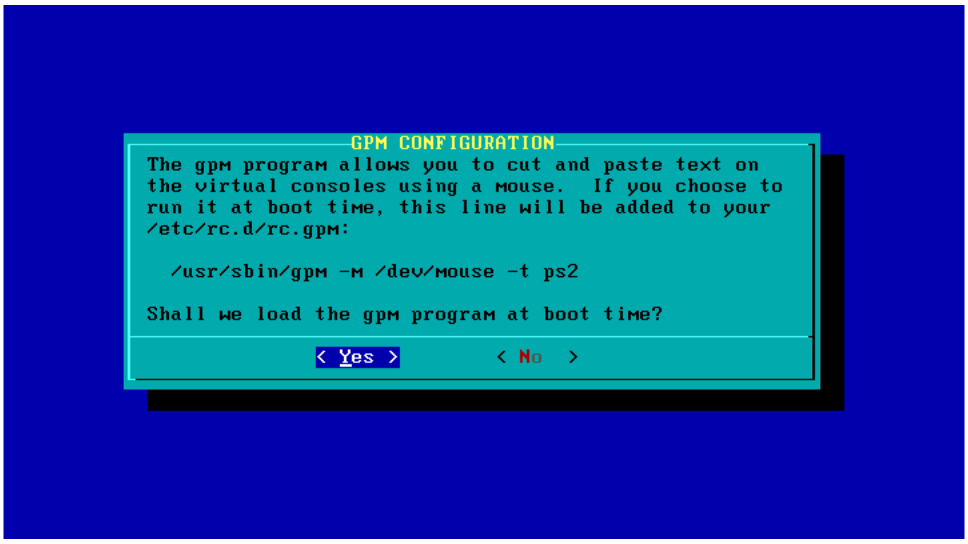
Figure 5-17. Configuring a mouse



```
MOUSE CONFIGURATION
This part of the configuration process will create a /dev/mouse link
pointing to your default mouse device.  You can change the /dev/mouse
link later if the mouse doesn't work, or if you switch to a different
type of pointing device.  We will also use the information about the
mouse to set the correct protocol for gpm, the Linux mouse server.
Please select a mouse type from the list below:

  ps2         PS/2 port mouse (most desktops and laptops)
  imps2       Microsoft PS/2 Intellimouse
  bare        2 button Microsoft compatible serial mouse
  ms          3 button Microsoft compatible serial mouse
  mman        Logitech serial MouseMan and similar devices
  msc         MouseSystems serial (most 3 button serial mice)
  pnp         Plug and Play (serial mice that do not work with ms)
  usb         USB connected mouse
  ↓(+)

          <  OK  >              <Cancel>
```

You will then be asked whether the **gpm** program should be loaded at boot time or not (Figure 5-18). **gpm** is a daemon that allows you to cut and paste text on the console.

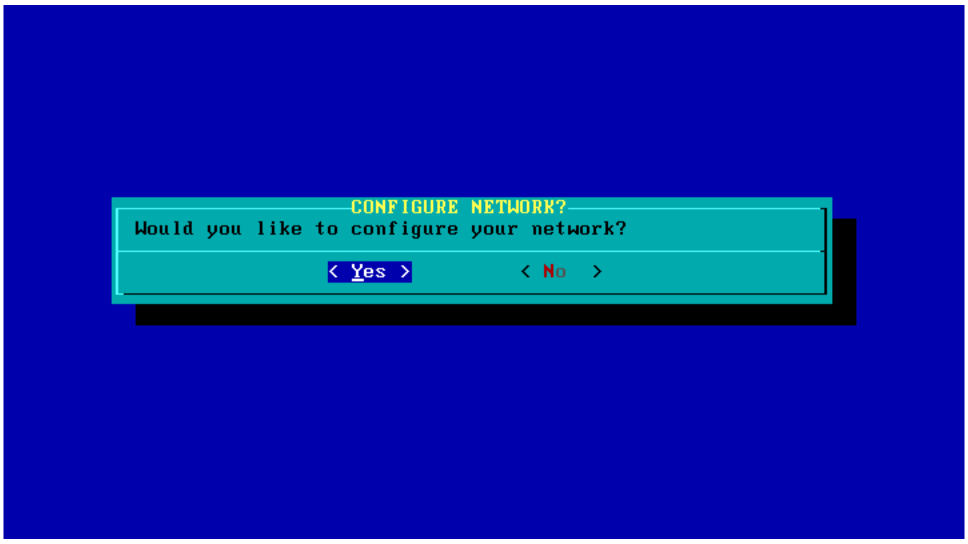
Figure 5-18. Choosing whether GPM should be started or not



```
GPM CONFIGURATION
The gpm program allows you to cut and paste text on
the virtual consoles using a mouse.  If you choose to
run it at boot time, this line will be added to your
/etc/rc.d/rc.gpm:

  /usr/sbin/gpm -m /dev/mouse -t ps2

Shall we load the gpm program at boot time?

          < Yes >         <  No  >
```

The next few steps will configure network connectivity. This is required on almost every networked system. The Slackware Linux setup will ask you if you want to set up network connectivity (Figure 5-19). If you answer "No" you can skip the next few network-related steps.

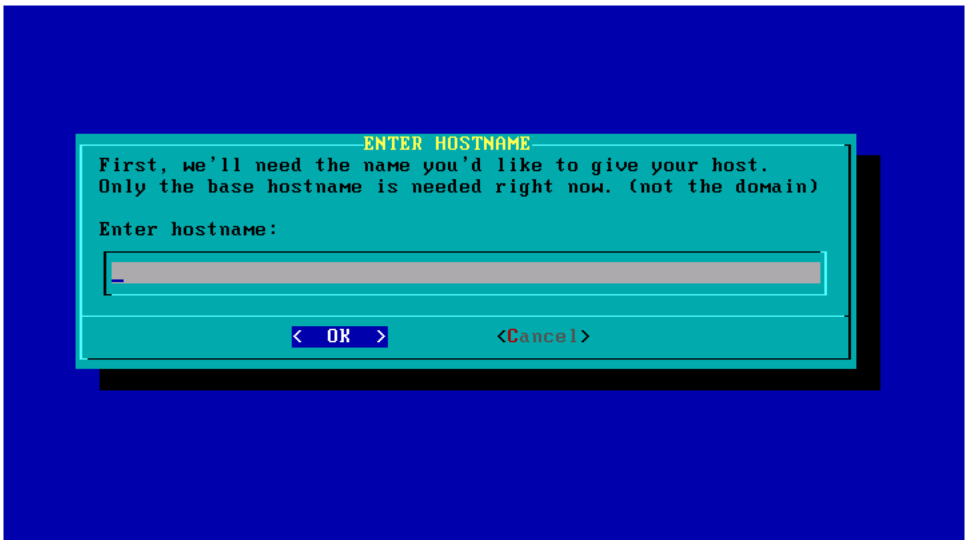
Figure 5-19. Choosing whether you would like to configure network connectivity



```
CONFIGURE NETWORK?
Would you like to configure your network?

              < Yes >          < No  >
```

You will now be asked to set the hostname (Figure 5-20). Please note that this is not the fully qualified domain name (FQDN), just the part that represents the host (normally the characters before the first dot in a FQDN).

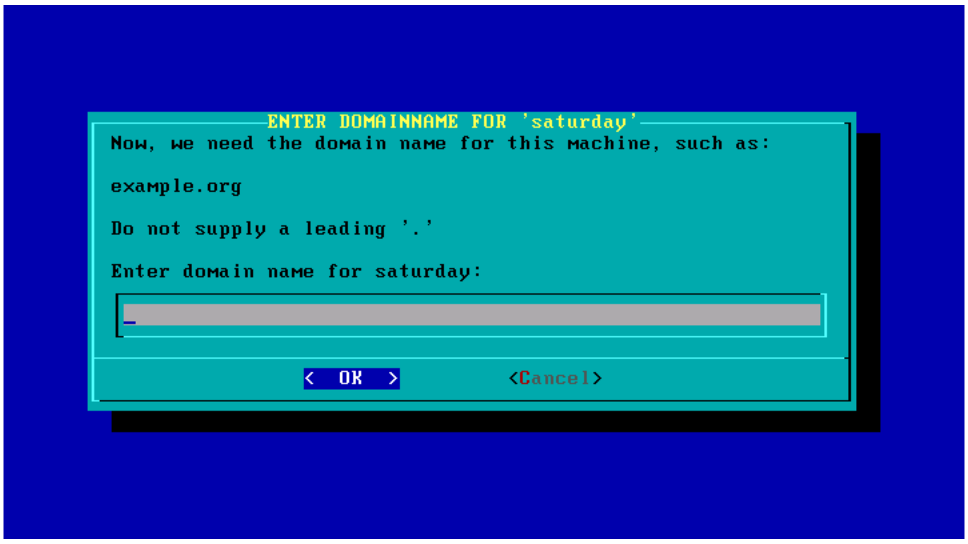
Figure 5-20. Setting the host name



```
ENTER HOSTNAME
First, we'll need the name you'd like to give your host.
Only the base hostname is needed right now. (not the domain)

Enter hostname:

_

              <  OK  >          <Cancel>
```

After setting the host name you can set the domain name part of the fully qualified domain name (FQDN) (Figure 5-21).

Figure 5-21. Setting the domain name

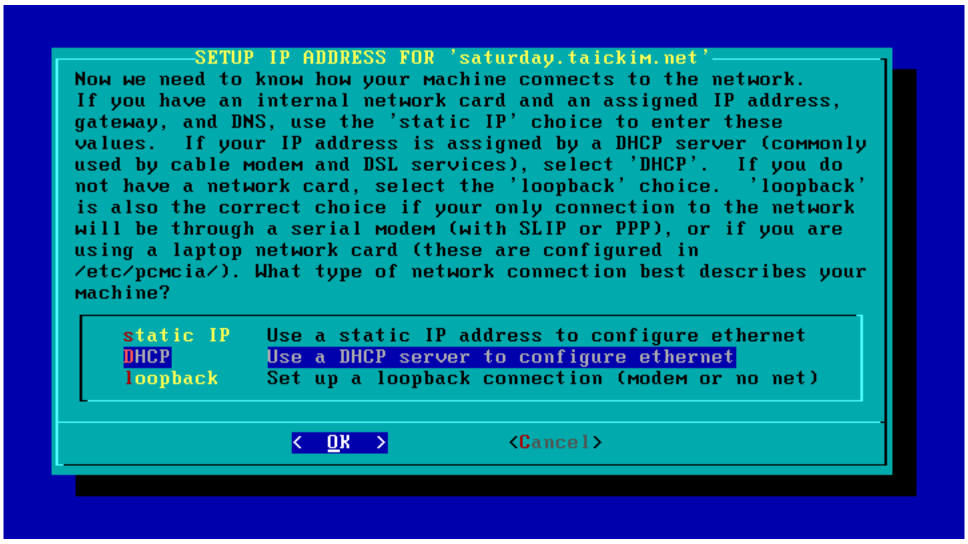


```
ENTER DOMAINNAME FOR 'saturday'
Now, we need the domain name for this machine, such as:

example.org

Do not supply a leading '.'

Enter domain name for saturday:
```

The rest of the network configuration depends on how the IP address is set on the network the machine is connected to. Some networks have a DHCP server that automatically assigns an IP address to hosts in the network. If this is the case for the network the machine select *DHCP* during this step of the installation (Figure 5-22). When DHCP is selected you will only be asked whether a host name should be sent to the server, normally you can leave this blank. If you use DHCP you can skip the rest of the network configuration described below.

If the network does not have a DHCP server you can choose the *static IP* option, which will allow you to set the IP address and related settings manually.

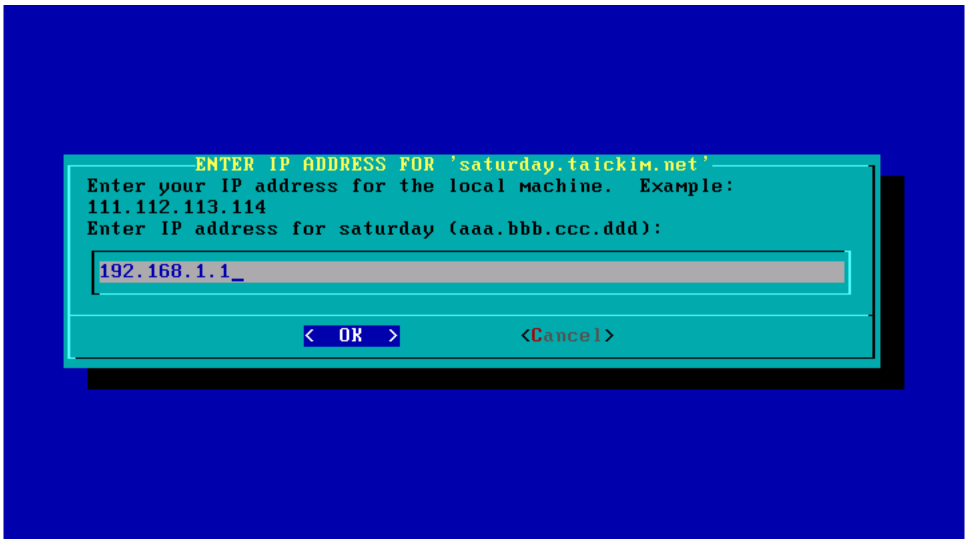
Figure 5-22. Manual or automatic IP address configuration



```
SETUP IP ADDRESS FOR 'saturday.taickim.net'
Now we need to know how your machine connects to the network.
 If you have an internal network card and an assigned IP address,
gateway, and DNS, use the 'static IP' choice to enter these
values.  If your IP address is assigned by a DHCP server (commonly
used by cable modem and DSL services), select 'DHCP'.  If you do
not have a network card, select the 'loopback' choice.  'loopback'
is also the correct choice if your only connection to the network
will be through a serial modem (with SLIP or PPP), or if you are
using a laptop network card (these are configured in
/etc/pcmcia/). What type of network connection best describes your
machine?

    static IP    Use a static IP address to configure ethernet
    DHCP         Use a DHCP server to configure ethernet
    loopback     Set up a loopback connection (modem or no net)
```

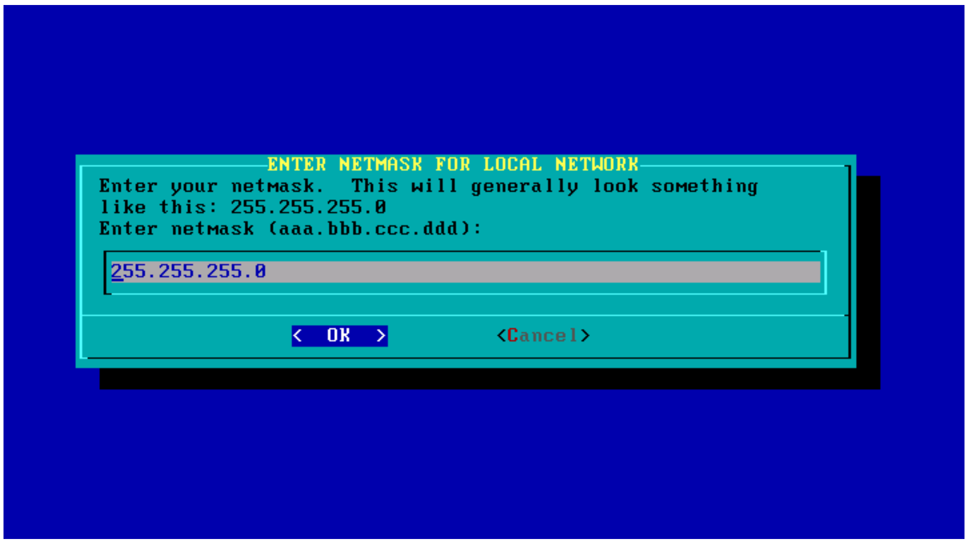
The first step of the manual configuration is to set the IP address of the first interface (eth0) of the machine (Figure 5-23).

Figure 5-23. Setting the IP addres


```
ENTER IP ADDRESS FOR 'saturday.taickim.net'
Enter your IP address for the local machine.  Example:
111.112.113.114
Enter IP address for saturday (aaa.bbb.ccc.ddd):

192.168.1.1_
```

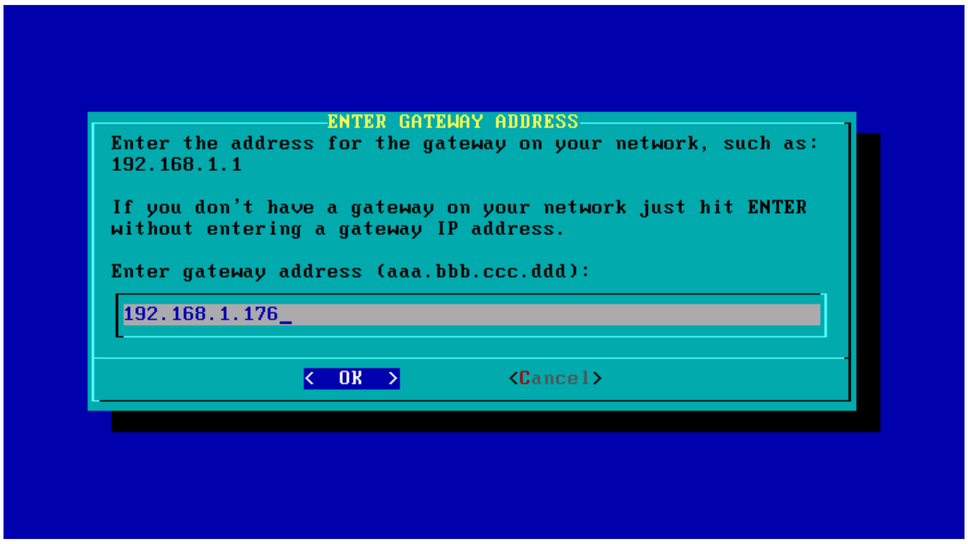
After setting the IP address you will be asked to enter the netmask. The netmask is usually dependent on the IP address class (Figure 5-24).

Figure 5-24. Setting the netmask


```
ENTER NETMASK FOR LOCAL NETWORK
Enter your netmask.  This will generally look something
like this: 255.255.255.0
Enter netmask (aaa.bbb.ccc.ddd):

255.255.255.0
```

You will then be asked to set the address of the gateway (Figure 5-25). The gateway is the machine on the network that provides access to other networks by routing IP packets. If your network has no network you can just hit the <ENTER> key.

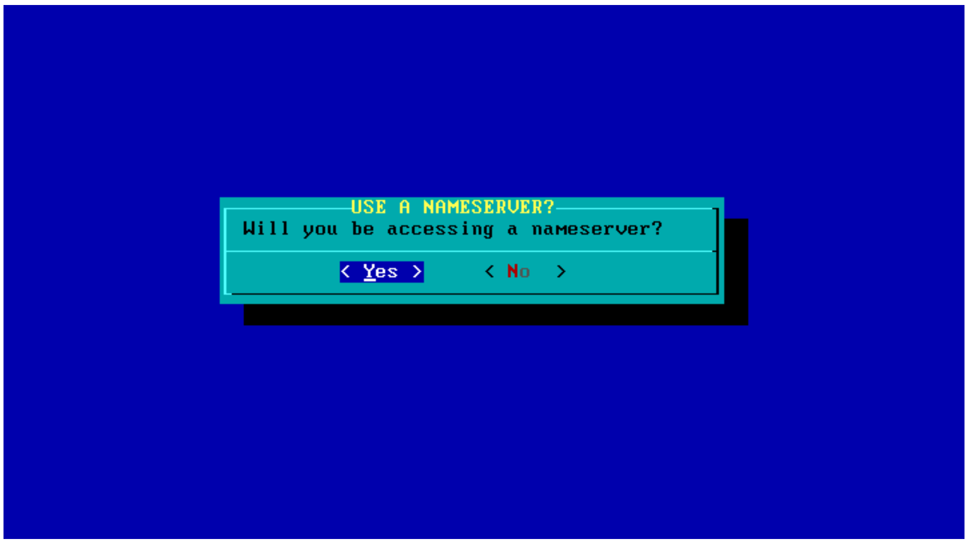
Figure 5-25. Setting the gateway



```
ENTER GATEWAY ADDRESS
Enter the address for the gateway on your network, such as:
192.168.1.1

If you don't have a gateway on your network just hit ENTER
without entering a gateway IP address.

Enter gateway address (aaa.bbb.ccc.ddd):

192.168.1.176_
```

The next dialog will ask you whether you want to use a nameserver or not (Figure 5-26). A nameserver is a server that can provide the IP address for a hostname. For example, if you surf to *www.slackbasics.org*, the nameserver will “convert” the *www.slackbasics.org* name to the corresponding IP address.

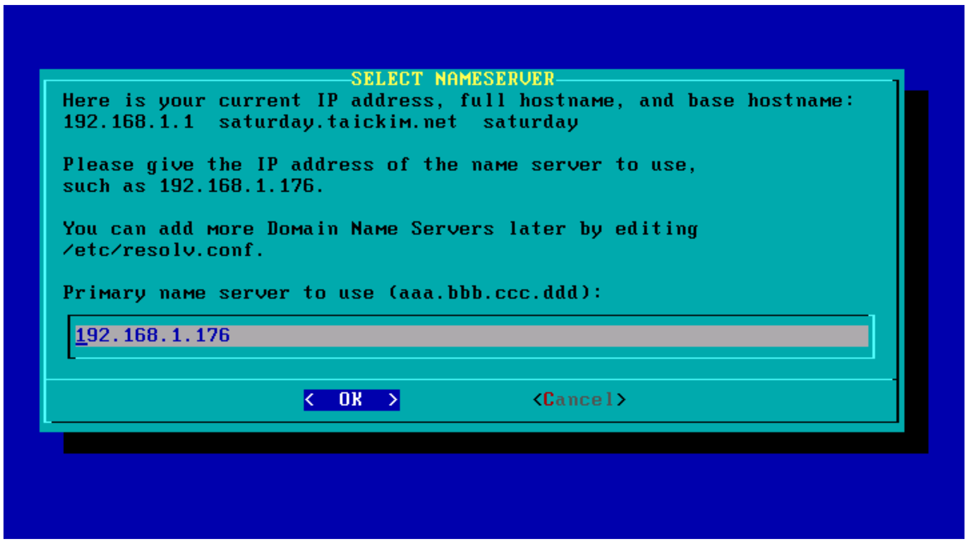
Figure 5-26. Choosing whether you want to use a nameserver or not



```
USE A NAMESERVER?
Will you be accessing a nameserver?
```

If you chose to use a nameserver, you will be given the opportunity to set the IP address of the nameserver (Figure 5-27).

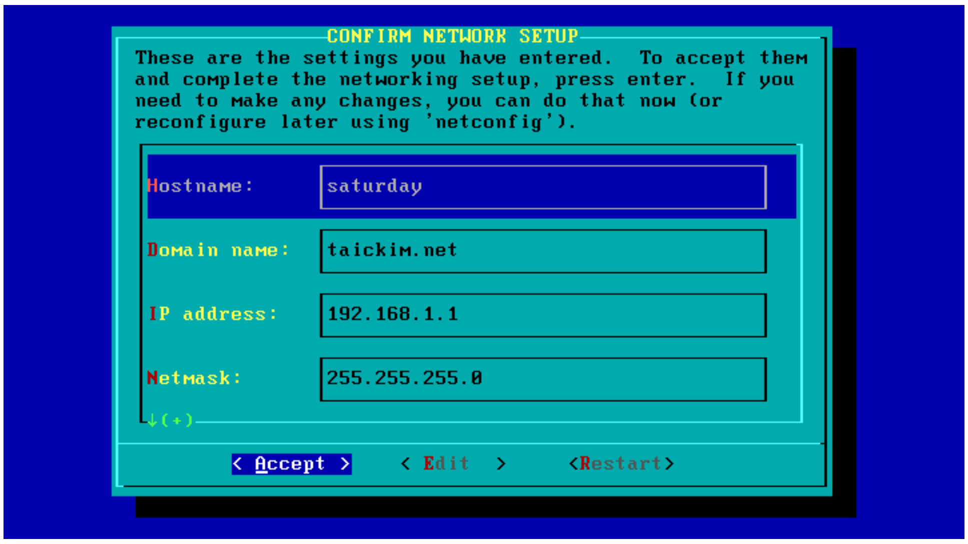
Figure 5-27. Setting the nameserver(s)



```
SELECT NAMESERVER
Here is your current IP address, full hostname, and base hostname:
192.168.1.1  saturday.taickim.net  saturday

Please give the IP address of the name server to use,
such as 192.168.1.176.

You can add more Domain Name Servers later by editing
/etc/resolv.conf.

Primary name server to use (aaa.bbb.ccc.ddd):

192.168.1.176

          <  OK  >          <Cancel>
```

The final network settings screen provides an overview of the settings, giving you the opportunity to correct settings that have errors (Figure 5-28).

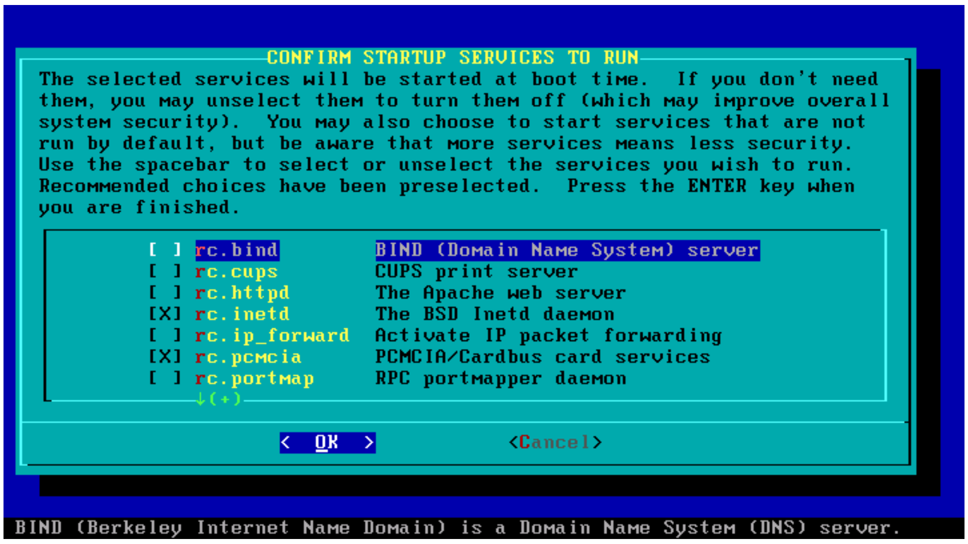
Figure 5-28. Confirming the network settings



```
CONFIRM NETWORK SETUP
These are the settings you have entered.  To accept them
and complete the networking setup, press enter.  If you
need to make any changes, you can do that now (or
reconfigure later using 'netconfig').

Hostname:       saturday

Domain name:    taickim.net

IP address:     192.168.1.1

Netmask:        255.255.255.0
↓(+)

      < Accept >    < Edit  >    <Restart>
```

After the network configuration you can set which services should be started (Figure 5-29). You can mark/unmark services with the <SPACE> key.

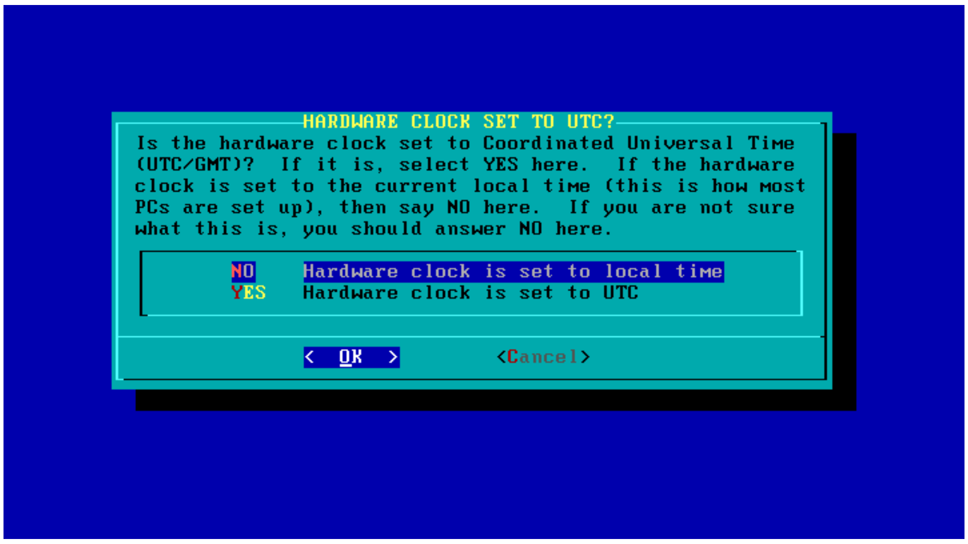
Figure 5-29. Enabling/disabling startup services



```
CONFIRM STARTUP SERVICES TO RUN
The selected services will be started at boot time.  If you don't need
them, you may unselect them to turn them off (which may improve overall
system security).  You may also choose to start services that are not
run by default, but be aware that more services means less security.
Use the spacebar to select or unselect the services you wish to run.
Recommended choices have been preselected.  Press the ENTER key when
you are finished.

          [ ] rc.bind          BIND (Domain Name System) server
          [ ] rc.cups          CUPS print server
          [ ] rc.httpd         The Apache web server
          [X] rc.inetd         The BSD Inetd daemon
          [ ] rc.ip_forward    Activate IP packet forwarding
          [X] rc.pcmcia        PCMCIA/Cardbus card services
          [ ] rc.portmap       RPC portmapper daemon
              ↓(+)

                 <  OK  >            <Cancel>

BIND (Berkeley Internet Name Domain) is a Domain Name System (DNS) server.
```

Traditionally the system clock is set to the UTC timezone on UNIX(-like) systems. If this is the case, select *Yes* during the next step (Figure 5-30). If you also use a non-UNIX(-like) OS on the same system, like Windows, it is usually a good idea to choose *No*, because some PC operating systems do not work with a separate system clock and software clock.

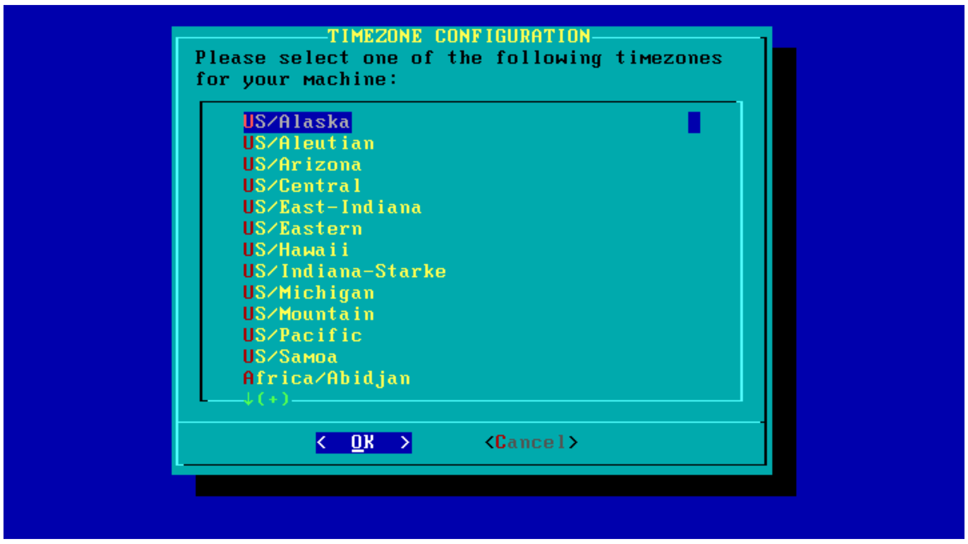
Figure 5-30. Choosing whether the clock is set to UTC



```
HARDWARE CLOCK SET TO UTC?
Is the hardware clock set to Coordinated Universal Time
(UTC/GMT)?  If it is, select YES here.  If the hardware
clock is set to the current local time (this is how most
PCs are set up), then say NO here.  If you are not sure
what this is, you should answer NO here.

        NO     Hardware clock is set to local time
        YES    Hardware clock is set to UTC

              <  OK  >         <Cancel>
```

You will then be given the opportunity to select the time zone (Figure 5-31). This is especially important on systems that have their system clock set to UTC, without selecting the correct timezone the software clock will not match the local time.

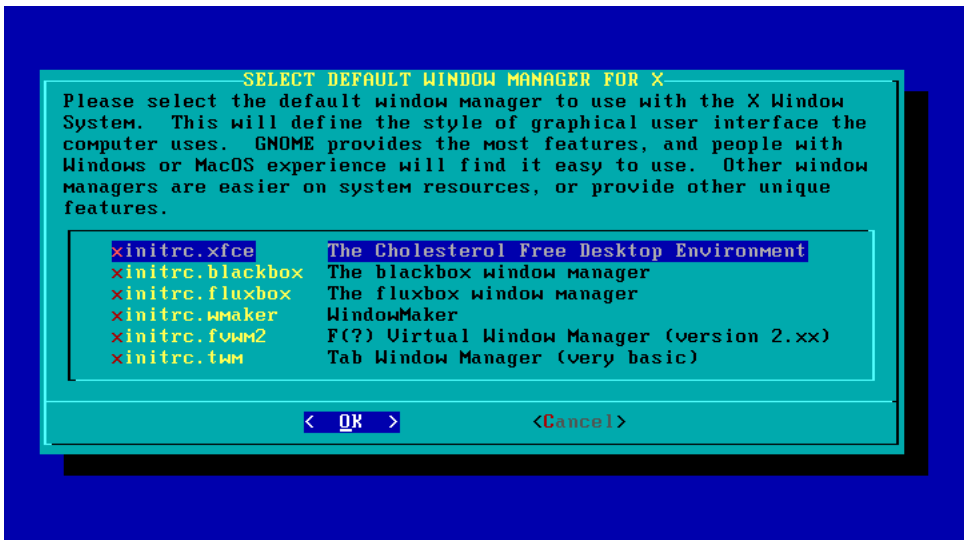
Figure 5-31. Setting the timezone



```
TIMEZONE CONFIGURATION
Please select one of the following timezones
for your machine:

    US/Alaska
    US/Aleutian
    US/Arizona
    US/Central
    US/East-Indiana
    US/Eastern
    US/Hawaii
    US/Indiana-Starke
    US/Michigan
    US/Mountain
    US/Pacific
    US/Samoa
    Africa/Abidjan
    ↓(+)

          <  OK  >        <Cancel>
```

If you installed the X Window System you can now set the default window manager (Figure 5-32). The most basic functionality of a window manager is to provide basic window managing functionality like title bars. But some options, like KDE provide a complete desktop environment.

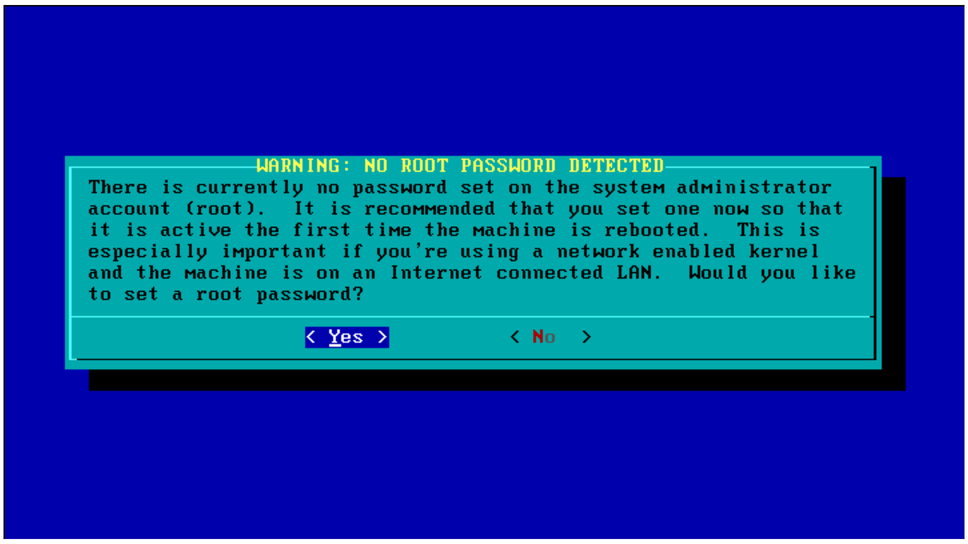
Figure 5-32. Choosing the default window manager



```
SELECT DEFAULT WINDOW MANAGER FOR X
Please select the default window manager to use with the X Window
System.  This will define the style of graphical user interface the
computer uses.  GNOME provides the most features, and people with
Windows or MacOS experience will find it easy to use.  Other window
managers are easier on system resources, or provide other unique
features.

    xinitrc.xfce        The Cholesterol Free Desktop Environment
    xinitrc.blackbox    The blackbox window manager
    xinitrc.fluxbox     The fluxbox window manager
    xinitrc.wmaker      WindowMaker
    xinitrc.fvwm2       F(?) Virtual Window Manager (version 2.xx)
    xinitrc.twm         Tab Window Manager (very basic)

          <  OK  >          <Cancel>
```

The final step is to set the root password (Figure 5-33). The setup will ask you whether you would like to set it or not. There is no reason not to do this, and without a root password your system is dangerously insecure.

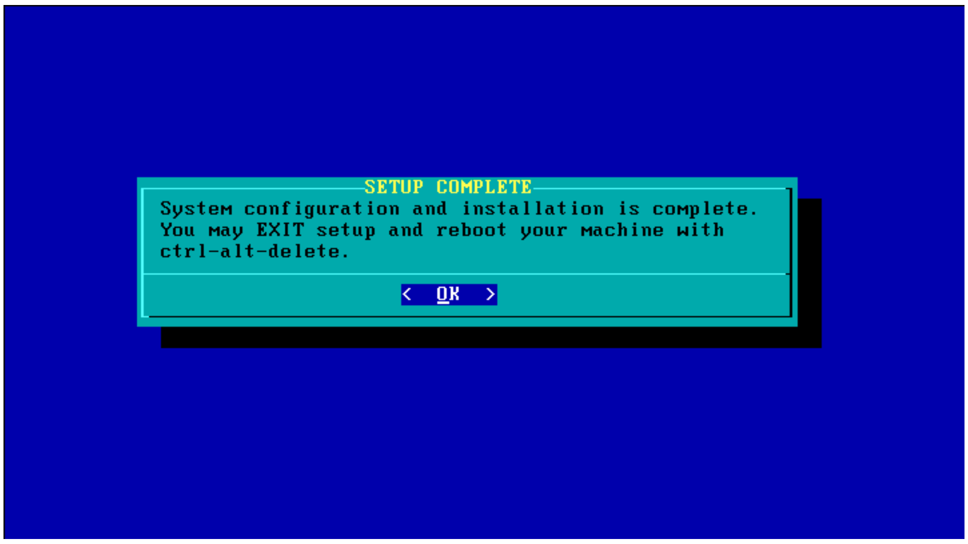
Figure 5-33. Setting the root password



```
WARNING: NO ROOT PASSWORD DETECTED
There is currently no password set on the system administrator
account (root).  It is recommended that you set one now so that
it is active the first time the machine is rebooted.  This is
especially important if you're using a network enabled kernel
and the machine is on an Internet connected LAN.  Would you like
to set a root password?

            < Yes >             < No  >
```

At this point you have completed the Slackware Linux installation. You can now reboot the system to start your fresh new Slackware Linux system. It was not that hard, was is? ;-)

Figure 5-34. Finished



```
SETUP COMPLETE
System configuration and installation is complete.
You may EXIT setup and reboot your machine with
ctrl-alt-delete.

                < OK >
```

Chapter 6. Custom installation

Sometimes you may want to do a custom installation of Slackware Linux, for example to learn better how GNU/Linux systems work, or to prepare an automatic installation script. This chapter outlines the steps that are required to install Slackware Linux manually. A sample installation script is also provided in the section called *Automated installation script*.

Partitioning a hard disk

If you have performed a normal installation, you should not have any problems partitioning a disk. You can use the **fdisk** and **cfdisk** commands to partition any disks. If you are scripting the installation of Slackware Linux it is useful to know that you can also pipe fdisk commands to the **fdisk** binary. For example:

```
# fdisk /dev/hda << EOF
n
p
1

+10000M
n
p
2

+512M
t
2
82
w
EOF
```

This would create a primary Linux partition of 10000MB, and a primary Linux swap partition of 512MB. You could also store the fdisk commands into different disk profiles, and use one of the profiles based on other information (e.g. the disk size). For example:

```
# cat /usr/share/fdisk-profiles/smalldisk | fdisk
```

Initializing and mounting filesystems

After making at least a swap and Linux partition, you can initialize and mount the filesystems. Especially on systems that are short on memory, it is a good idea to initialize, and use swap first. We will use the partition layout listed above in the following examples. To set up and use swap, execute:

```
# mkswap /dev/hda2
# swapon /dev/hda2
```

The meaning of these commands is quite obvious. **mkswap** initializes the swap space, and **swapon** puts it to use. You will only have to execute **mkswap** once, but **swapon** has to be executed during every system boot. This can be done automatically by adding an entry for the swap partition to `/etc/fstab`, we will do this at a later step.

For now, it is important to initialize the target partitions. This can be done with the **mkfs** command. You can specify which filesystem should be used by adding the `-t` parameter. **mkfs** will automatically invoke a **mkfs.filesystem** command, based on the filesystem you chose. Be aware that the filesystems that can be used is dependent on the installation kernel that you booted. If you booted the bare.i kernel, you can use the *ext3*, *ext3* and *reiserfs* filesystems. To initialize an *ext3* filesystem, and mount it, you should execute the following commands:

```
# mkfs -t ext3 /dev/hda1
# mount /dev/hda1 /mnt
```

If you have made separate partitions for certain directories in the root filesystem, e.g. `/home`, you can also initialize them and mount them at this point. For example:

```
# mkfs -t ext3 /dev/hda2
# mkdir /mnt/home
# mount /dev/hda2 /mnt/home
```

Finally, you will have to mount your source media. If you use a CD-ROM, this is easy. Suppose that `/dev/hdc` is the CD-ROM device node, you can mount the CD-ROM with:

```
# mount /dev/hdc /var/log/mount
```

Mounting the installation medium through NFS requires some more steps. First of all, you will have to load the network disk. You can do this by running the **network** command and inserting a network disk floppy. You can also load this disk from another medium, for example from an USB stick. Suppose that you have mounted an USB stick on /var/log/mount, you could load the network disk with: **network /var/log/mount/network.dsk**. After loading the network disk, you will have to configure the network interface. If the NFS server is on the same network, you only have to assign an IP address to the network interface. For example, to use the address *192.168.2.201*, you could execute the following command:

```
# ifconfig eth0 192.168.2.201
```

You can then load the portmapper, which is necessary for the NFS client to function correctly:

```
# /sbin/rpc.portmap
```

If the portmapper started correctly, you can finally mount the NFS volume. Suppose, that you want to mount *192.168.2.1:/home/pub/slackware-current*, you could issue the following command:

```
# mount -r -t nfs -o nolock 192.168.2.1:/home/pub/slackware-current /var/log/mount
```

If there were no errors when the volume was mounted, it should be accessible through `/var/log/mount`

Installing packages

Everything is now set up to start installing packages from the installation medium. Since **installpkg** is available during the installation, you can use it to install Slackware Linux packages. To install it to the target partition(s) that is mounted on `/mnt`, add the *-root*. The following command installs all packages from the source medium:

```
# installpkg -root /mnt /var/log/mount/slackware/*/*.tgz
```

If you created tagfiles to define which packages should be installed, it is also possible to use them now. Suppose that you have stored a tagfile for each disk set in `/usr/share/tagfiles/small-desktop`, you can install packages based on the tagfiles in the following manner:

```
# for p in [a-z]*; do
 installpkg -root /mnt -tagfile /usr/share/tagfiles/small-desktop/$p/tagfile /var/log/mo
done
```

Post-install configuration

fstab

One of the necessary configuration steps is to create a fstab file, so that the system can look up what volumes need mounting. The format of the `/etc/fstab` file is described in the section called *The fstab file* in Chapter 8. As a bare minimum you will have to add entries for the `/` filesystem `/proc` pseudo-filesystem, the `devpts` pseudo-filesystem, and the swap partition.

With the sample partitioning used earlier in this chapter, you could create a `/etc/fstab` like this:

```
#  cat > /mnt/etc/fstab << EOF
/dev/hda2  swap          swap      defaults       0   0
/dev/hda1        /             ext3      defaults         1   1
devpts           /dev/pts      devpts    gid=5,mode=620   0   0
proc             /proc         proc      defaults         0   0
```

LILO

To make the system bootable you will have to configure and install the Linux Loader (LILO). The configuration of LILO is covered in the section called *The bootloader* in Chapter 20. For this section we will just show an example LILO configuration, that can be used with the partition layout described in this chapter. The first step is to create the `/etc/lilo.conf` file:

```
# cat > /mnt/etc/lilo.conf << EOF
boot = /dev/hda
vga = normal
timeout = 50
image=/boot/vmlinuz
 root = /dev/hda2
 label = Slackware
 read-only
EOF
```

You can then install LILO with `/mnt` as the LILO root. With `/mnt` as the root, `/etc/lilo.conf` from the target partition will be used:

```
# lilo -r /mnt
```

Networking

Configuration of networking in Slackware Linux is covered in Chapter 23. This section will cover one example of a host that will use DHCP to get an IP address. The `/etc/networks` file contains information about known Internet networks. Because we will get network information via DHCP, we will just use *127.0.0.1* as the local network.

```
# cat > /mnt/etc/networks << EOF
loopback        127.0.0.0
localnet        127.0.0.0
EOF
```

Since we will get a hostname via DHCP, we will set up a temporary hostname:

```
# cat > /mnt/etc/HOSTNAME << EOF
sugaree.example.net
EOF
```

Now that we have set up the hostname, we should also take care of making the hostname and *localhost* resolvable, by creating a `/etc/hosts` database:

```
# cat > /mnt/etc/hosts << EOF
127.0.0.1 localhost
127.0.0.1 sugaree.example.net sugaree
EOF
```

We do not have to create a `/etc/resolv.conf`, since it will automatically be created by **dhcpcd**, the DHCP client. So, finally, we can set up the interface in `/etc/rc.d/rc.inet1.conf`:

```
# cat > /mnt/etc/rc.d/rc.inet1.conf << EOF
IPADDR[0]=""
NETMASK[0]=""
USE_DHCP[0]="yes"
DHCP_HOSTNAME[0]=""
EOF
```

Tuning startup scripts

The final step is to decide which startup scripts should be started. The number of services that are available depends on what packages you have installed. You can get simply get a list of available scripts with **ls**:

```
# ls -l /mnt/etc/rc.d/rc.*
```

If the executable bits are set on a script, it will be started, otherwise it will not. Obviously you should keep essential scripts, like the runlevel scripts executable. You can set the executable bit on a script with:

```
# chmod +x /etc/rc.d/rc.scriptname
```

And remove it with:

```
# chmod -x /etc/rc.d/rc.scriptname
```

Configuring the dynamic linker run-time bindings

GNU/Linux uses a cache for loading dynamic libraries, besides that many programs rely on generic version numbers of libraries (e.g. `/usr/lib/libgtk-x11-2.0.so`, rather than `/usr/lib/libgtk-x11-2.0.so.0.600.8`). The cache and library symlinks can be updates in one **ldconfig** run:

```
# chroot /mnt /sbin/ldconfig
```

You may not know the **chroot** command, it is a command that executes a command with a different root than the active root. In this example the root directory is changed to `/mnt`, and from there Linux runs **/sbin/ldconfig**. After the command has finished the system will return to the shell, which uses the original root. To use **chroot** this **ldconfig** command has to be executed once first, because without initializing the dynamic linker run-time bindings most commands will not execute, since the system can not resolve the library dependencies.

Setting the root password

Now that the dynamic library cache and links are set up, you can execute commands on the installed system. We will make use of that during this step to set the *root* password. The **password** can be used to set the password for an existing user (the *root* user is part of the initial `/etc/passwd` file). We will use the **chroot** again to execute the command on the target partition:

```
# chroot /mnt /usr/bin/passwd root
```

Setting the timezone

On UNIX-like systems setting the timezone is important. For instance, the timezone is used by NTP to synchronize the system time correctly, or by different networking programs to compute the time difference between a client and a server. On GNU/Linux the timezone can be set by linking `/etc/localtime` to a timezone file. You can find the timezone for your region by browsing the `/mnt/usr/share/zoneinfo` directory. For example to use *Europe/Amsterdam* as the timezone, you can execute the following commands:

```
# cd /mnt
# rm -rf etc/localtime
# ln -sf /usr/share/zoneinfo/Europe/Amsterdam etc/localtime
```

After setting the time zone, the system still does not know whether the hardware clock is set to the local time, or to the Coordinated Universal Time (UTC). If you use another operating system on the same machine that does not use use UTC it is best to set the hardware clock to the local time. On UNIX(-like) systems it is the custom to set the system time to UTC. You can set what time the system clock uses, by creating the file `/etc/hardwareclock`, and put the word *localtime* in it if your clock is set to the local time, or *UTC* when it is set to UTC. For example, to use UTC, you could create the file in the following manner:

```
# cat > /mnt/etc/hardwareclock << EOF
UTC
EOF
```

Creating the X11 font cache

If you are going to use X11, you will have to initialize the font cache for TrueType and Type1 fonts. This can be done with the **fc-cache** command:

```
# chroot /mnt /usr/X11R6/bin/fc-cache
```

Automated installation script

It is easy to combine the steps of a custom installation into one script, which performs the steps automatically. This is ideal for making default server installs or doing mass roll-outs of Linux clients. This sections contains a sample script that was written by William Park. It is easy to add an installation script to the Slackware Linux media, especially if you use an installation CD-ROM or boot the installation from an USB stick.

The installation system is stored in one compressed image file, that is available on distribution media as `isolinux/initrd.img`. You can make a copy of this image to your hard disk, and decompress it with **gunzip**:

```
# mv initrd.img initrd.img.gz
# gunzip initrd.img.gz
```

After decompressing the image, you can mount the the file as a disk, using the loopback device:

```
# mount -o loop initrd.img /mnt/hd
```

You can now add a script to the initrd file by adding it to the directory structure that is available under the mount point. After making the necessary changes, you can unmount the filesystem and compress it:

```
# umount /mnt/hd
# gzip initd.img
# mv initrd.img.gz initrd.img
```

You can now put the new `initrd.img` on the installation medium, and test the script.

```
#! /bin/sh
# Copyright (c) 2003-2005 by William Park <opengeometry@yahoo.ca>.
# All rights reserved.
#
# Usage: slackware-install.sh

rm_ln ()  # Usage: rm_ln from to
{
 rm -rf $2; ln -sf $1 $2
}


################################################################################
echo "Partitioning harddisk..."

(   echo -ne "n\np\n1\n\n+1000M\n" # /dev/hda1 --> 1GB swap
 echo -ne "n\np\n2\n\n+6000M\n" # /dev/hda2 --> 6GB /
 echo -ne "t\n1\n82\nw\n"
) | fdisk /dev/hda

mkswap /dev/hda1  # swap
swapon /dev/hda1

mke2fs -c /dev/hda2  # /
mount /dev/hda2 /mnt


################################################################################
echo "Installing packages..."

mount -t iso9660 /dev/hdc /cdrom # actually, /var/log/mount
cd /cdrom/slackware
for p in [a-z]*; do  # a, ap, ..., y
 installpkg -root /mnt -priority ADD $p/*.tgz
done

cd /mnt


################################################################################
echo "Configuring /dev/* stuffs..."
```

```
rm_ln psaux dev/mouse  # or, 'input/mice' for usb mouse
rm_ln ttyS0 dev/modem
rm_ln hdc dev/cdrom
rm_ln hdc dev/dvd


################################################################################
echo "Configuring /etc/* stuffs..."

cat > etc/fstab << EOF
/dev/hda1   swap         swap     defaults          0  0
/dev/hda2   /            ext2     defaults          1  1
devpts      /dev/pts     devpts   gid=5,mode=620    0  0
proc        /proc        proc     defaults          0  0
#
/dev/cdrom  /mnt/cdrom   iso9660  noauto,owner,ro   0  0
/dev/fd0    /mnt/floppy  auto     noauto,owner      0  0
tmpfs       /dev/shm     tmpfs    noauto            0  0
EOF

cat > etc/networks << EOF
loopback 127.0.0.0
localnet 192.168.1.0
EOF
cat > etc/hosts << EOF
127.0.0.1 localhost
192.168.1.1 node1.example.net node1
EOF
cat > etc/resolv.conf << EOF
search example.net
nameserver 127.0.0.1
EOF
cat > etc/HOSTNAME << EOF
node1.example.net
EOF

## setup.05.fontconfig
chroot . /sbin/ldconfig  # must be run before other program
chroot . /usr/X11R6/bin/fc-cache

chroot . /usr/bin/passwd root

## setup.06.scrollkeeper
chroot . /usr/bin/scrollkeeper-update

## setup.timeconfig
rm_ln /usr/share/zoneinfo/Canada/Eastern  etc/localtime
cat > etc/hardwareclock << EOF
localtime
EOF

## setup.liloconfig
cat > etc/lilo.conf << EOF
boot=/dev/hda
delay=100
vga=normal # 80x25 char
# VESA framebuffer console:
```

```
#   pixel char 8bit 15bit 16bit 24bit
#   ----- ---- ---- ----- ----- -----
#   1600x1200  796 797 798 799
#   1280x1024 160x64 775 793 794 795
#   1024x768 128x48 773 790 791 792
#   800x600 100x37 771 787 788 789
#   640x480 80x30 769 784 785 786
image=/boot/vmlinuz  # Linux
 root=/dev/hda2
 label=bare.i
 read-only
# other=/dev/hda1  # Windows
#     label=win
#     table=/dev/hda
EOF
lilo -r .

## setup.xwmconfig
rm_ln xinitrc.fvwm95  etc/X11/xinit/xinitrc


################################################################################
echo "Configuring /etc/rc.d/rc.* stuffs..."

cat > etc/rc.d/rc.keymap << EOF
#! /bin/sh
[ -x /usr/bin/loadkeys ] && /usr/bin/loadkeys us.map
EOF
chmod -x etc/rc.d/rc.keymap

## setup.mouse
cat > etc/rc.d/rc.gpm << 'EOF'
#! /bin/sh
case $1 in
 stop)
     echo "Stopping gpm..."
     /usr/sbin/gpm -k
     ;;
 restart)
     echo "Restarting gpm..."
     /usr/sbin/gpm -k
     sleep 1
     /usr/sbin/gpm -m /dev/mouse -t ps2
     ;;
 start)
     echo "Starting gpm..."
     /usr/sbin/gpm -m /dev/mouse -t ps2
     ;;
 *)
     echo "Usage $0 {start|stop|restart}"
     ;;
esac
EOF
chmod +x etc/rc.d/rc.gpm

## setup.netconfig
cat > etc/rc.d/rc.inet1.conf << EOF
```

```
IPADDR=(192.168.1.1)  # array variables
NETMASK=(255.255.255.0)
USE_DHCP=()   # "yes" or ""
DHCP_HOSTNAME=()
GATEWAY=""
DEBUG_ETH_UP="no"
EOF

cat > etc/rc.d/rc.netdevice << EOF
/sbin/modprobe 3c59x
EOF
chmod +x etc/rc.d/rc.netdevice

## setup.setconsolefont
mv etc/rc.d/rc.font{.sample,}
chmod -x etc/rc.d/rc.font

## setup.services
chmod +x etc/rc.d/rc.bind
chmod +x etc/rc.d/rc.hotplug
chmod +x etc/rc.d/rc.inetd
chmod +x etc/rc.d/rc.portmap
chmod +x etc/rc.d/rc.sendmail
#
chmod -x etc/rc.d/rc.atalk
chmod -x etc/rc.d/rc.cups
chmod -x etc/rc.d/rc.httpd
chmod -x etc/rc.d/rc.ip_forward
chmod -x etc/rc.d/rc.lprng
chmod -x etc/rc.d/rc.mysqld
chmod -x etc/rc.d/rc.pcmcia
chmod -x etc/rc.d/rc.samba
chmod -x etc/rc.d/rc.sshd
```

II. GNU/Linux Basics

Chapter 7. The Bash shell

Introduction

The shell is the traditional interface used by UNIX and GNU/Linux. In contrast to the X Window System it is an interface that works with commands. In the beginning this can be a bit awkward, but the shell is very powerful. Even in these days the shell is almost unavoidable ;).

The default shell on Slackware Linux is Bash. Bash means “Bourne-Again SHell”, which is a pun on the name of one of the traditional UNIX shells, the “Bourne Shell”. Slackware Linux also provides some other shells, but Bash is the main topic of this chapter.

Starting the shell

The procedure for starting the shell depends on whether you use a graphical or text-mode login. If you are logging on in text-mode the shell is immediately started after entering the (correct) password. If you are using a graphical login manager like gdm, log on as you would normally, and look in your window manager or desktop environment menu for an entry named “XTerm”. XTerm is a terminal emulator, after the terminal emulator is started the shell comes up.

The shell might remind some people of MS-DOS. Be happy, it has nothing to do with DOS, the only similarity is that you can enter commands ;).

Shell basics

This chapter might be a difficult to read for the first time, because you might not know any shell commands. Many important commands are described in the next chapters, but those chapters are not really useful without any knowledge of the shell. So, it is not a bad idea to browse quickly through this chapter, and the next few chapters, to get an idea what this shell thing is all about. After that quick overview you should be able to understand this chapter.

Executing commands

The most important job for the shell is to execute your commands. Let’s look at a simple example. Most UNIX variants have a command named **whoami**, which shows as which user you are logged in. Try typing **whoami**, and press the <Enter> after that. The <Enter> tells the shell that it should execute the command that you have typed on the current line. The output looks like this:

```
daniel@tazzy:~$ whoami
daniel
daniel@tazzy:~$
```

As you can see the control is handed back to the shell after the command is finished.

Browsing through shell commands

It often happens that you have to execute commands that you executed earlier. Fortunately, you do not have to type them all over again. You can browse through the history of executed commands with the up and down arrows. Besides that it is also possible to search for a command. Press <Control> + <r> and start typing the command you want to execute. You will notice that bash will display the first match it can find. If this is not the match you were looking for you can continue typing the command (till it is unique and a match appears), or press <Control> + <r> once more to get the next match. When you have found the command you were looking for, you can execute it by pressing <Enter>.

Completion

Completion is one of the most useful functionalities of UNIX-like shells. Suppose that you have a directory with two files named `websites` and `recipe`. And suppose you want to **cat** the file `websites` (**cat** shows the contents of a file), by specifying `websites` as a parameter to cat. Normally you would type “cat websites”, and execute the command. Try typing “cat w”, and hit the <Tab> key. Bash will automatically expand what you typed to “cat websites”.

But what happens if you have files that start with the same letter? Suppose that you have the `recipe1.txt` and `recipe2.txt` files. Type “cat r” and hit <Tab>, Bash will complete the filename as far as it can. It would leave you with “cat recipe”. Try hitting <Tab> again, and Bash will show you a list of filenames that start with “recipe”, in this case both recipe files. At this point you have to help Bash by typing the next character of the file you need. Suppose you want to **cat** `recipe2`, you can push the <2> key. After that there are no problems completing the filename, and hitting <Tab> completes the command to “cat recipe2.txt”.

It is worth noting that completion also works with commands. Most GNU/Linux commands are quite short, so it will not be of much use most of the time.

It is a good idea to practice a bit with completion, it can save alot of keystrokes if you can handle completion well. You can make some empty files to practive with using the **touch** command. For example, to make a file named `recipe3.txt`, execute **touch recipe3.txt**.

Wildcards

Most shells, including Bash, support wildcards. Wildcards are special characters that can be used to do pattern matching. The table listed below displays some commonly used wildcards. We are going to look at several examples to give a general idea how wildcards work.

Table 7-1. Bash wildcards

Wildcard	Matches
*	A string of characters
?	A single character
[]	A character in an array of characters

Matching a string of characters

As you can see in the table above the “*” character matches a string of characters. For example, **.html* matches everything ending with *.html*, *d*.html* matches everything starting with a *d* and ending with *.html*.

Suppose that you would like to list all files in the current directory with the *.html* extension, the following command will do the job:

```
$ ls *.html
book.html         installation.html      pkgmgmt.html  usermgmt.html
filesystem.html   internet.html          printer.html  xfree86.html
gfdl.html         introduction.html      proc.html
help.html         slackware-basics.html  shell.html
```

Likewise we could remove all files starting with an *in*:

```
$ rm in*
```

Matching single characters

The “?” wildcard works as the “*” wildcard, but matches single characters. Suppose that we have three files, `file1.txt`, `file2.txt` and `file3.txt`. The string *file?.txt* matches all three of these files, but it does not match `file10.txt` (“10” are two characters).

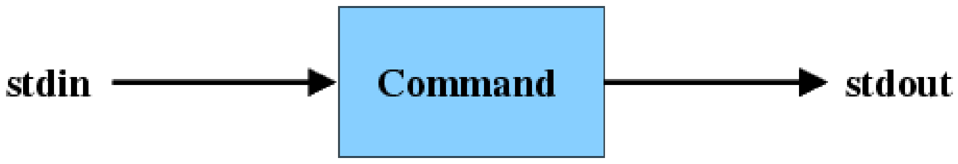
Matching characters from a set

The “[]” wildcard matches every character between the brackets. Suppose we have the files from the previous example, `file1.txt`, `file2.txt` and `file3.txt`. The string *file[23].txt* matches `file2.txt` and `file3.txt`, but not `file1.txt`.

Redirections and pipes

One of the main features of UNIX-like shells are redirections and pipes. Before we start to look at both techniques we have to look how most UNIX-like commands work. When a command is not getting data from a file, it will open a special pseudo-file named *stdin*, and wait for data to appear on it. The same principle can be applied for command output, when there is no explicit reason for saving output to a file, the pseudo-file *stdout* will be opened for output of data. This principle is shown schematically in Figure 7-1

Figure 7-1. Standard input and output



You can see *stdin* and *stdout* in action with the **cat** command. If cat is started without any parameters it will just wait for input on *stdin* and output the same data on *stdout*. If no redirection is used keyboard input will be used for *stdin*, and *stdout* output will be printed to the terminal:

```
$ cat
Hello world!
Hello world!
```

As you can see cat will print data to *stdout* after inputting data to *stdin* using the keyboard.

Redirection

The shell allows you to take use of *stdin* and *stdout* using the "<" and ">". Data is redirected in which way the sharp bracket points. In the following example we will redirect the md5 summaries calculated for a set of files to a file named `md5sums`:

```
$ md5sum * > md5sums
$ cat md5sums
6be249ef5cacb10014740f61793734a8  test1
220d2cc4d5d5fed2aa52f0f48da38ebe  test2
631172a1cfca3c7cf9e8d0a16e6e8cfe  test3
```

As we can see in the **cat** output the output of the **md5sum** * output was redirected to the `md5sums` file. We can also use redirection to provide input to a command:

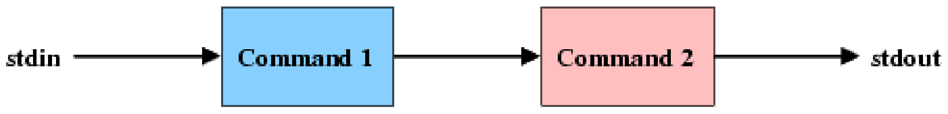
```
$ md5sum < test1
6be249ef5cacb10014740f61793734a8  -
```

This feeds the contents of the `test1` to **md5sum**.

pipes

You can also connect the input and output of commands using so-called *pipes*. A pipe between commands can be made with the "|" character. Two or more combined commands are called a *pipeline*. Figure 7-2 shows a schematic overview of a pipeline consisting of two commands.

Figure 7-2. A pipeline



The "syntax" of a pipe is: **command1 | command2 ... | commandn**. If you know how the most basic UNIX-like commands work you can now let these commands work together. Let's look at a quick example:

```
$ cat /usr/share/dict/american-english | grep "aba" | wc -l
123
```

The first command, **cat**, reads the dictionary file `/usr/share/dict/american-english`. The output of the **cat** command is piped to **grep**, which prints out all files containing the phrase "aba". In turn, the output of "grep" is piped to **wc -l**, which counts the number of lines it receives from *stdin*. Finally, when the stream is finished **wc** prints the number of lines it counted. So, combined three commands to count the number of words containing the phrase "aba" in this particular dictionary.

There are hundreds of small utilities that handle specific tasks. As you can imagine, together these commands provide a very powerful toolbox by making combinations using pipes.

Chapter 8. Files and directories

Introduction

UNIX-like operating systems use a hierarchical filesystem to store files and directories. Directories can contain files and other directories, the top directory (`/`) is named the root directory (not to be confused with the `/root` directory). Most filesystems also support file links (which provide alternative names for a file) and soft links. Other filesystems can be “connected” to an arbitrary directory. This process is named “mounting”, and the directory in which the filesystem is mounted is named the “mount point”.

This chapter covers the basic navigation of the filesystem, commands which are used to remove and create directories, filesystem permissions, links and mounting.

The basics

pwd

pwd(1) is a simple utility which shows the directory you are currently working in. The **pwd** does not require any parameters. This is an example output of **pwd**:

```
$ pwd
/home/danieldk
```

ls

ls is similar to the **dir** command in DOS and Windows. **ls** can be used to display files and directories located in specific directories. Running the **ls** command without any parameters shows the contents of the current directory:

```
$ ls
COPYING  CVS  Makefile  README  html  images  pdf  src  tex
```

Naturally it is also possible to show the contents of other directories. You can do this by specifying the path as a parameter tot the **ls** command:

```
$ ls /
bin   cdrom   dev  floppy  initrd  lost+found  opt   root  sys  usr  windows
boot  cdrom1  etc  home    lib     mnt         proc  sbin  tmp  var
```

A disadvantage of the default output is that it provides little information about files and directories. For example, it is not possible to see whether some entry is a file or directory, what size a file is, or who the owner of the file is. The **ls** has the *-l* parameter to show more information:

```
$ ls -l
total 52
-rw-r--r--    1 daniel   daniel      20398 Jul 16 14:28 COPYING
drwxr-xr-x    2 daniel   daniel       4096 Jul 16 14:28 CVS
-rw-r--r--    1 daniel   daniel        768 Jul 16 14:28 Makefile
```

```
-rw-r--r--    1 daniel   daniel        408 Jul 16 14:28 README
drwxr-xr-x    3 daniel   daniel       4096 Jul 16 14:28 html
drwxr-xr-x    3 daniel   daniel       4096 Jul 16 14:28 images
drwxr-xr-x    3 daniel   daniel       4096 Jul 16 14:28 pdf
drwxr-xr-x    3 daniel   daniel       4096 Jul 20 00:11 src
drwxr-xr-x    3 daniel   daniel       4096 Jul 16 14:28 tex
```

cd

Another important command is the **cd** command. It can be used to change the current working directory:

```
$ cd /home/danieldk/
```

With the **pwd** command you can see it worked:

```
$ pwd
/home/danieldk
```

mkdir

As you might have guessed, the **mkdir**(1) command can be used to create directories. For example:

```
$ pwd
/home/danieldk
$ mkdir test
$ cd test
$ pwd
/home/danieldk/test
```

It might happen that you want to create a directory in a parent directory which does not exist yet. For example, if you want to create the `test2/hello/` directory, but the `test2` directory does not yet exist. In this case you can make both directories with only one **mkdir** command:

```
$ mkdir -p test2/hello
```

cp

Files can be copied with the **cp**(1) command, the basic syntax is **cp source destination**. For example, suppose that we have a file named `memo` which we would like to copy to the `writings` directory. You can do this with the following command:

```
$ cp memo writings/
```

It is also possible to copy a file in the same directory. For example, if we would like to make a new memo based on `memo`, named `memo2`, we could execute the following command:

```
$ cp memo memo2
```

It is also possible to copy directories recursively, this can be done by adding the *-r* parameter. The following command copies the `memos` directory, and all subdirectories, and (sub)files to the `writings` directory:

```
$ cp -r memos writings/
```

mv

The **mv**(1) command is comparable to **cp**, but it is used to move files. Suppose that we have the same situation as in the first **cp** example, but you would rather like to move `memo` to the `writings` directory. The following command would do that:

```
$ mv memo writings/
```

It is also possible to move directories. But, **mv** always works recursively. For example, the following command will move the `memos` directory to the `writings` directory:

```
$ mv memos writings/
```

rm

The **rm**(1) command is used to remove files and directories. Let's look at a simple example:

```
$ rm hello.c
```

This commands removes the file `hello.c`. Sometimes the **rm** asks for a confirmation. You can ignore this with the *-f* parameter:

```
$ rm -f *
```

This command removes all files in the current directory without asking for confirmation. It is also possible to delete directories or even whole directory trees. **rm** provides the *-r* parameter to do this. Suppose that we want to delete the `ogle` directory and all subdirectories and files in that directory, this can be done in the following way:

```
$ rm -r -f ogle/
```

Many commands allow you to combine parameters. The following example is equivalent to the last one:

```
$ rm -rf ogle/
```

Permissions

A short introduction

Every file in GNU/Linux has permissions. As you might have noticed, file permissions can be shown with the **ls -l** command:

```
$ ls -l logo.jpg
-rw-r--r--    1 danieldk users        9253 Dec 23 19:12 logo.jpg
```

The permissions are shown in the first column. Permissions that can be set are read(r), write(w) and execute(x). These permissions can we set for three classes: owner(u), group(g) and others(o). The permissions are visible in the second to ninth character in the first column. These nine characters are divided in three groups. The first three characters represent the permissions for the owner, the next three characters represent the permissions for the group, finally the last three characters represent the permissions for other users. Thus, the example file shown above can be written to by the owner and can be read by all three classes of users (owner, group and others).

Each GNU/Linux system has many distinct users (a list of users can be found in `/etc/passwd`), and a user can be a member of certain groups. This kind of user management provides makes it possible to manage detailed permissions for each file. In the example shown above *danieldk* is the owner of the file and group permissions apply to the group *users*. In this example groups rights do not differ from the rights of other users.

chown

The **chown**(1) command is used to set the file owner and to which group group permissions should apply to. Suppose we want to make *danieldk* the owner of the file `logo2.jpg`, this can be done with the **chown**:

```
# chown danieldk logo2.jpg
```

Using the **ls** we can see that the owner is now *danieldk*:

```
# ls -l logo2.jpg
-rw-r--r--    1 root     root        9253 Dec 29 11:35 logo2.jpg
# chown danieldk logo2.jpg
# ls -l logo2.jpg
-rw-r--r--    1 danieldk root        9253 Dec 29 11:35 logo2.jpg
```

But group permissions still apply for the *root* group. This group can be changed by adding a colon after the owner, followed by the name of the group (in this example the group is nedslackers):

```
# chown danieldk:nedslackers logo2.jpg
# ls -l logo2.jpg
-rw-r--r--    1 danieldk nedslackers        9253 Dec 29 11:35 logo2.jpg
```

It is also possible to change ownership recursively, this can be done with the recursive (*-R*) parameter:

```
# chown -R danieldk:users oggs/
```

chmod

File permissions can be manipulated using the **chmod**(1) command. The most basic syntax of chmod is **chmod [u,g,o][+/-][r,w,x] filename**. The first parameter consists of the following elements: 1. which classes this manipulation permission applies to, 2. if the permissions should be added (+) or removed (-), and 3. which permissions should be manipulated. Suppose we want to make the file `memo` writable for the owner of the file and the groups for which the group permissions apply. This can be done with the following command:

```
$ chmod ug+w memo
```

As you can see below the `memo` is now writable for the file owner and group:

```
$ ls -l memo
-r--r--r--    1 daniel   users          12 Mar  9 16:28 memo
bash-2.05b$ chmod ug+w memo
bash-2.05b$ ls -l notes
-rw-rw-r--    1 daniel   users          12 Mar  9 16:28 memo
```

Just like the **chown** command it is also possible to do recursive (*-R*) operations. In the following example the `secret/`, including subdirectories and files in this directory, is made unreadable for the group set for this directory and other users:

```
$ chmod -R go-r secret/
```

Archives

Introduction

Sooner or later a GNU/Linux user will encounter tar archives, tar is the standard format for archiving files on GNU/Linux. It is often used in conjunction with **gzip** or **bzip2**. Both commands can compress files and archives. Table 8-1 lists frequently used archive extensions, and what they mean.

Table 8-1. Archive file extensions

Extension	Meaning
.tar	An uncompressed tar archive
.tar.gz	A tar archive compressed with gzip
.tgz	A tar archive compressed with gzip
.tar.bz2	A tar archive compressed with bzip2
.tbz	A tar archive compressed with bzip2

The difference between **bzip2** and **gzip** is that **bzip2** can find repeating information in larger blocks, resulting in better compression. But **bzip2** is also a lot slower, because it does more data analysis.

Extracting archives

Since many software and data in the GNU/Linux world is archived with **tar** it is important to get used to extracting tar archives. The first thing you will often want to do when you receive a tar archive is to list its contents. This can be achieved by using the *t* parameter. However, if we just execute **tar** with this parameter and the name of the archive it will just sit and wait until you enter something to the standard input:

```
$ tar t test.tar
```

This happens because **tar** reads data from its standard input. If you forgot how redirection works, it is a good idea to reread the section called *Redirections and pipes* in Chapter 7. Let's see what happens if we redirect our tar archive to tar:

```
$ tar t < test.tar
test/
test/test2
test/test1
```

That looks more like the output you probably expected. This archive seems to contain a directory `test`, which contains the files `test2` and `test2`. It is also possible to specify the archive file name as an parameter to **tar**, by using the *f* parameter:

```
$ tar tf test.tar
test/
test/test2
test/test1
```

This looks like an archive that contains useful files ;). We can now go ahead, and extract this archive by using the *x* parameter:

```
$ tar xf test.tar
```

We can now verify that tar really extracted the archive by listing the contents of the directory with **ls**:

```
$ ls test/
test1  test2
```

Extracting or listing files from a gzipped or bzipped archive is not much more difficult. This can be done by adding a *z* or *b* for respectively archives compressed with **gzip** or **bzip2**. For example, we can list the contents of a gzipped archive with:

```
$ tar ztf archive2.tar.gz
```

And a bzipped archive can be extracted with:

```
$ tar bxf archive3.tar.bz2
```

Creating archives

You can create archives with the *c* parameter. Suppose that we have the directory `test` shown in the previous example. We can make an archive with the `test` directory and the files in this directory with:

```
$ tar cf important-files.tar test
```

This will create the `important-files.tar` archive (which is specified with the *f* parameter). We can now verify the archive:

```
$ tar tf important-files.tar
test/
test/test2
test/test1
```

Creating a gzipped or bzipped archive goes along the same lines as extracting compressed archives: add a `z` for gzipping an archive, or `b` for bzipping an archive. Suppose that we wanted to create a **gzip** compressed version of the archive created above. We can do this with:

```
tar zcf important-files.tar.gz test
```

Extended attributes

Introduction

Extended attributes (EAs) are relatively new on GNU/Linux. Extended attributes are a special kind of values that are associated with a file or directory. EAs provide the means to add extra attributes besides the common attributes (modification time, traditional file permissions, etc.). For example, one could add the attribute “Photographer” to a collection of JPEG files. Extended attributes are not physically stored in the file, but as meta-data in the filesystem.

Extended attributes are only supported by 2.6.x and newer 2.4.x kernels. Besides that they are not supported on all filesystems, the commonly used Ext2, Ext3 and XFS filesystems do support extended attributes.

Installing the necessary utilities

The extended attribute software is available in Slackware Linux through the *xfsprogs* package, which can be found in the “a” disk set. Don’t be misled by the name of the package, the extended attribute tools also work with other filesystems, like Ext3.

Showing extended attributes

Extended attributes can be queried using the **getfattr** command. Just using getfattr with a file as a parameter will show the attributes that are known for that particular file, without the values set for the attributes. For example:

```
$ getfattr note.txt
# file: note.txt
user.Author
```

This file has one extended attribute, “user.Author”. An attribute has the following form: “namespace.attribute”. There are four defined namespaces: “security”, “system”, “trusted”, and “user”. The role of these namespaces are described in the attr(5) manual page. The “user” namespace is of particular interest to us, because this namespace is used to assign arbitrary attributes to files.

The values associated with an attribute can be shown using the `-d` (dump) parameter. For example:

```
$ getfattr -d note.txt
# file: note.txt
user.Author="Daniel"
```

In this example the attribute “user.Author” has the value “Daniel” for the file `note.txt`.

Setting extended attributes

Attributes are set with the **setfattr** command. An attribute can be added to a file using the `-n` (name) parameter, be sure to specify the namespace and the attribute name, for example:

```
$ setfattr -n user.Author note2.txt
```

The value of the attribute can be set using the `-v` (value) parameter:

```
$ setfattr -n user.Author -v Mike note2.txt
```

But it is also possible to add an attribute and setting its value in one run, by specifying both the `-n` and `-v` parameters. For example, the following command adds the MD5 sum of the file as an extended attribute:

```
$ setfattr -n user.MD5 -v `md5sum note2.txt` note2.txt
$ getfattr -d note2.txt
# file: note2.txt
user.Author="Mike"
user.MD5="78be7a3148027ae7a897aad95e7d9c58"
```

Mounting filesystems

Introduction

Like most Unices Linux uses a technique named “mounting” to access filesystems. Mounting means that a filesystem is connected to a directory in the root filesystem. One could for example mount a CD-ROM drive to the `/mnt/cdrom` directory. Linux supports many kinds of filesystems, like Ext2, Ext3, ReiserFS, JFS, XFS, ISO9660 (used for CD-ROMs), UDF (used on some DVDs) and DOS/Windows filesystems, like FAT, FAT32 and NTFS. These filesystems can reside on many kinds of media, for example hard drives, CD-ROMs and Flash drives. This section explains how filesystems can be mounted and unmounted.

mount

The **mount**(8) is used to mount filesystems. The basic syntax is: “mount /dev/devname /mountpoint”. The device name can be any block device, like hard disks or CD-ROM drives. The mount point can be an arbitrary point in the root filesystem. Let’s look at an example:

```
# mount /dev/cdrom /mnt/cdrom
```

This mounts the `/dev/cdrom` on the `/mnt/cdrom` mountpoint. The `/dev/cdrom` device name is normally a link to the real CD-ROM device name (for example, `/dev/hdc`). As you can see, the concept is actually very simple, it just takes some time to learn the device names ;). Sometimes it is necessary to specify which kind of filesystem you are trying to mount. The filesystem type can be specified by adding the `-t` parameter:

```
# mount -t vfat /dev/sda1 /mnt/flash
```

This mounts the vfat filesystem on `/dev/sda1` to `/mnt/flash`.

umount

The **umount**(1) command is used to unmount filesystems. **umount** accepts two kinds of parameters, mount points or devices. For example:

```
# umount /mnt/cdrom
# umount /dev/sda1
```

The first command unmounts the filesystem that was mounted on `/mnt/cdrom`, the second commands unmounts the filesystem on `/dev/sda1`.

The fstab file

The GNU/Linux system has a special file, `/etc/fstab`, that specifies which filesystems should be mounted during the system boot. Let's look at an example:

```
/dev/hda10      swap            swap        defaults          0   0
/dev/hda5       /               xfs         defaults          1   1
/dev/hda6       /var            xfs         defaults          1   2
/dev/hda7       /tmp            xfs         defaults          1   2
/dev/hda8       /home           xfs         defaults          1   2
/dev/hda9       /usr            xfs         defaults          1   2
/dev/cdrom      /mnt/cdrom      iso9660     noauto,owner,ro   0   0
/dev/fd0        /mnt/floppy     auto        noauto,owner      0   0
devpts          /dev/pts        devpts      gid=5,mode=620    0   0
proc            /proc           proc        defaults          0   0
```

As you can see each entry in the `fstab` file has five entries: fs_spec, fs_file, fs_vfstype, fs_mntops, fs_freq, and fs_passno. We are now going to look at each entry.

fs_spec

The fs_spec option specifies the block device, or remote filesystem that should be mounted. As you can see in the example several /dev/hda partitions are specified, as well as the CD-ROM drive and floppy drive. When NFS volumes are mounted an IP address and directory can be specified, for example: `192.168.1.10:/exports/data`.

fs_file

fs_file specifies the mount point. This can be an arbitrary directory in the filesystem.

fs_vfstype

This option specifies what kind of filesystem the entry represents. For example this can be: ext2, ext3, reiserfs, xfs, nfs, vfat, or ntfs.

fs_mntops

The fs_mntops option specifies which parameters should be used for mounting the filesystem. The **mount**(8) manual page has an extensive description of the available options. These are the most interesting options:

- *noauto*: filesystems that are listed in `/etc/fstab` are normally mounted automatically. When the "noauto" option is specified, the filesystem will not be mounted during the system boot, but only after issuing a **mount** command. When mounting such filesystem, only the mount point or device name has to be specified, for example: **mount /mnt/cdrom**
- *user*: adding the "user" option will allow normal users to mount the filesystem (normally only the superuser is allowed to mount filesystems).
- *owner*: the "owner" option will allow the owner of the specified device to mount the specified device. You can see the owner of a device using **ls**, e.g. **ls -l /dev/cdrom**.
- *noexec*: with this option enabled users can not run files from the mounted filesystem. This can be used to provide more security.
- *nosuid*: this option is comparable to the "noexec" option. With "nosuid" enabled SUID bits on files on the filesystem will not be allowed. SUID is used for certain binaries to provide a normal user to do something privileged. This is certainly a security threat, so this option should really be used for removable media, etc. A normal user mount will force the nosuid option, but a mount by the superuser will not!
- *unhide*: this option is only relevant for normal CD-ROMs with the ISO9660 filesystem. If "unhide" is specified hidden files will also be visible.

fs_freq

If the "fs_freq" is set to 1 or higher, it specifies after how many days a filesystem dump (backup) has to be made. This option is only used when **dump**(8) is set up correctly to handle this.

fs_passno

This field us used by **fsck**(8) to determine the order in which filesystems are checked during the system boot.

Encrypting and signing files

Introduction

There are two security mechanisms for securing files: signing files and encrypting files. Signing a file means that a special digital signature is generated for a file. You, or other persons can use the signature to verify the integrity of the file. File encryption encodes a file in a way that only a person for which the file was intended to read can read the file.

This system relies on two keys: the private and the public key. Public keys are used to encrypt files, and files can only be decrypted with the private key. This means that one can sent his public key out to other persons. Others can use this key to send encrypted files, that only the person with the private key can decode. Of course, this means that the security of this system depends on how well the private is kept secret.

Slackware Linux provides an excellent tool for signing and encrypting files, named GnuPG. GnuPG can be installed from the "n" disk set.

Generating your private and public keys

Generating public and private keys is a bit complicated, because GnuPG uses DSA keys by default. DSA is an encryption algorithm, the problem is that the maximum key length of DSA is 1024 bits, this is considered too short for the longer term. That is why it is a good idea to use 2048 bit RSA keys. This section describers how this can be done.

> **Note:** 1024-bit keys were believed to be secure for a long time. But Bernstein's paper *Circuits for Integer Factorization: a Proposal* contests this, the bottom line is that it is quite feasible for national security agencies to produce hardware that can break keys in a relatively short amount of time. Besides that it has be shown that 512-bit RSA keys can be broken in a relatively short time using common hardware. More information about these issues can by found in this e-mail to the cypherpunks list: http://lists.saigon.com/vault/security/encryption/rsa1024.html

We can generate a key by executing:

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

The first question is what kind of key you would like to make. We will choose *(4) RSA (sign only)*:

```
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) RSA (sign only)
Your selection? 4
```

You will then be asked what the size of the key you want to generate has to be. Type in *2048* to generate a 2048 bit key, and press enter to continue.

```
What keysize do you want? (1024) 2048
```

The next question is simple to answer, just choose what you like. Generally speaking it is not a bad idea to let the key be valid infinitely. You can always deactivate the key with a special revocation certificate.

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

GnuPG will then ask for confirmation. After confirming your name and e-mail address will be requested. GnuPG will also ask for a comment, you can leave this blank, or you could fill in something like “Work” or “Private”, to indicate what the key is used for. For example:

```
Real name: John Doe
Email address: john@doe.com
Comment: Work
You selected this USER-ID:
    "John Doe (Work) <john@doe.com>"
```

GnuPG will the ask you to confirm your user ID. After confirming it GnuPG will ask you to enter a password. Be sure to use a good password:

```
You need a Passphrase to protect your secret key.

Enter passphrase:
```

After entering the password twice GnuPG will generate the keys. But we are not done yet. GnuPG has only generated a key for signing information, not for encryption of information. To continue, have a look at the output, and look for the key ID. In the information about the key you will see *pub 2048R/*. The key ID is printed after this fragment. In this example:

```
public and secret key created and signed.
key marked as ultimately trusted.

pub  2048R/8D080768 2004-07-16 John Doe (Work) <john@doe.com>
     Key fingerprint = 625A 269A 16B9 C652 B953  8B64 389A E0C9 8D08 0768
```

the key ID is *8D080768*. If you lost the output of the key generation you can still find the key ID in the output of the **gpg --list-keys** command. Use the key ID to tell GnuPG that you want to edit your key:

```
$ gpg --edit-key <Key ID>
```

With the example key above the command would be:

```
$ gpg --edit-key 8D080768
```

GnuPG will now display a command prompt. Execute the **addkey** command on this command prompt:

```
Command> addkey
```

GnuPG will now ask the password you used for your key:

```
Key is protected.

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16

Enter passphrase:
```

After entering the password GnuPG will ask you what kind of key you would like to create. Choose *RSA (encrypt only)*, and fill in the information like you did earlier (be sure to use a 2048 bit key). For example:

```
Please select what kind of key you want:
   (2) DSA (sign only)
   (3) ElGamal (encrypt only)
   (4) RSA (sign only)
   (5) RSA (encrypt only)
Your selection? 5
What keysize do you want? (1024) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
```

And confirm that the information is correct. After the key is generated you can leave the GnuPG command prompt, and save the new key with the **save** command:

```
Command> save
```

Congratulations! You have now generated the necessary keys to encrypt and decrypt e-mails and files. You can now configure your e-mail client to use GnuPG. It is a good idea to store the contents of the `.gnupg` directory on some reliable medium, and store that in a safe place! If your private key is lost you can't decrypt files and messages that were encrypted with your public key. If the private key, and your password are stolen, the security of this system is completely compromised.

Exporting your public key

To make GnuPG useful, you have to give your public key to people who send you files or e-mails. They can use your public key to encrypt files, or use it to verify whether a file has a correct signature or not. The key can be exported using the *`--export`* parameter. It is also a good idea to specify the *`--output`* parameter, this will save the key in a file. The following command would save the public key of *John Doe*, used in earlier examples, to the file `key.gpg`:

```
$ gpg --output key.gpg --export john@doe.com
```

This saves the key in binary format. Often it is more convenient to use the so-called "ASCII armored output", which fits better for adding the key to e-mails, or websites. You export an ASCII armored version of the key by adding the *`--armor`* parameter:

```
$ gpg --armor --output key.gpg --export john@doe.com
```

If you look at the `key.gpg` file you will notice that the ASCII armored key is a much more comfortable format.

Signatures

With GPG you can make a signature for a file. This signature is unique, because your signature can only be made with your private key. This means that other people can check whether the file was really sent by you, and whether it was in any way altered or not. Files can be signed with the *`--detach-sign`* parameter. Let us look at an example. This command will make a signature for the `memo.txt` file. The signature will be stored in `memo.txt.sig`.

```
$ gpg --output memo.txt.sig --detach-sign memo.txt

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16

Enter passphrase:
```

As you can see, GnuPG will ask you to enter the password for your private key. After you have entered the right key the signature file (`memo.txt.sig`) will be created.

You can verify a file with its signature using the *--verify* parameter. Specify the signature file as a parameter to the *--verify* parameter. The file that needs to be verified can be specified as the final parameter:

```
$  gpg --verify memo.txt.sig memo.txt
gpg: Signature made Tue Jul 20 23:47:45 2004 CEST using RSA key ID 8D080768
gpg: Good signature from "John Doe (Work) <john@doe.com>"
```

This will confirm that the file was indeed signed by *John Doe (Work) <john@doe.com>*, with the key *8D080768*, and that the file is unchanged. Suppose the file was changed, GnuPG would have complained about it loudly:

```
$ gpg --verify memo.txt.sig memo.txt
gpg: Signature made Tue Jul 20 23:47:45 2004 CEST using RSA key ID 8D080768
gpg: BAD signature from "John Doe (Work) <john@doe.com>"
```

Encryption

One of the main features of GnuPG is encryption. Due to its use of symmetric cryptography, the person who encrypts a file and the person who decrypts a file do not need to share a key. You can encrypt a file with the private key of another person, and she can decrypt it with her private key. You can encrypt files with the *--encrypt*. If you do not specify a user ID for which the file should be encrypted, GnuPG will prompt for the user ID. You can specify the user ID with the *-r* parameter. In the following example, the file `secret.txt` will be encrypted for another person named *John Doe*:

```
$ gpg --encrypt -r "John Doe" secret.txt
```

The user ID is quoted with double quotes for making sure that the ID is interpreted as a single program argument. After the encryption is completed, the encrypted version of the file will be available as `secret.txt.gpg`.

The user who receives the file can decrypt it with the *--decrypt* parameter of the **gpg** command:

```
$ gpg --output secret.txt --decrypt secret.txt.gpg

You need a passphrase to unlock the secret key for
user: "John Doe (Work) <john@doe.com>"
2048-bit RSA key, ID 8D080768, created 2004-07-16 (main key ID EC3ED1AB)

Enter passphrase:

gpg: encrypted with 2048-bit RSA key, ID 8D080768, created 2004-07-16
      "John Doe (Work) <john@doe.com>"
```

In this example the *--output* parameter is used store the decrypted content in `secret.txt`.

Chapter 9. Text Utilities

Introduction

On of the central ideas of UNIX(-like) operating systems is that “everything is a file”. Even devices can be treated as a file. Basically there are three types of files in UNIX:

- *Binary files*, for example executables, and libraries.
- *Device files*, for example `/dev/zero`, and `/dev/hda`.
- *Text files*, pretty much anything else.

Due to the fact that text files have such an important role in UNIX-like operating systems, like GNU/Linux, this book has a separate chapter dedicated to the subject of processing text files. In the beginning its use may not be that obvious, but once you get used to these tools you will see that you will probably use some of these tools on a daily basis.

Note: This chapter relies heavily on the use of pipes and redirection, so it it a good idea to read the section called *Redirections and pipes* in Chapter 7 if you have not done that yet.

The basics

cat

The **cat** command is one of the simplest commands around. Its default behaviour is just to send anything it receives on the standard input to the standard input until the end of file character (^D) is sent to the standard input. You can see this by executing cat, and entering some text:

```
$ cat
Hello world!
Hello world!
Testing... 1 2 3
Testing... 1 2 3
```

You can also concatenate files with **cat**, by providing the files you would like to concatenate as an argument. The concatenated files will be sent to the standard output:

```
$ cat test1
This is the content of test1.
$ cat test2
This is the content of test2.
$ cat test1 test2
This is the content of test1.
This is the content of test2.
```

As you can see it is also possible to send a file to the standard output by specifying one file as **cat**'s argument. This is an alternative to redirection. For example, one could either of these command:

```
$ less < test1
$  cat test1 | less
```

echo

The **echo** is used to send something to the standard output by specifying it as an argument to **echo**. For example, one could use echo to send a simple message to the standard output:

```
$ echo "Hello world!"
Hello world!
```

wc

One of the common things people often want to do with a text is counting the number of words, or lines in a text. The **wc** command can be used for this purpose. The file to be counted can be specified as an argument to **wc**

```
$ wc essay.txt
 174 1083 8088 essay.txt
```

As you can see the default output returns three numbers. These are (in order): the *number of lines*, the *number of words*, and the *number of characters*. It is also possibly to return only one of these numbers, with respectively `-l`, `-w`, and `-m`. For example, if we only want to know the number of lines in the file, we could do the following:

```
$ wc -l essay.txt
174 essay.txt
```

In some situations you might want to use the output of **wc** in a pipeline, or as an argument to another command. The problem with specifying the file name as an argument is that **wc** will also show the name of the file (as you can see in the example above). You can work around this behavior by redirecting the file contents to **wc**. For example:

```
$  wc -l < essay.txt
174
```

tr

The **tr** is used to translate or delete characters. All uses of **tr** require one or two sets. A set is a string of characters. You can specify most characters in a set. The **tr**(1) manual page provides an overview of some character sequences with a special meaning. Table 9-1 describes some of the fequently used special character sequences.

Table 9-1. Special tr character sequences

Sequence	Meaning
\\	backslash (\)
\n	new line
char1-char2	All characters from *char1* to *char2* (e.g. “a-z”)
[:alnum:]	All alphanumeric characters
[:alpha:]	All letters in the alphabet
[:punct:]	Punctuation characters

Characters can be deleted from one text with the `-d`, and a set that specifies the characters that should be deleted. Let’s start with an easy example: suppose that you want to remove all the new line characters from a text stored in `text`, and that you would like to redirect the output to a file named `text-continuous`. Obviously, we need a set with only one character, namely the new line character, which is specified with “\n”. This can be accomplished with the following command:

```
$ cat text | tr -d "\n" > text-continuous
```

It happens quite often that you want to delete everything, but the characters that are specified in a set. You can do this by using the `-c`, which automatically complements the specified set. For example, if you would like to remove every character from a text, except for alphabetical characters, new lines, and spaces, you can use the `-c`, and the following set: *“[:alpha:]\n ”*. Combined in a command it would look like this:

```
$ cat text | tr -c -d "[:alpha:]\n "
```

Using **tr** for translating characters does not require any extra parameters, but two sets. In the two sets the first character of the first set is replaced with the first character of the second set, etc. Suppose that we use these two sets: “abc” and “def”. With these sets the following translations occur: “a -> d”, “b -> e”, and “c -> f”. If the first set is longer than the second set, then the second set is expanded by repeating the last character of the second set. If the second set is longer than the first set, extra characters in the second set are ignored.

Translations can be useful for many purposes. For example, it can be used to make a wordlist from a text. This can be done by replacing all spaces with a newline:

```
$ cat essay.txt | tr " " "\n" | less
```

As you can imagine the output might still contain non-alphabetic characters. We can combine the command above with the delete functionality of **tr** to make a wordlist without unwanted characters:

```
$ cat essay.txt | tr " " "\n" | tr -c -d "[:alpha:]\n" > wordlist
```

sort

The **command** is used to sort lines in a file. To sort a text alphabetically, you can just pipe or redirect the data to **sort**. **sort** also accepts file names as its parameters. When multiple files are specified, the files will be concatenated before sorting the lines. Suppose that you have a word list that is unordered, which is stored in the file `wordlist_unsorted`. You can sort the contents of the file, and output it to `wordlist_sorted` with:

```
$ sort wordlist_unsorted > wordlist_sorted
```

The **sort** accepts many different parameters, which are all described in the **sort**(1) manual page. An often-used parameter we will shortly discuss is *-u*. With this parameter only unique words will be sent to *stdout* (in other words: double occurences will be ignored). By combining **sort -u** and the **tr**, discussed in the section called *tr*, you can make a sorted wordlist of a text:

```
$ cat essay.txt | tr " " "\n" | tr -c -d "[:alpha:]\n" | sort -u > wordlist_sorted
```

If the command listed in this example is combined with **wc**, one could count the size of the used vocabulary in the text. In the sorted word list each line represents an unique word from the original text. So, we can count the total number of words that were used, by counting the total number of lines in the sorted list:

```
$ cat essay.txt | tr " " "\n" | tr -c -d "[:alpha:]\n" | sort -u | wc -l
```

uniq

The **uniq** can be compared to the *-u* parameter of the **sort**; it removes all but one entry of successive identical lines (in a sorted list). The main difference is that it provides some extra parameters that can be used to manipulate the output. The default behavior works like **sort -u**, and reduces duplicate entries:

```
$ sort wordlist_unsorted | uniq > wordlist_sorted
```

To make a list of how often a line occurs in a text, one could count how many identical lines the sorted list contains of every line. Uniq can add the number of identical lines with the *-c* parameter. To make a list of how many times each word occurs in a text, you can combine **tr**, **sort**, and **uniq**:

```
$ cat essay.txt | tr " " "\n" | tr -c -d "[:alpha:]\n" | sort | uniq -c > wordlist_sorte
```

Chapter 10. Process management

Introduction

UNIX-like operating systems work with processes. A process is an unit the operating system schedules for CPU time and the memory manager manages memory for. Basically a process consists of program code (named text), data (used by a program) and a stack. The stack is used by the program to store variables. Programs are at least one process. A program/process can ask the system to create a new copy of itself, which is called a fork. For example, a web server could fork itself to let the new process handle a request.

A process can be parted in threads. The difference between forking a process and creating a thread is that different threads share the address space of the process. A forked process is a separate process with its own address space. Forking is more expensive in terms of memory requirement and CPU time.

A user can control a process by sending signals to the process. For example, the SIGTERM command is used to terminate a process, and the SIGHUP signal to restart a process.

Process basics

This section describes some basic commands that are used for process management.

ps

The **ps**(1) command is used to report which processes are currently active. By running **ps** without any parameters you can see which processes are active in the current user session. Let's look at an example:

```
$ ps
  PID TTY          TIME CMD
 1191 pts/2    00:00:00 bash
 1216 pts/2    00:00:00 ps
```

In this example the **bash** and **ps** commands are running. As you can see each process has a process ID (PID). You will need the process number if you want to send a signal to a process, for example a kill signal. The **ps** has many parameters to modify the output. For example, the `x` shows all processes without a controlling tty:

```
$ ps x
  PID TTY      STAT   TIME COMMAND
 1044 tty1     S      0:00 -bash
 1089 tty1     S      0:00 /bin/sh /usr/X11R6/bin/startx
 1100 tty1     S      0:00 xinit /home/daniel/.xinitrc --
 1108 tty1     S      0:00 /usr/bin/wmaker
 1113 tty1     S      0:00 sylpheed
 1114 tty1     S      0:00 /bin/sh /opt/firefox/run-mozilla.sh /opt/firefox/fire
 1120 tty1     S      0:52 /opt/firefox/firefox-bin
 1125 tty1     S      0:00 /usr/libexec/gconfd-2 20
 1146 tty1     S      0:00 xchat
 1161 tty1     S      0:00 xterm -sb
 1163 pts/0    S      0:00 bash
 1170 pts/0    S      0:00 vi proc.xml
 1189 tty1     S      0:00 xterm -sb
 1191 pts/2    S      0:00 bash
 1275 pts/2    R      0:00 ps x
```

Have a look at the **ps**(1) manual page for a summary of available parameters.

kill

The **kill**(1) sends a signal to a process. If no signal is specified the TERM signal is send, which asks a process to exit gracefully. Let's have a look at the normal mode of execution:

```
$ ps ax | grep mc
 1045 tty4     S      0:00 /usr/bin/mc -P /tmp/mc-daniel/mc.pwd.756
$ kill 1045
$ ps ax | grep mc
$
```

As you can see the **ps** is used to look for the **mc** process. There is one occurrence of **mc** running with PID 1045. This process is killed, and the second **ps** command shows that the process is indeed terminated.

As we said earlier the **kill** command can also be used to send other signals. The **kill -l** displays a list of signals that can be sent:

```
 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     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
```

The SIGKILL signal is often used to kill processes that refuse to terminate with the default SIGTERM signal. The signal can be specified by using the number as a parameter, for example, the following command would send a SIGKILL signal to PID 1045:

```
$ kill -9 1045
```

It is also possible to specify the signal without the "SIG" letters as a parameter. In the following example the SIGHUP signal is sent to the **inetd** to restart it:

```
# ps ax | grep inetd
  727 ?        S      0:00 /usr/sbin/inetd
# kill -HUP 727
```

Advanced process management

Background processes

Normally a process takes over the screen and keyboard after it is started. It is also possible to start processes as a background process, this means that the shell starts the process, but keeps control over the terminal. In most shells a process can be started as a background process by placing an ampersand (&) after the command. For example:

```
$ rm -rf ~/bunch/of/files &
```

A process that runs in the background can be brought to the foreground using the **fg %<job ID>** command. You can see which jobs are running, with their job numbers, using the **jobs** command. For example:

```
$ sleep 1000 &
[1] 947
$ jobs
[1]+  Running                 sleep 1000 &
$ fg %1
sleep 1000
```

The first command, **sleep 1000 &**, starts sleep in the background. **sleep** is a command does nothing but waiting the number of seconds that are specified as a parameter. The output of the **jobs** command shows that **sleep** is indeed running, with Job ID *1*. Finally we move **sleep** to the foreground. As you can see, the shell will print which command is moved to the foreground.

Stopping processes

A process that is running can be stopped by pressing the <Control> and <z> keys simultaneously. Stopped processes can be moved to the foreground with the **fg** command. Running **fg** without any parameters moves the last process that was stopped to the foreground. Other processes can be moved to the foreground by specifying the job ID as a parameter to **fg**.

A stopped process can also be told to continue as a background process, by executing **bg <job ID>**. Executing **fg** without any parameter will move the last stopped process to the background.

Altering priorities

The Linux kernel allows a user to change the priority of a program. For example, suppose that you want to run a process that requires a lot of CPU time, but you do not want to hinder other users. In this case you can start the process with a low priority, the process will only get CPU time when there are not many other processes demanding CPU time. Or you can give processes that are important a higher priority.

GNU/Linux provides two commands to alter the priority of a process. The **nice**(1) command can be used to specify the priority when you are launching a process. With the **renice**(1) command you can alter the priority of a process that is already running. The priority is a numerical value from -20 (highest priority) to 19 (lowest priority). Let's start with an example of **nice** in action:

```
$ nice -n 19 ./setiathome
```

As you can see the *-n* parameter is used to specify the priority value. In this case the **./setiathome** will have a very low priority. Be aware that only the superuser can use negative priority values. Thus, a normal user cannot give a process a higher priority, as illustrated by this example:

```
$ nice -n -1 nice
nice: cannot set priority: Permission denied
```

But it will work as the *root* user:

```
$ su -c "nice -n -1 nice"
Password:
-1
```

The **renice** command has a somewhat different syntax. The easiest way to use it is to specify the new priority and the process ID as parameters to the **renice** command, as is shown in the following example:

```
$ renice +5 5811
5811: old priority 0, new priority 5
```

As with **nice** a non-*root* user cannot set negative priority values:

```
$ renice -5 5811
renice: 5811: setpriority: Permission denied
```

Note: A normal user can not increase the priority of the process beyond the default priority of 0. Such facilities could be misused by careless users. After all, the command **nice** is derived from being nice to the other users on the system ;^).

III. Editing and typesetting

Chapter 11. LaTeX

Introduction

LaTeX is a typesetting system that can be used to produce high-quality articles, books, letters and other publications. LaTeX is based on TeX, a lower-level typesetting language that was designed by Donald E. Knuth. LaTeX does not work like a WYSIWYG (what you see is what you get) word processor, the kind of document preparation system most people are accustomed to. With LaTeX you do not have to care about formatting the document, only about writing the document.

LaTeX files are plain-text files that contain LaTeX macros. LaTeX formats the document based on the macros that are used. In the beginning using LaTeX may be a bit awkward to a new user. But after a while you will discover that using LaTeX has some distinct advantages. To name a few:

- LaTeX-formatted documents look very professional.
- You do not have to care about the layout of your documents. You just add structure to your documents, and LaTeX takes care of the formatting.
- LaTeX files are plain text, and can easily be changed using standard UNIX tools, such as **vi**, **sed** or **awk**
- LaTeX provides very good support for typesetting things like mathematical formula, references and Postscript images.

LaTeX is very extensive, so this chapter will only touch the surface of LaTeX. But it should be enough to get you started to be able to make simple documents.

Preparing basic LaTeX documents

Minimal document structure

Each LaTeX document has some basic minimal structure that describes the document. Let's look at an example:

```
\documentclass[10pt,a4paper]{article} ❶

\title{The history of symmetric ciphers} ❷
\author{John Doe} ❸

\begin{document} ❹

This is a basic document. ❺

\end{document} ❻
```

You can already see the basic syntactical structure of LaTeX commands. A command is started with a backslash, followed by the name of the command. Each macro has a mandatory argument that is placed in accolades, and an optional argument that is placed in square brackets.

❶ The first command in every document is the *documentclass*. This command specifies what kind of document LaTeX is dealing with. The type of document is specified as a mandatory parameter. You can also specify some optional parameters, such as the font size and the paper size. In this case the font size is changed from the default 12pt to 10pt, and A4 is used as the paper size. The document classes that are available in LaTeX are shown in Table 11-1.

❷ After the documentclass you can add some meta-information to the document, like the title of the document. In this case the title is *The history of symmetric ciphers*.

❸ The author command specifies the author of the book.

❹ The *\begin* command marks the beginning of an environment. There are many different environments, but they all implicate certain typesetting conventions for the text that is in an environment. In this case we start an *document* environment. This is a very basic environment, LaTeX interprets everything in this environment as the body of the text.

❺ The content of the document can be placed within the *document*, in this case a friendly warning about the nature of the document.

❻ All environments eventually have to be closed. The *document* environment is the last environment that is closed, because it denotes the end of the body of the text.

Table 11-1. LaTeX document classes

Class
article
book
letter
report

Generating printable formats

Once you have a LaTeX file, you can use the **latex** command to generate a DVI (Device Independent format) file:

```
$ latex crypto.tex
This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
(./crypto.tex
LaTeX2e <2003/12/01>
Babel <v3.8d> and hyphenation patterns for american, french, german, ngerman, b
ahasa, basque, bulgarian, catalan, croatian, czech, danish, dutch, esperanto, e
stonian, finnish, greek, icelandic, irish, italian, latin, magyar, norsk, polis
h, portuges, romanian, russian, serbian, slovak, slovene, spanish, swedish, tur
kish, ukrainian, nohyphenation, loaded.
(/usr/share/texmf/tex/latex/base/article.cls
Document Class: article 2004/02/16 v1.4f Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/size10.clo)) (./crypto.aux) [1] (./crypto.aux)
 )
 Output written on crypto.dvi (1 page, 248 bytes).
 Transcript written on crypto.log.
```

As the LaTeX command reports a DVI file is created after running the **latex** command. You can view this file with an X viewer for DVI files, **xdvi**:

```
$ xdvi crypto.dvi
```

This file is not directly printable (although DVI files can be printed with **xdvi**). An ideal format for printable files is Postscript. You can generate a Postscript file from the DVI file with one simple command:

```
$ dvips -o crypto.ps crypto.dvi
```

The `-o` specifies the output (file) for the Postscript document. If this parameter is not specified, the output will be piped through **lpr**, which will schedule the document to be printed.

PDF (Portable Document Format) is another popular format for electronic documents. PDF can easily be generated from Postscript:

```
$ ps2pdf crypto.ps
```

The resulting output file will be the same as the input file, with the `.ps` extension replaced with **.pdf**.

Sections

Now that you know how to create a basic LaTeX document, it is a good time to add some more structure. Structure is added using sections, subsections, and subsubsections. These structural elements are made with respectively the *\section*, *\subsection* and *\subsubsection* commands. The mandatory parameter for a section is the title for the section. Normal sections, subsections and subsubsections are automatically numbered, and added to the table of contents. By adding a star after a section command, for example *\section*{title}* section numbering is suppressed, and the section is not added to the table of contents. The following example demonstrates how you can use sections:

```
\documentclass[10pt,a4paper]{article}

\title{The history of symmetric ciphers}
\author{John Doe}

\begin{document}

\section{Pre-war ciphers}

To be done.

\section{Modern ciphers}

\subsection*{Rijndael}

Rijndael is a modern block cipher that was designed by Joan Daemen and
Vincent Rijmen.

In the year 2000 the US National Institute of Standards and Technologies
selected Rijndael as the winner in the contest for becoming the Advanced
Encryption Standard, the successor of DES.

\end{document}
```

The example above is pretty straightforward, but this is a good time to look at how LaTeX treats end of line characters and empty lines. Empty lines are ignored by LaTeX, making the text a continuous flow. An empty line starts a new paragraph. All paragraphs but the first paragraph are stared with a extra space left of the first word.

Font styles

Usually you may want to work with different font styles too. LaTeX has some commands that can be used to change the appearance of the current font. The most commonly used font commands are \emph for emphasized text, and \textbf. Have a look at Table 11-2 for a more extensive list of font styles. Emphasis and bold text are demonstrated in this example paragraph:

```
Working with font styles is easy. \emp{This text is emphasized} and
\textbf{this text is bold}.
```

Table 11-2. LaTeX font styles

Command	Description
\emph	Add emphasis to a font.
\textbf	Print the text in bold.
\textit	Use an italic font.
\textsl	Use a font that is slanted.
\textsc	Use CAPS.
\texttt	Use a typewriter font.
\textsf	Use a sans-serif font.

IV. Electronic mail

Chapter 12. Reading and writing e-mail with mutt

Introduction

Mutt is a mail user agent (MUA) that can be used for reading and writing e-mail. Mutt is a text-mode installation, meaning that it can be used on the console, over SSH and in an X terminal. Due to its menu interface, it is very easy to read large amounts of e-mail in a short time, and mutt can be configured to use your favorite text editor.

This chapter will discuss how you can customize mutt for your needs, how to use it, and how PGP/GnuPG support is used.

Usage

Mutt is pretty simple to use, though it may take some time to get used to the keys that are used to navigate, read and write e-mails. The next few sections describe some of the more important keys. Mutt provides a more thorough overview of available keys after pressing the <h> key.

Browsing the list of e-mails

After invoking the **mutt** command, an overview of all e-mails will show up. You can browse through the list of e-mails with the up and down arrow keys, or the <k> and >j> keys.

Reading e-mails

To read an e-mail, use the <Enter> key, after selecting an e-mail in the overview of e-mails. When reading an e-mail you can use the <Page Up> and <Page Down> to browse through an e-mail. You can still use the navigational keys used to browse the list of e-mail to browse to other e-mails.

If an e-mail has any attachments, you can see them by pressing the <v> key. You can view individual attachments by selecting them and pressing the <Enter> key. To save an attachment to a file, press the <s> key.

Sending e-mails

You can compose a new e-mail with the <c> key, or reply to a selected e-mail with the <r> key. Mutt will ask you to specify the recipient (*To:*), and a subject (*Subject:*). After entering this information an editor is launched (**vi** is used by default), which you can use to compose the e-mail. After saving the e-mail, and quitting the editor, mutt will give you the opportunity to make any changes to the e-mail. If you decide that you want to alter the e-mail, you can restart the editor with the <e> key. You can change the recipient or the subject with respectively <t> or <s>. Finally, you can send the e-mail by pressing <y>. If you would like to cancel the e-mail, press <q>. Mutt will ask you whether you want to postpone the e-mail. If you do so, you will be given the opportunity to re-do the e-mail the next time you compose a message.

Basic setup

There are a few mutt settings you often want to configure. This section describes these settings. User-specific mutt customizations can be made in the `.muttrc` in the user's home directory. You can change global mutt settings in `/etc/mutt/Muttrc`.

Customized headers

Each e-mail has headers with various information. For example, the header contains information about the path an e-mail has traversed after it has been sent. The sender (*From:*) and recipient (*To:*) e-mail addresses are also stored in the headers, as well as the subject (*Subject:*) of an e-mail.

> **Note:** In reality the *To:* header is not used to determine the destination of an e-mail during the deliverance process of the e-mail. MTAs use the *envelope address* to determine the destination of the e-mail. Though, most MUAs use the *To:* address that the user fills in as the envelope address.

You can add your own headers to an e-mail with the *my_hdr* configuration option. This option has the following syntax: *my_hdr <header name>: <header contents>*. For example, you can add information about what OS you are running by adding the following line to your mutt configuration:

```
my_hdr X-Operating-System: Slackware Linux 10.1
```

You can also override some of the headers that are normally used, such as the sender address that is specified in the *From:* header:

```
my_hdr From: John Doe <john.doe@example.org>
```

The sendmail binary

By default mutt uses the sendmail MTA to deliver e-mails that were sent. You can use another command to send e-mail by altering the *sendmail* configuration variable. The sendmail replacement must handle the same parameter syntax as sendmail. For example, if you have installed MSMTP to deliver e-mails, you can configure mutt to use it by adding the following line to your mutt configuration:

```
set sendmail="/usr/bin/msmtp"
```

When you have completely replaced sendmail with another MTA, for instance Postfix, it is usually not needed to set this parameter, because most MTAs provide an alternative sendmail binary file.

Signing/encrypting e-mails

Introduction

Mutt provides excellent support for signing or encrypting e-mails with GnuPG. One might wonder why he or she should use one of these techniques. While most people do not feel the need to encrypt most of their e-mails, it generally is a good idea to sign your e-mails. There are, for example, a lot of viruses these days that use other people's e-mail addresses in the From: field of viruses. If the people who you are communicating with know that you sign your e-mails, they will not open fake e-mail from viruses. Besides that it looks much more professional if people can check your identity, especially in business transactions. For example, who would you rather trust, vampire_boy93853@hotmail.com, or someone using a professional e-mail address with digitally signed e-mails?

This section describes how you can use GnuPG with mutt, for more information about GnuPG read the section called *Encrypting and signing files* in Chapter 8.

Configuration

An example configuration for using GnuPG in mutt can be found in `/usr/share/doc/mutt/samples/gpg.rc`. In general the contents of this file to your mutt configuration will suffice. From the shell you can add the contents of this file to you `.muttrc` with the following command:

```
$ cat /usr/share/doc/mutt/samples/gpg.rc >> ~/.muttrc
```

There are some handy parameters that you can additionally set. For example, if you always want to sign e-mails, add the following line to your mutt configuration:

```
set crypt_autosign = yes
```

Another handy option is *crypt_replyencrypt*, which will automatically encrypt replies to messages that were encrypted. To enable this, add the following line to your mutt configuration:

```
set crypt_replyencrypt = yes
```

Usage

If you have set some of the automatical options, like *crypt_autosign* GnuPG usage of mutt is mostly automatic. If not, you can press the <p> key during the final step of sending an e-mail. In the bottom of the screen various GnuPG/PGP options will appear, which you can access via the letters that are enclosed in parentheses. For example, <s> signs e-mails, and <e> encrypts an e-mail. You can always clear any GnuPG option you set by pressing <p> and then <c>.

Chapter 13. Sendmail

Introduction

Sendmail is the default Mail Transfer Agent (MTA) that Slackware Linux uses. sendmail was originally written by Eric Allman, who still maintains sendmail. The primary role of the sendmail MTA is delivering messages, either locally or remotely. Delivery is usually done through the SMTP protocol. The means that sendmail can accept e-mail from remote sites through the SMTP port, and that sendmail delivers site destined for remote sites to other SMTP servers.

Installation

Sendmail is available as the *sendmail* package in the “n” disk set. If you want to generate your own sendmail configuration files, the *sendmail-cf* package is also required. For information about how to install packages on Slackware Linux, refer to Chapter 18.

You can let Slackware Linux start sendmail during each boot by making the `/etc/rc.d/rc.sendmail` executable. You can do this by executing:

```
# chmod a+x /etc/rc.d/rc.sendmail
```

You can also start, stop and restart sendmail by using *start*, *stop*, and *restart* as a parameter to the sendmail initialization script. For example, you can restart sendmail in the following way:

```
# /etc/rc.d/rc.sendmail restart
```

Configuration

The most central sendmail configuration file is `/etc/mail/sendmail.cf`; this is where the behavior of sendmail is configured, and where other files are included. The syntax of `/etc/mail/sendmail.cf` is somewhat obscure, because this file is compiled from a much simpler `.mc` files that uses M4 macros that are defined for sendmail.

Some definitions can easily be changed in the `/etc/mail/sendmail.cf` file, but for other changes it is better to create your own `.mc` file. Examples in this chapter will be focused on creating a customized mc file.

Working with mc files

In this section we will look how you can start off with an initial mc file, and how to compile your own .cf file to a cf file. There are many interesting example mc files available in `/usr/share/sendmail/cf/cf`. The most interesting examples are `sendmail-slackware.mc` (which is used for generating the default Slackware Linux `sendmail.cf`), and `sendmail-slackware-tls.mc` which adds TLS support to the standard Slackware Linux sendmail configuration. If you want to create your own sendmail configuration, it is a good idea to start with a copy of the standard Slackware Linux mc file. For instance, suppose that we would like to create a configuration file for the server named *straw*, we could execute:

```
# cd /usr/share/sendmail/cf/cf
# cp sendmail-slackware.mc sendmail-straw.mc
```

and start editing `sendmail-straw.mc`. After the configuration file is modified to our tastes, M4 can be used to compile a cf file:

```
# m4 sendmail-straw.mc > sendmail-straw.cf
```

If we want to use this new configuration file as the default configuration, we can copy it to `/etc/mail/sendmail.cf`:

```
# cp sendmail-straw.cf /etc/mail/sendmail.cf
```

Using a smarthost

If you would like to use another host to deliver e-mail to locations to which the sendmail server you are configuring can not deliver you can set up sendmail to use a so-called “smart host”. Sendmail will send the undeliverable e-mail to the smart host, which is in turn supposed to handle the e-mail. You do this by defining *SMART_HOST* in your mc file. For example, if you want to use *smtp2.example.org* as the smart host, you can add the following line:

```
define(‘SMART_HOST’,‘stmp2.example.org’)
```

Alternative host/domain names

By default sendmail will accept mail destined for localhost, and the current hostname of the system. You can simply add additional hosts or domains to accept e-mail for. The first step is to make sure that the following line is added to your mc configuration:

```
FEATURE(‘use_cw_file’)dnl
```

When this option is enabled you can add host names and domain names to accept mail for to `/etc/mail/local-host-names`. This file is a newline separated database of names. For example, the file could look like this:

```
example.org
mail.example.org
www.example.org
```

Virtual user table

Often you may want to map e-mail addresses to user names. This is needed when the user name differs from the part before the "@" part of an e-mail address. To enable this functionality, make sure the following line is added to your mc file:

```
FEATURE(‘virtusertable’,‘hash -o /etc/mail/virtusertable.db’)dnl
```

The mappings will now be read from `/etc/mail/virtusertable.db`. This is a binary database file that should not directly edit. You can edit `/etc/mail/virtusertable` instead, and generate `/etc/mail/virtusertable.db` from that file.

The `/etc/mail/virtusertable` file is a simple plain text file. That has a mapping on each line, with an e-mail address and a user name separated by a tab. For example:

```
john.doe@example.org	john
john.doe@mail.example.org		john
```

In this example both e-mail for *john.doe@example.org* and *john.doe@mail.example.org* will be delivered to the *john* account. It is also possible to deliver some e-mail destined for a domain that is hosted on the server to another e-mail address, by specifying the e-mail address to deliver the e-mail to in the second in the second column. For example:

```
john.doe@example.org	john.doe@example.com
```

After making the necessary changes to the `virtusertable` file you can generate the database with the following command:

```
# makemap hash /etc/mail/virtusertable < /etc/mail/virtusertable
```

Chapter 14. Spamassassin

Introduction

Spam E-Mail is one of the major problems of the current Internet. Fortunately there are some tools which can help you reducing the amount of spam you get. One of them is using blacklists. A blacklist is a list of known open e-mail relays. With a blacklist you can block e-mails which are sent using one of these open relays. Unfortunately there are a few problems with this method, first of all the database of open relays can never be complete. Besides that some spam is sent using normal ISP-specific SMTP servers.

Another method is scanning the incoming e-mails for common characteristics of spam. This method is comparable with a virus scanner; a filter program knows a large number of spam characteristics and gives each characteristic found in a mail a point. If a defined number of points is exceeded the mail is marked as spam. Spamassassin is that kind of filter.

Installing spamassasin

SpamAssassin

SpamAssassin is written in Perl, and is available through CPAN (Comprehensive Perl Archive Network). You can install SpamAssassin through the CPAN shell. Execute the following command to enter the CPAN shell:

```
# perl -MCPAN -e shell
```

Once the shell is started you can install SpamAssassin:

```
# o conf prerequisites_policy ask
# install Mail::SpamAssassin
```

The first command will make the installation of dependencies interactive (in other words, the CPAN shell will ask you to confirm their installation). The second command asks the CPAN shell to install SpamAssassin.

During the installation you will be asked to fill in the e-mail address of the e-mail address or URL that should be used in the report that is sent when something is suspected to be spam. You will also be asked whether you would like to conduct some tests or not. You can answer “n” to these questions.

Starting spamd

It is not a bad idea to start a daemonized version of SpamAssassin which runs in the background. This will speed up mail filtering. The daemon can be started using the following command:

```
# spamd -c -d
```

It is a good idea tot add this to `/etc/rc.local`, so **spamd** will automatically be started during the initialization process.

Using spamassassin with procmail

Sendmail will use procmail automatically on the default Slackware Linux installation if procmail is installed. Procmail is a program which processes e-mails and allows you to apply filters. At first we are going to have a look at how to add spamassassin headers to a processed e-mail, after that we are going to look at a method to separate spam from normal e-mail.

Marking messages as spam

The first step is to mark messages as either spam or non-spam. Create a `/etc/procmailrc` file, if you do not already have one. This is the system-wide procmail configuration file, and applies to all incoming e-mails. Use the `~/.procmailrc` file if you want to enable spam marking for an individual account. Add the following lines to the configuration file:

```
:0 fw
* < 256000
| /usr/bin/spamc -f
```

The first line says we want to pipe all messages to an external command. The second line makes sure only messages smaller than 256000 bytes are filtered. Spam messages are usually not that large, and adding this line can decrease the system load. Finally, the third line specifies that the messages should be piped through **/usr/bin/spamc** with the *-f* parameter.

Moving spam mail to a separate mailbox

Procmail can also be used to move spam to a separate mailbox file. It is not a bad idea to configure this on a user basis, because some users might want to use the filters of their e-mail program to separate spam. In the following example we will move spam messages to the `~/mail/spam` mailbox file. To do this add the following lines to `~/.procmailrc`:

```
MAILDIR=$HOME/mail

:0:
* ^X-Spam-Status: Yes
spam
```

First of all MAILDIR is defined, this will create and use the mailboxes in `~/mail/`. In the next two lines we say that we want to filter out e-mails with the *X-Spam-Status: Yes* header, which is added by spamassassin if it believes an e-mail is spam. Finally the mailbox to which the e-mails should be moved is specified.

V. System administration

Chapter 15. User management

Introduction

GNU/Linux is a multi-user operating system. This means that multiple users can use the system, and they can use the system simultaneously. The GNU/Linux concepts for user management are quite simple. First of all, there are several user accounts on each system. Even on a single user system there are multiple user accounts, because GNU/Linux uses unique accounts for some tasks. Users can be members of groups. Groups are used for more fine grained permissions, for example, you could make a file readable by a certain group. There are a few reserved users and groups on each system. The most important of these is the *root* account. The *root* user is the system administrator. It is a good idea to avoid logging in as root, because this greatly enlarges security risks. You can just log in as a normal user, and perform system administration tasks using the **su** and **sudo** commands.

The available user accounts are specified in the `/etc/passwd`. You can have a look at this file to get an idea of which user account are mandatory. As you will probably notice, there are no passwords in this file. Passwords are kept in the separate `/etc/shadow` file, as an encrypted string. Information about groups is stored in `/etc/group`. It is generally speaking not a good idea to edit these files directly. There are some excellent tools that can help you with user and group administration. This chapter will describe some of these tools.

Adding and removing users

useradd

The **useradd** is used to add user accounts to the system. Running **useradd** with a user name as parameter will create the user on the system. For example:

```
# useradd bob
```

Creates the user account *bob*. Please be aware that this does not create a home directory for the user. Add the *-m* parameter to create a home directory. For example:

```
# useradd -m bob
```

This would add the user *bob* to the system, and create the `/home/bob` home directory for this user. Normally the user is made a member of the *users* group. Suppose that we would like to make *crew* the primary group for the user *bob*. This can be done using the *-g* parameter. For example:

```
# useradd -g crew -m bob
```

It is also possible to add this user to secondary groups during the creation of the account with the *-G*. Group names can be separated with a comma. The following command would create the user *bob*, which is a member of the *crew* group, and the *www-admins* and *ftp-admins* secondary groups:

```
# useradd -g crew -G www-admins,ftp-admins -m bob
```

By default the **useradd** only adds users, it does not set a password for the added user. Passwords can be set using the **passwd** command.

passwd

As you probably guessed the **passwd** command is used to set a password for a user. Running this command as a user without a parameter will change the password for this user. The password command will ask for the old password,once and twice for the new password:

```
$ passwd
Changing password for bob
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

The *root* user can set passwords for users by specifying the user name as a parameter. The **passwd** command will only ask for the new password. For example:

```
# passwd bob
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

adduser

The **adduser** command combines **useradd** and **passwd** in an interactive script. It will ask you to fill in information about the account to-be created. After that it will create an account based on the information you provided. The screen listing below shows a sample session.

```
# adduser

Login name for new user []: john

User ID ('UID') [ defaults to next available ]: <Enter>

Initial group [ users ]: <Enter>

Additional groups (comma separated) []: staff

Home directory [ /home/john ] <Enter>

Shell [ /bin/bash ] <Enter>

Expiry date (YYYY-MM-DD) []: <Enter>

New account will be created as follows:

---------------------------------------
Login name.......:  john
UID..............:  [ Next available ]
Initial group....:  users
```

```
Additional groups:  [ None ]
Home directory...:  /home/john
Shell............:  /bin/bash
Expiry date......:  [ Never ]

This is it... if you want to bail out, hit Control-C.  Otherwise, press
ENTER to go ahead and make the account.
<Enter>

Creating new account...


Changing the user information for john
Enter the new value, or press ENTER for the default
        Full Name []: John Doe
        Room Number []: <Enter>
        Work Phone []: <Enter>
        Home Phone []: <Enter>
        Other []: <Enter>
Changing password for john
Enter the new password (minimum of 5, maximum of 127 characters)
Please use a combination of upper and lower case letters and numbers.
New password: password
Re-enter new password: password


Account setup complete.
```

You can use the default values, or leave some fields empty, by tapping the <Enter> key.

userdel

Sometimes it is necessary to remove a user account from the system. GNU/Linux offers the **userdel** tool to do this. Just specify the username as a parameter to remove that user from the system. For example, the following command will remove the user account *bob* from the system:

```
# userdel bob
```

This will only remove the user account, not the user's home directory and mail spool. Just add the *-r* parameter to delete the user's home directory and mail spool too. For example:

```
# userdel -r bob
```

Avoiding root usage with su

It is a good idea to avoid logging in as *root*. There are many reasons for not doing this. Accidentally typing a wrong command could cause bad things to happen, and malicious programs can make a lot of damage when you are logged in as *root*. Still, there are many situations in which you need to have root access. For example, to do system administration, or to install new software. Fortunately the **su** can give you temporal root privileges.

Using **su** is very simple. Just executing **su** will ask you for the root password, and will start a shell with root privileges after the password is correctly entered:

```
$ whoami
bob
$ su
Password:
# whoami
root
# exit
exit
$ whoami
bob
```

In this example the user *bob* is logged on, the **whoami** output reflects this. The user executes su and enters the *root* password. **su** launches a shell with root privileges, this is confirmed by the **whoami** output. After exiting the *root* shell, control is returned to the original running shell running with the privileges of the user *bob*.

It is also possible to execute just one command as the *root* user with the `-c` parameter. The following example will run **lilo**:

```
$ su -c lilo
```

If you want to give parameters to the command you would like to run, use quotes (e.g. **su -c "ls -l /"**). Without quotes **su** cannot determine whether the parameters should be used by the specified command, or by **su** itself.

Restricting su access

You can refine access to **su** with suauth(5). It is a good security practice to only allow members of a special group to **su** to *root*. For instance, you can restrict root **su**-ing in a BSD fashion to members of the *wheel* group by adding the following line to `/etc/suauth`:

```
root:ALL EXCEPT GROUP wheel:DENY
```

Disk quota

Introduction

Disk quota is a mechanism that allows the system administrator to restrict the number of disk blocks and inodes that a particular user and group can use. Not all filesystems supported by Linux support quota, widely used filesystems that support quota are ext2, ext3 and XFS. Quota are turned on and managed on a per filesystem basis.

Enabling quota

Quota can be enabled per filesystem in `/etc/fstab`, by using the *`usrquota`* and *`grpquota`* filesystem options. For example, suppose that we have the following entry for the `/home` partition in `/etc/fstab`:

```
/dev/hda8        /home            xfs        defaults         1   2
```

We can now enable user quota by adding the *`usrquota`* filesystem option:

```
/dev/hda8        /home            xfs        defaults,usrquota 1   2
```

At this point the machine can be rebooted, to let the Slackware Linux initialization scripts enable quota. You can also enable quota without rebooting the machine, by remounting the partition, and running the **quotaon** command:

```
# mount -o remount /home
# quotaon -avug
```

Editing quota

User and group quotas can be edited with the “edquota” utility. This program allows you to edit quotas interactively with the **vi** editor. The most basic syntax of this command is **edquota username**. For example:

```
# edquota joe
```

This will launch the **vi** editor with the quota information for the user *joe*. It will look like this:

```
Disk quotas for user joe (uid 1143):
  Filesystem                   blocks       soft       hard     inodes     soft     hard
  /dev/hda5                      2136          0          0         64        0        0
```

In this example quotas are only turned on for one file system, namely the filesystem on `/dev/hda5`. As you can see there are multiple columns. The *blocks* column shows how many block the user uses on the file system, and the *inodes* column the number of inodes a user occupies. Besides that there are *soft* and *hard* columns after both *blocks* and *inodes*. These columns specify the soft and hard limits on blocks and inodes. A user can exceed the soft limit for a grace period, but the user can never exceed the hard limit. If the value of a limit is *0*, there is no limit.

> **Note:** The term “blocks” might be a bit confusing in this context. In the quota settings a block is 1KB, not the block size of the file system.

Let’s look at a simple example. Suppose that we would like to set the soft limit for the user *joe* to *250000*, and the hard limit to *300000*. We could change the quotas listed above to:

```
Disk quotas for user joe (uid 1143):
  Filesystem                   blocks       soft       hard     inodes     soft     hard
  /dev/hda5                         2136      250000      300000           64         0         0
```

The new quota settings for this user will be active after saving the file, and quitting **vi**.

Getting information about quota

It is often useful to get statistics about the current quota usage. The **repquota** command can be used to get information about what quotas are set for every user, and how much of each quota is used. You can see the quota settings for a specific partition by giving the name of the partition as a parameter. The *-a* parameter will show quota information for all partitions with quota enabled. Suppose that you would like to see quota information for `/dev/hda5`, you can use the following command:

```
repquota /dev/hda5
*** Report for user quotas on device /dev/hda5
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --       0       0       0              3     0     0
[..]
joe       --    2136  250000  300000             64     0     0
[..]
```

Chapter 16. Printer configuration

Introduction

GNU/Linux supports a large share of the available USB, parallel and network printers. Slackware Linux provides two printing systems, CUPS (Common UNIX Printing System) and LPRNG (LPR Next Generation). This chapter covers the CUPS system.

Independent of which printing system you are going to use, it is a good idea to install some printer filter collections. These can be found in the “ap” disk set. If you want to have support for most printers, make sure the following packages are installed:

```
a2ps
enscript
espgs
gimp-print
gnu-gs-fonts
hpijs
ifhp
```

Both printing systems have their own advantages and disadvantages. If you do not have much experience with configuring printers under GNU/Linux, it is a good idea to use CUPS, because CUPS provides a comfortable web interface which can be accessed through a web browser.

Preparations

To be able to use CUPS the “cups” package from the “a” disk set has to be installed. After the installation CUPS can be started automatically during each system boot by making `/etc/rc.d/rc.cups` executable. This can be done with the following command:

```
# chmod a+x /etc/rc.d/rc.cups
```

After restarting the system CUPS will also be restarted automatically. You can start CUPS on a running system by executing the following command:

```
# /etc/rc.d/rc.cups start
```

Configuration

CUPS can be configured via a web interface. The configuration interface can be accessed with a web browser at the following URL: http://localhost:631/. Some parts of the web interface require that you authenticate yourself. If an authentication window pops up you can enter “root” as the user name, and fill in the root account password.

A printer can be added to the CUPS configuration by clicking on “Administrate”, and clicking on the “Add Printer” button after that. The web interface will ask for three options:

- *Name* - the name of the printer. Use a simple name, for example “epson”.
- *Location* - the physical location of the printer. This setting is not crucial, but handy for larger organizations.
- *Description* - a description of the printer, for example “Epson Stylus Color C42UX”.

You can proceed by clicking the “Continue” button. On the next page you can configure how the printer is connected. If you have an USB printer which is turned on, the web interface will show the name of the printer next to the USB port that is used. After configuring the printer port you can select the printer brand and model. After that the printer configuration is finished, and the printer will be added to the CUPS configuration.

An overview of the configured printers can be found on the “Printers” page. On this page you can also do some printer operations. For example, “Print Test Page” can be used to check the printer configuration by printing a test page.

Access control

The CUPS printing system provides a web configuration interface, and remote printer access through the Internet Printing Protocol (IPP). The CUPS configuration files allow you to configure fine-grained access control to printers. By default access to printers is limited to *localhost* (*127.0.0.1*).

You can refine access control in the central CUPS daemon configuration file, `/etc/cups/cupsd.conf`, which has a syntax that is comparable to the Apache configuration file. Access control is configured through *Location* sections. For example, the default global (IPP root directory) section looks like this:

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
</Location>
```

As you can see deny statements are handled first, and then allow statements. In the default configuration access is denied from all hosts, except for *127.0.0.1*, *localhost*. Suppose that you would like to allow hosts from the local network, which uses the *192.168.1.0/24* address space, to use the printers on the system you are configuring CUPS on. In this case you could add the line that is bold:

```
<Location />
Order Deny,Allow
Deny From All
Allow From 127.0.0.1
Allow From 192.168.1.0/24
</Location>
```

You can refine other locations in the address space by adding additional location sections. Settings for sub-directories override global settings. For example, you could restrict access to the *epson* printer to the hosts with IP addresses *127.0.0.1* and *192.168.1.203* by adding the following section:

```
<Location /printers/epson>
Order Deny,Allow
Deny From All
```

```
Allow From 127.0.0.1
Allow From 192.168.1.203
</Location>
```

Ghostscript paper size

Ghostscript is a PostStript and Portable Document Format (PDF) interpreter. Both PostScript and PDF are languages that describe data that can be printed. Ghostscript is used to convert PostScript and PDF to raster formats that can be displayed on the screen or printed. Most UNIX programs output PostScript, the CUPS spooler uses GhostScript to convert this PostScript to rasterized format that a particular printer understands.

There are some Ghostscript configuration settings that may be useful to change in some situations. This section describes how you can change the default paper size that Ghostscript uses.

> **Note:** Some higher-end printers can directly interpret PostScript. Rasterization is not needed for these printers.

By default Ghostscript uses US letter paper as the default paper size. The paper size is configured in `/usr/share/ghostscript/x.xx/lib/gs_init.ps`, in which *x.xx* should be replaced by the Ghostscript version number. Not far after the beginning of the file there are two lines that are commented out with a percent (%) sign, that look like this:

```
% Optionally choose a default paper size other than U.S. letter.
% (a4) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

You can change the Ghostscript configuration to use A4 as the default paper size by removing the percent sign and space that are at the start of the second line, changing it to:

```
% Optionally choose a default paper size other than U.S. letter.
(a4) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

It is also possible to use another paper size than Letter or A4 by replacing *a4* in the example above with the paper size you want to use. For example, you could set the default paper size to US Legal with:

```
% Optionally choose a default paper size other than U.S. letter.
(legal) /PAPERSIZE where { pop pop } { /PAPERSIZE exch def } ifelse
```

It is also possible to set the paper size per invocation of Ghostscript by using the *`-sPAPERSIZE=size`* parameter of the **gs** command. For example, you could use add *-sPAPERSIZE=a4* parameter when you start **gs** to use A4 as the paper size for an invocation of Ghostscript.

An overview of supported paper sizes can be found in the `gs_statd.ps`, that can be found in the same directory as `gs_init.ps`.

Chapter 17. X11

X Configuration

The X11 configuration is stored in `/etc/X11/xorg.conf`. Many distributions provide special configuration tools for X, but Slackware Linux only provides the standard X11 tools (which are actually quite easy to use). In most cases X can be configured automatically, but sometimes it is necessary to edit `/etc/X11/xorg.conf` manually.

Automatical configuration

The X11 server provides an option to automatically generate a configuration file. X11 will load all available driver modules, and will try to detect the hardware, and generate a configuration file. Execute the following command to generate a `xorg.conf` configuration file:

```
$ X -configure
```

If X does not output any errors, the generated configuration can be copied to the `/etc/X11` directory. And X can be started to test the configuration:

```
$ cp /root/xorg.conf /etc/X11/
$ startx
```

Interactive configuration

X11 provides two tools for configuring X interactively, **xorgcfg** and **xorgconfig**. **xorgcfg** tries to detect the video card automatically, and starts a tool which can be used to tune the configuration. Sometimes **xorgcfg** switches to a video mode which is not supported by the monitor. In that case **xorgcfg** can also be used in text-mode, by starting it with **xorgcfg -textmode**.

xorgconfig differs from the tools described above, it does not detect hardware and will ask detailed questions about your hardware. If you only have little experience configuring X11 it is a good idea to avoid **xorgconfig**.

Window manager

The “look and feel” of X11 is managed by a so-called window manager. Slackware Linux provides the following widely user window managers:

- WindowMaker: A relatively light window manager, which is part of the GNUStep project.
- BlackBlox: Light window manager, BlackBox has no dependencies except the X11 libraries.
- KDE: A complete desktop environment, including browser, e-mail program and an office suite (KOffice).
- GNOME: Like KDE a complete desktop environment. It is worth noting that Dropline Systems (http://www.dropline.net/) provides a special GNOME environment for Slackware Linux.

If you are used to a desktop environment, using KDE or GNOME is a logical choice. But it is a good idea to try some of the lighter window managers. They are faster, and consumer less memory, besides that most KDE and GNOME applications are perfectly usable under other window managers.

On Slackware Linux the following command can be used to select a window manager:

```
$ xwmconfig
```

This configuration program shows the installed window managers, from which you can choose one. You can set the window manager globally by executing **xwmconfig** as root.

Chapter 18. Package Management

Pkgtools

Introduction

Slackware Linux does not use a complex package system, unlike many other Linux distributions. Package have the `.tgz` extension, and are usually ordinary tarballs which contain two extra files: an installation script and a package description file. Due to the simplicity of the packages the Slackware Linux package tools do not have the means to handle dependencies. But many Slackware Linux users prefer this approach, because dependencies often cause more problems than they solve.

Slackware Linux has a few tools to handle packages. The most important tools will be covered in this chapter. To learn to understand the tools we need to have a look at package naming. Let's have a look at an example, imagine that we have a package with the file name `bash-2.05b-i386-2.tgz`. In this case the name of the package is bash-2.05b-i386-2. In the package name information about the package is separated by the '-' character. A package name has the following meaning: *programname-version-architecture-packagerevision*

pkgtool

The **pkgtool** command provides a menu interface for some package operations. De most important menu items are *Remove* and *Setup*. The *Remove* option presents a list of installed packages. You can select which packages you want to remove with the space bar and confirm your choices with the return key. You can also deselect a package for removal with the space bar.

The *Setup* option provides access to a few tools which can help you with configuring your system, for example: **netconfig**, **pppconfig** and **xwmconfig**.

installpkg

The **installpkg** command is used to install packages. **installpkg** needs a package file as a parameter. For example, if you want to install the package `bash-2.05b-i386-2.tgz` execute:

```
# installpkg bash-2.05b-i386-2.tgz
```

upgradepkg

upgradepkg can be used to upgrade packages. In contrast to **installpkg** it only installs packages when there is an older version available on the system. The command syntax is comparable to **installpkg**. For example, if you want to upgrade packages using package in a directory execute:

```
# upgradepkg *.tgz
```

As said only those packages will be installed of which an other version is already installed on the system.

removepkg

The **removepkg** can be used to remove installed packages. For example, if you want to remove the “bash” package (it is not recommended to do that!), you can execute:

```
# removepkg bash
```

As you can see only the name of the package is specified in this example. You can also remove a package by specifying its full name:

```
# removepkg bash-2.05b-i386-2
```

Slackpkg

Introduction

Slackpkg is a package tool written by Roberto F. Batista and Evaldo Gardenali. It helps users to install and upgrade Slackware Linux packages using one of the Slackware Linux mirrors. Slackpkg is included in the `extra/` directory on the second CD of the Slackware Linux CD set.

Configuration

Slackpkg is configured through some files in `/etc/slackpkg`. The first thing you should do is configuring which mirror slackpkg should use. This can be done by editing the `/etc/slackpkg/mirrors`. This file already contains a list of mirrors, you can just uncomment a mirror close to you. For example:

```
ftp://ftp.nluug.nl/pub/os/Linux/distr/slackware/slackware-10.1/
```

This will use the Slackware Linux 10.1 tree on the ftp.nluug.nl mirror. Be sure to use a tree that matches your Slackware Linux version. If you would like to track slackware-current you would uncomment the following line instead (when you would like to use the NLUUG mirror):

```
ftp://ftp.nluug.nl/pub/os/Linux/distr/slackware/slackware-current/
```

Slackpkg will only accept one mirror. Commenting out more mirrors will not work.

Importing the Slackware Linux GPG key

By default slackpkg checks packages using the package signatures and the public Slackware Linux GPG key. Since this is a good idea from a security point of view, you probably do not want to change this behaviour. To be able to verify packages you have to import the *security@slackware.com* GPG key. If you have not used GPG before you have to create the GPG directory in the home directory of the *root* user:

```
# mkdir ~/.gnupg
```

The next step is to search for the public key of *security@slackware.com*. We will do this by querying the *pgp.mit.edu* server:

```
# gpg --keyserver pgp.mit.edu --search security@slackware.com
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
gpg: searching for "security@slackware.com" from HKP server pgp.mit.edu
Keys 1-2 of 2 for "security@slackware.com"
(1)     Slackware Linux Project <security@slackware.com>
          1024 bit DSA key 40102233, created 2003-02-25
(2)     Slackware Linux Project <security@slackware.com>
          1024 bit DSA key 40102233, created 2003-02-25
Enter number(s), N)ext, or Q)uit >
```

As you can see we have got two (identical) hits. Select the first one by entering “1”. GnuPG will import this key in the keyring of the *root* user:

```
Enter number(s), N)ext, or Q)uit > 1
gpg: key 40102233: duplicated user ID detected - merged
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 40102233: public key "Slackware Linux Project <security@slackware.com>" importe
gpg: Total number processed: 1
gpg:               imported: 1
```

Be sure to double check the key you received. The key ID and fingerprint of this particular key can be found on the Internet on many trustworthy sites. The key ID is, as mentioned above *40102233*. You can get the key fingerprint with the `--fingerprint` parameter:

```
# gpg --fingerprint security@slackware.com
pub  1024D/40102233 2003-02-26 Slackware Linux Project <security@slackware.com>
     Key fingerprint = EC56 49DA 401E 22AB FA67  36EF 6A44 63C0 4010 2233
sub  1024g/4E523569 2003-02-26 [expires: 2012-12-21]
```

Once you have imported and checked this key you can start to use slackpkg, and install packages securely.

Updating the package lists

Before upgrading and installing packages you have to let slackpkg download the package lists from the mirror you are using. It is a good idea to do this regularly to keep these lists up to date. The latest package lists can be fetched with:

```
$ slackpkg update
```

Upgrading packages

The *upgrade* parameter is used to upgrade installed packages. You have to add an extra parameter to actually tell **slackpkg** what you want to upgrade, this differs for a stable Slackware Linux version and slackware-current. Upgrades for stable Slackware Linux releases are in the `patches` directory of FTP mirrors. You can update a slackware-stable installation (e.g. Slackware Linux 10.1) with:

```
# slackpkg upgrade patches
```

In this case **slackpkg** will use the packages from the `patches` directory. In slackware-current updated packages are put in the normal `slackware` package sub-directories. So, we can pass that as an parameter to **slackpkg upgrade**:

```
# slackpkg upgrade slackware
```

You can also upgrade individual packages by specifying the name of the package to be upgraded, for example:

```
# slackpkg upgrade pine
```

Installing packages

The *install* is used to install packages:

```
# slackpkg install rexima
```

Be aware that neither slackpkg, nor the Slackware Linux package tools do dependency checking. If some program does not work due to missing libraries, you have to add them yourself with slackpkg.

Using rsync

Another popular method of keeping Slackware Linux up to date is by keeping a local mirror. The ideal way of doing this is via rsync. rsync is a program that can synchronize two trees of files. The advantage is that rsync only transfers the differences in files, making it very fast. After syncing with a mirror you can upgrade Slackware Linux with **upgradepkg**, or make a new installation CD. The following example synchronizes a local current tree with an up-to-date tree from on a mirror:

```
# rsync -av --delete \
--exclude=slackware/kde \
--exclude=slackware/kdei \
--exclude=slackware/gnome \
--exclude=bootdisks \
--exclude=extra \
--exclude=testing \
--exclude=pasture \
--exclude=rootdisks \
--exclude=source \
--exclude=zipslack \
rsync://fill-in-mirror/pub/slackware/slackware-current/ \
/usr/share/mirrors/slackware-current
```

The *-a* parameter implies a few other options that try to make a copy that is as exact as possible (in terms of preserving symlinks, permissions and owners). The *--delete* deletes files that are not available on the mirror anymore. It is good idea to use this parameter, because otherwise your tree may get bloated very quickly with older package versions. With the *--exclude* parameter you can specify which files or directories should be ignored.

After syncing the tree you can use **upgradepkg** to update your Slackware Linux installation. For example:

```
# upgradepkg /usr/share/mirrors/slackware-current/slackware/*/*.tgz
```

Tagfiles

Introduction

Tagfiles are a relatively unknown feature of Slackware Linux. A tagfile is a file that can be used to instruct **installpkg** what packages should be installed from a collection of packages. For instance, the Slackware Linux installer generates a tagfile during the *Expert* and *Menu* installation methods to store which packages should be installed during the installation process.

The nice aspect of tagfiles is that you can easily create tagfiles yourself. By writing your own tagfiles you can automate the package installation, which is ideal for larger client or server roll-outs (or smaller set-ups if it gives you more comfort than installing packages manually). The easiest way to create your own tagfiles is by starting out with the tagfiles that are part of the official Slackware Linux distribution. In the following sections we are going to look at how this is done.

Creating tagfiles

Tagfiles are simple plain-text files. Each line consists of a package name and a flag, these two elements are separated by a colon and a space. The flag specifies what should be done with a package. The fields are described in Table 18-1. Let’s look at a few lines from the tagfile in the “a” disk set:

```
aaa_base: ADD
aaa_elflibs: ADD
acpid: REC
apmd: REC
bash: ADD
bin: ADD
```

It should be noted that you can also add comments to tagfiles with the usual comment (#) character. As you can see in the snippet above there are different flags. The table listed below describes the four different flags.

Table 18-1. Tagfile fields

Flag	Meaning
ADD	A package marked by this flag will automatically be installed
SKP	A package marked by this flag will automatically be skipped
REC	Ask the user what to do, recommend installation of the package.
OPT	Ask the user what to do, the package will be described as optional.

As you can read from the table **installpkg** will only act automatically when either *ADD* or *SKP* is used.

It would be a bit tedious to write a tagfile for each Slackware Linux disk set. The official Slackware Linux distribution contains a tagfile in the directory for each disk set. You can use these tagfiles as a start. The short script listed below can be used to copy the tagfiles to the current directory, preserving the disk set structure.

```
#!/bin/sh

if [ ! $# -eq 1 ] ; then
  echo "Syntax: $0 [directory]"
  exit
fi

for i in $1/*/tagfile; do
  setdir=`echo $i | egrep -o '\w+/tagfile$' | xargs dirname`
  mkdir $setdir
  cp $i $setdir
done
```

After writing the script to a file you can execute it, and specify a `slackware/` directory that provides the disk sets. For example:

```
$ sh copy-tagfiles.sh /mnt/flux/slackware-current/slackware
```

After doing this the current directory will contain a directory structure like this, in which you can edit the individual tag files:

```
a/tagfile
ap/tagfile
d/tagfile
e/tagfile
[...]
```

Using tagfiles

On an installed system you can let **installpkg** use a tagfile with the *`-tagfile`* parameter. For example:

```
# installpkg -root /mnt-small -tagfile a/tagfile /mnt/flux/slackware-current/slackware/a
```

Of course, tagfiles would be useless if they cannot be used during the installation of Slackware Linux. This is certainly possible: after selecting which disk sets you want to install you can choose in what way you want to select packages (the dialog is named *SELECT PROMPTING MODE*. Select *tagpath* from this menu. You will then be asked to enter the path to the directory structure with the tagfiles. The usual way to provide tagfiles to the Slackware Linux installation is to put them on a floppy or another medium, and mounting this before or during the installation. E.g. you can switch to the second console with by pressing the <ALT> and <F2> keys, and create a mount point and mount the disk with the tagfiles:

```
# mkdir /mnt-tagfiles
# mount /dev/fd0 /mnt/mnt-tagfiles
```

After mounting the disk you can return to the virtual console on which you run **setup**, by pressing <ALT> and <F1>.

Chapter 19. Building a kernel

Introduction

The Linux kernel is shortly discussed in the section called *What is Linux?* in Chapter 2. One of the advantages of Linux is that the full sources are available (as most of the Slackware Linux system). This means that you can recompile the kernel. There are many situations in which recompiling the kernel is useful. For example:

- *Making the kernel leaner*: One can disable certain functionality of the kernel, to decrease its size. This is especially useful in environments where memory is scarce.
- *Optimizing the kernel*: it is possible to optimize the kernel. For instance, by compiling it for a specific processor.
- *Hardware support*: Support for some hardware is not enabled by default in the Linux kernel provided by Slackware Linux. A common example is support for SMP systems.
- *Using custom patches*: There are many unofficial patches for the Linux kernel. Generally speaking it is a good idea to avoid unofficial patches. But some third party software, like Win4Lin (http://www.netraverse.com), require that you install an additional kernel patch.

This chapter focuses on the default kernel series used in Slackware Linux 10.1, Linux 2.4, though most of these instructions also apply to Linux 2.6. Compiling a kernel is not really difficult, just keep around a backup kernel that you can use when something goes wrong. Kernel compilation involves these steps:

- Configuring the kernel.
- Making dependencies.
- Building the kernel.
- Building modules.
- Installing the kernel and modules.
- Updating the LILO configuration.

In this chapter, we suppose that the kernel sources are available in `/usr/src/linux`. If you have installed the kernel sources from the “k” disk set, the kernel sources are available in `/usr/src/linux-kernelversion`, and `/usr/src/linux` is a symbolic link to the real kernel source directory. So, if you use the standards Slackware Linux kernel package you are set to go.

Configuration

As laid out above, the first step is to configure the kernel source. To ease the configuration of the kernel, it is a good idea to copy the default Slackware Linux kernel configuration to the kernel sources. The Slackware Linux kernel configuration files are stored on the distribution medium as `kernels/<Kernel>/config`. Suppose that you use the `bare.i` kernel (which is the default kernel), and that you have a Slackware Linux CD-ROM mounted at `/mnt/cdrom`, you can copy the Slackware Linux kernel configuration with:

```
# cp /mnt/cdrom/kernels/bare.i/config /usr/src/linux/.config
```

At this point you can start to configure the kernel. There are three configuration front-ends to the kernel configuration. The first one is *`config`*, which just asks you what you want to do for each kernel option. This takes a lot of time. So, normally this is not a good way to configure the kernel. A more user friendly approach is the *`menuconfig`* front-end, which uses a menuing system that you can use to configure the kernel. There is also an X front-end, named *`xconfig`*. You can start a configuration front-end by going to the kernel source directory, and executing **make <front-end>**. For example, to configure the kernel with the menu front-end you can use the following commands:

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

In the kernel configuration there are basically three options for each choice: “n” disables functionality, “y” enables functionality, and “m” compiles the functionality as a module. The default Slackware Linux kernel configuration is a very good configuration, it compiles only the bare functionality needed to boot the system in the kernel, the rest is compiled as a module. Whatever choices you make, you need to make sure that both the driver for your IDE/SCSI chip set is available in the kernel and the filesystem driver. If they are not, the kernel is not able to mount the root filesystem, and no further modules can be loaded.

Compilation

The first step of the kernel compilation is to let the kernel build infrastructure check the dependencies. This can be done with **make depend**:

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

If **make depend** quits because there are errors, you have to recheck the kernel configuration. The output of this command will usually give some clues where things went wrong. If everything went fine, you can start compiling the kernel with:

```
# make bzImage
```

This will compile the kernel and make a compressed kernel image named `bzImage` in `/usr/src/linux/arch/i386/boot`. After compiling the kernel you have to compile the modules separately:

```
# make modules
```

When the module compilation is done you are ready to install the kernel and the kernel modules.

Installation

Installing the kernel

The next step is to install the kernel and the kernel modules. We will start with installing the kernel modules, because this can be done with one command within the kernel source tree:

```
# make modules_install
```

This will install the modules in `/lib/modules/<kernelversion>`. If you are replacing a kernel with the same version number, it is a good idea to remove the old modules before installing the new ones. E.g.:

```
# rm -rf /lib/modules/2.4.26
```

You can "install" the kernel by copying it to the `/boot` directory. You can give it any name you want, but it is a good idea to use some naming convention. E.g. `vmlinuz-somename-version`. For instance, if we would name it `vmlinuz-custom-2.4.28`, we can copy it from within the kernel source tree with:

```
# cp arch/i386/boot/bzImage /boot/vmlinuz-custom-2.4.28
```

At this point you are almost finished. The last step is to add the new kernel to the Linux boot loader.

Configuring LILO

LILO (Linux Loader) is the default boot loader that Slackware Linux uses. The configuration of LILO works in two steps; the first step is to alter the LILO configuration in `/etc/lilo.conf`. The second step is to run the **lilo**, which will write the LILO configuration to the boot loader. The LILO configuration already has an entry for the current kernel you are running. It is a good idea to keep this entry, as a fall-back option if your new kernel does not work. If you scroll down to the bottom of `/etc/lilo.conf` you will see this entry, it looks comparable to this:

```
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/hda5
  label = Slack
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
# Linux bootable partition config ends
```

The easiest way to add the new kernel is to duplicate the existing entry, and then editing the first entry, changing the *`image`*, and *`label`* options. After changing the example above it would look like this:

```
# Linux bootable partition config begins
image = /boot/vmlinuz-custom-2.4.28
  root = /dev/hda5
  label = Slack
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
```

```
image = /boot/vmlinuz
  root = /dev/hda5
  label = SlackOld
  read-only # Non-UMSDOS filesystems should be mounted read-only for checking
# Linux bootable partition config ends
```

As you can see the *image* points to the new kernel in the first entry, and we changed the label of the second entry to "SlackOld". LILO will automatically boot the first image. You can now install this new LILO configuration with the **lilo** command:

```
# lilo
Added Slack *
Added SlackOld
```

The next time you boot both entries will be available, and the "Slack" entry will be booted by default.

> **Note:** If you want LILO to show a menu with the entries configured via `lilo.conf` on each boot, make sure that you add a line that says
>
> ```
> prompt
> ```
>
> to `lilo.conf`. Otherwise LILO will boot the default entry that is set with *default=<name>*, or the first entry when no default kernel is set. You can access the menu with entries at any time by holding the <Shift> key when LILO is started.

Chapter 20. System initialization

This chapter describes the initialization of Slackware Linux. Along the way various configuration files that are used to manipulate the initialization process are described.

The bootloader

Arguably the most important piece of an operating system is the kernel. The kernel manages hardware resources and software processes. The kernel is started by some tiny glue between the system BIOS (Basic Input/Output System) and the kernel, called the bootloader. The bootloader handles the complications that come with loading a specific (or less specific) kernel.

Most bootloader actually work in two stages. The first stage loader loads the second stage loader, that does the real work. The boot loader is divided in two stages on x86 machines, because the BIOS only loads one sector (the so-called boot sector) that is 512 bytes in size.

Slackware Linux uses the LILO (LInux LOader) boot loader. This bootloader has been in development since 1992, and is specifically written to load the Linux kernel. Lately LILO has been replaced by the GRUB (GRand Unified Bootloader) in most GNU/Linux distributions. GRUB is available as an extra package on the Slackware Linux distribution media.

LILO configuration

LILO is configured through the `/etc/lilo.conf` configuration file. Slackware Linux provides an easy tool to configure LILO. This configuration tool can be started with the **liloconfig** command, and is described in the installation chapter (the section called *Installing Slackware Linux* in Chapter 5).

Manual configuration of LILO is pretty simple. The LILO configuration file usually starts off with some global settings:

```
# Start LILO global section
boot = /dev/sda ❶
#compact        # faster, but won't work on all systems.
prompt ❷
timeout = 50 ❸
# Normal VGA console
vga = normal ❹
```

❶ The *boot* option specifies where the LILO bootloader should be installed. If you want to use LILO as the main bootloader for starting Linux and/or other operating systems, it is a good idea to install LILO in the MBR (Master Boot Record) of the hard disk that you use to boot the system. LILO is installed to the MBR by omitting the partition number, for instance `/dev/hda` or `/dev/sda`. If you want to install LILO to a specific partition, add a partition number, like `/dev/sda1`. Make sure that you have another bootloader in the MBR, or that the partition is made active using **fdisk**. Otherwise you may end up with an unbootable system.

Be cautious if you use partitions with a XFS filesystem! Writing LILO to an XFS partition will overwrite a part of the filesystem. If you use an XFS root (`/`) filesystem, create a non-XFS `/boot` filesystem to which you install LILO, or install LILO to the MBR.

❷ The *prompt* option will set LILO to show a boot menu. From this menu you can select which kernel or operating system should be booted. If you do not have this option enabled, you can still access the bootloader menu by holding the <Shift> key when the bootloader is started.

❸ The *timeout* value specifies how long LILO should wait before the default kernel or OS is booted. The time is specified in tenths of a second, so in the example above LILO will wait 5 seconds before it proceeds with the boot.

❹ You can specify which video mode the kernel should use with the *vga* option. When this is set to *normal* the kernel will use the normal 80x25 text mode.

The global options are followed by sections that add Linux kernels or other operating systems. Most Linux kernel sections look like this:

```
image = /boot/vmlinuz ❶
  root = /dev/sda5 ❷
  label = Slack ❸
  read-only ❹
```

❶ The *image* option specifies the kernel image that should be loaded for this LILO item.

❷ The *root* parameter is passed to the kernel, and will be used by the kernel as the root (`/`) filesystem.

❸ The *label* text is used as the label for this entry in the LILO boot menu.

❹ *read-only* specifies that the root filesystem should be mounted read-only. The filesystem has to be mounted in read-only state to conduct a filesystem check.

LILO installation

LILO does not read the `/etc/lilo.conf` file during the second stage. So, you will have to write changes to the second stage loader when you have changed the LILO configuration. This is also necessary if you install a new kernel with the same filename, since the position of the kernel on the disk may have changed. Reinstalling LILO can simply be done with the **lilo** command:

```
# lilo
Added Slack26 *
Added Slack
```

init

After the kernel is loaded and started, the kernel will start the **init** command. **init** is the parent of all processes, and takes care of starting the system initialization scripts, and spawning login consoles through **agetty**. The behavior of **init** is configured in `/etc/inittab`.

The `/etc/inittab` file is documented fairly well. It specifies what scripts the system should run for different runlevels. A runlevel is a state the system is running in. For instance, runlevel 1 is single user mode, and runlevel 3 is multiuser mode. We will have a short look at a line from `/etc/inittab` to see how it works:

```
rc:2345:wait:/etc/rc.d/rc.M
```

This line specifies that **/etc/rc.d/rc.M** should be started when the system switches to runlevel 2, 3, 4 or 5. The only line you probably ever have to touch is the default runlevel:

```
id:3:initdefault:
```

In this example the default runlevel is set to *3* (multiuser mode). You can set this to another runlevel by replacing 3 with the new default runlevel. Runlevel 4 can particularly be interesting on desktop machines, since Slackware Linux will try to start the GDM, KDM or XDM display manager (in this particular order). These display managers provide a graphical login, and are respectively part of GNOME, KDE and X11.

Another interesting section are the lines that specify what command should handle a console. For instance:

```
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
```

This line specifies that **agetty** should be started on *tty1* (the first virtual terminal) in runlevels 1, 2, 3 and 5. The **agetty** command opens the tty port, and prompts for a login name. **agetty** will then spawn **login** to handle the login. As you can see from the entries, Slackware Linux only starts one console in runlevel 6, namely *tty6*. One might ask what happened to *tty0*, *tty0* certainly exists, and represents the active console.

Since */etc/inittab* is the right place to spawn **agetty** instances to listen for logins, you can also let one or more agetties listen to a serial port. This is especially handy when you have one or more terminals connected to a machine. You can add something like the following line to start an **agetty** instance that listens on COM1:

```
s1:12345:respawn:/sbin/agetty -L ttyS0 9600 vt100
```

Initialization scripts

As explained in the **init** (the section called *init*) section, **init** starts some scripts that handle different runlevels. These scripts perform jobs and change settings that are necessary for a particular runlevel, but they may also start other scripts. Let's look at an example from `/etc/rc.d/rc.M`, the script that **init** executes when the system switches to a multiuser runlevel:

```
# Start the sendmail daemon:
if [ -x /etc/rc.d/rc.sendmail ]; then
  . /etc/rc.d/rc.sendmail start
fi
```

These lines say "execute **/etc/rc.d/rc.sendmail start** if `/etc/rc.d/rc.sendmail` is executable". This indicates the simplicity of the Slackware Linux initialization scripts. Different functionality, for instance network services, can be turned on or off, by twiddling the executable flag on their initialization script. If the initialization script is executable, the service will be started, otherwise it will not. Setting file flags is described in the section called *chmod* in Chapter 8, but we will have a look at a quick example how you can enable and disable sendmail.

To start sendmail when the system initializes, execute:

```
# chmod +x /etc/rc.d/rc.sendmail
```

To disable starting of sendmail when the system initializes, execute:

```
# chmod -x /etc/rc.d/rc.sendmail
```

Most service-specific initialization scripts accept three parameters to change the state of the service: *start*, *restart* and *stop*. These parameters are pretty much self descriptive. For example, if you would like to restart sendmail, you could execute:

```
# /etc/rc.d/rc.sendmail restart
```

If the script is not executable, you have to tell the shell that you would like to execute the file with **sh**. For example:

```
# sh /etc/rc.d/rc.sendmail start
```

Hotplugging

Slackware Linux has supported hotplugging since Slackware Linux 9.1. When enabled, the kernel passes notifications about device events to the **hotplug** command. If a device was added to the system, the hotplugging system will look whether there are any module mappings for the device. If there are, the appropriate device driver module for the device is automatically loaded. Hotplug will remove the module when the device is removed.

The hotplugging system is initialized in `/etc/rc.d/rc.M` by executing **/etc/rc.d/rc.hotplug start**. As with most functionality, you can enable or disable hotplugging by twiddling the executable flag of the `/etc/rc.d/rc.hotplug` script (see the section called *Initialization scripts*).

If hotplug automatically loads modules that you do not want to load, you can add them to the `/etc/hotplug/blacklist` file. This file lists modules that should never be loaded by the hotplugging system, module entries are line separated.

Executing scripts on hotplug events

One of the useful aspects of the hotplug scripts is that you can execute scripts when a device is added or removed. When a device event occurs, hotplug will execute all scripts with the `.hotplug` suffix in `/etc/hotplug.d/<class>/`, in which <*class*> denotes the class in which the device is in that caused the event. There most important classes are:

Table 20-1. Hotplug device classes

Prefix	Description
default	Scripts for this device class are started after any hotplug event.
ieee1394	IEEE1394/Firewire devices.

Prefix	Description
input	Input devices like keyboards and mice.
net	Network devices. Network devices will also trigger bus-specific hotplug events.
pci	Devices on the PCI bus.
scsi	SCSI disks, CD-ROM and tape devices.
usb	Devices that use the USB bus.

Besides the device class, there is something some other convention that is worth observing: it is best to add a number and a dash as a prefix to the script, because this will allow you to order the priority of the scripts. For example, `10-dosomething.hotplug` will be executed before `20-dosomething.hotplug`.

> **Note:** There is black no magic going on with the priority of the script :^). `/sbin/hotplug` uses a wildcard to loop through the scripts, and `10` matches earlier than `20`:
>
> ```
> for I in "${DIR}/$1/"*.hotplug "${DIR}/"default/*.hotplug ; do
> ```
>
> Hopefully this will give some taste of the power of shell scripting.

Now, let's look at an example stub script, that does nothing besides outputting debug messages:

```
#!/bin/sh

cd /etc/hotplug
. ./hotplug.functions

DEBUG=yes export DEBUG

debug_mesg "arguments ($*) env ('env')"

case $ACTION in
  add|register)
    # Stub
  ;;
  remove|unregister)
    # Stub
  ;;
esac
```

By default, the hotplug scripts log at the *notice* level. The standard syslog configuration on Slackware Linux does not log this. To debug hotplugging scripts, it is best to change the logging level. You can do this by replacing the following line in `/etc/hotplug/hotplug.functions`:

```
$LOGGER -t $(basename $0)"[$$]" "$@"
```

to:

```
$LOGGER -p user.info -t $(basename $0)"[$$]" "$@"
```

After this change hotplug debug messages will be visible in `/var/log/messages`.

As you can see the script makes use of the *$ACTION* variable to see what kind of hotplug action took place. Let’s take a look at an example action when this script is used for handling USB events, and named `10-stub.hotplug`:

```
Jul 19 16:13:23 mindbender 10-stub.hotplug[18970]: arguments (usb) env
(SUBSYSTEM=usb OLDPWD=/ DEVPATH=/devices/pci0000:00/0000:00:10.0/usb2/2-1/2-1.3/2-1.3:1.
PATH=/bin:/sbin:/usr/sbin:/usr/bin ACTION=add PWD=/etc/hotplug HOME=/ SHLVL=2
DEVICE=/proc/bus/usb/002/009 PHYSDEVDRIVER=snd-usb-audio INTERFACE=1/2/0
PRODUCT=d8d/651/1 TYPE=0/0/0 DEBUG=yes PHYSDEVBUS=usb SEQNUM=1528 _=/usr/bin/env)
```

That is nice, when I plugged an USB device, the script gets loaded (a couple of times, this is just one of the interesting bits). Actually, this is my USB Phone, which is represented as an USB audio device under GNU/Linux. The problem with USB audio devices is that the device volumes that are saved with **alsactl store** are not automatically set when the device is plugged. But right know we can easily execute **alsactl restore** when the device is plugged. Please note that the script gets very useful information through some environment variables.

To solve the volume problem, we can create a script, say `/etc/hotplug.d/usb/50-usb-audio-volume.hotplug`, that looks like this:

```
#!/bin/sh

cd /etc/hotplug
. ./hotplug.functions
# DEBUG=yes export DEBUG

debug_mesg "arguments ($*) env (`env`)"

case $ACTION in
  add|register)
    if [ $PHYSDEVDRIVER = "snd-usb-audio" ]; then
      /usr/sbin/alsactl restore
    fi
  ;;
  remove|unregister)
  ;;
  *)
    debug_mesg "Unknown action '$ACTION'"
  ;;
esac
```

Based on the environment variables this script can be refined to restrict running **alsactl** for specific USB audio devices.

Chapter 21. Security

Introduction

With the increasing usage of the Internet and wireless networks security is getting more important every day. It is impossible to cover this subject in a single chapter of an introduction to GNU/Linux. This chapter covers some basic security techniques that provide a good start for desktop and server security.

Before we go on to specific subjects, it is a good idea to make some remarks about passwords. Computer authorization largely relies on passwords. Be sure to use good passwords in all situations. Avoid using words, names, birth dates and short passwords. These passwords can easily be cracked with dictionary attacks or brute force attacks against hosts or password hashes. Use long passwords, ideally eight characters or longer, consisting of random letters (including capitals) and numbers.

Closing services

Introduction

Many GNU/Linux run some services that are open to a local network or the Internet. Other hosts can connect to these services by connecting to specific ports. For example, port 80 is used for WWW traffic. The `/etc/services` file contains a table with all commonly used services, and the port numbers that are used for these services.

A secure system should only run the services that are necessary. So, suppose that a host is acting as a web server, it should not have ports open (thus servicing) FTP or SMTP. With more open ports security risks increase very fast, because there is a bigger chance that the software servicing a port has a vulnerability, or is badly configured. The following few sections will help you tracking down which ports are open, and closing them.

Finding open ports

Open ports can be found using a port scanner. Probably the most famous port scanner for GNU/Linux is **nmap**. **nmap** is available through the “n” disk set.

The basic **nmap** syntax is: **nmap host**. The *host* parameter can either be a hostname or IP address. Suppose that we would like to scan the host that **nmap** is installed on. In this case we could specify the *localhost* IP address, *127.0.0.1*:

```
$ nmap 127.0.0.1

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on localhost (127.0.0.1):
(The 1596 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
80/tcp     open        http
```

```
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

In this example you can see that the host has five open ports that are being serviced; ftp, ssh, telnet, http and X11.

inetd

There are two ways to offer TCP/IP services: by running server applications stand-alone as a daemon or by using the internet super server, **inetd**(8). inetd is a daemon which monitors a range of ports. If a client attempts to connect to a port inetd handles the connection and forwards the connection to the server software which handles that kind of connection. The advantage of this approach is that it adds an extra layer of security and it makes it easier to log incoming connections. The disadvantage is that it is somewhat slower than using a stand-alone daemon. It is thus a good idea to run a stand-alone daemon on, for example, a heavily loaded FTP server.

You can check whether **inetd** is running on a host or not with **ps**, for example:

```
$ ps ax | grep inetd
 2845 ?        S      0:00 /usr/sbin/inetd
```

In this example inetd is running with PID (process ID) 2845. **inetd** can be configured using the `/etc/inetd.conf` file. Let's have a look at an example line from inetd.conf:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

This line specifies that inetd should accept FTP connections and pass them to **tcpd**. This may seem a bit odd, because **proftpd** normally handles FTP connections. You can also specify to use proftpd directly in `inetd.conf`, but it is a good idea to give the connection to **tcpd**. This program passes the connection to **proftpd** in turn, as specified. **tcpd** is used to monitor services and to provide host based access control.

Services can be disabled by adding the comment character (#) at the beginning of the line. It is a good idea to disable all services and enable services you need one at a time. After changing `/etc/inetd.conf` **inetd** needs to be restarted to activate the changes. This can be done by sending the HUP signal to the inetd process:

```
# ps ax | grep 'inetd'
   2845 ?        S      0:00 /usr/sbin/inetd
# kill -HUP 2845
```

If you do not need **inetd** at all, it is a good idea to remove it. If you want to keep it installed, but do not want Slackware Linux to load it at the booting process, execute the following command as root:

```
# chmod a-x /etc/rc.d/rc.inetd
```

Chapter 22. Miscellaneous

Scheduling tasks with cron

Slackware Linux includes an implementation of the classic UNIX cron daemon that allows users to schedule tasks for execution at regular intervals. Each user can create, remove, or modify an individual crontab file. This crontab file specifies commands or scripts to be run at specified time intervals. Blank lines in the crontab or lines that begin with a hash (“#”) are ignored.

Each entry in the crontab file must contain 6 fields separated by spaces. These fields are minute, hour, day, month, day of week, and command. Each of the first five fields may contain a time or the “*” wildcard to match all times for that field. For example, to have the **date** command run every day at 6:10 AM, the following entry could be used.

```
10 6 * * * date
```

A user crontab may be viewed with the **crontab -l** command. For a deeper introduction to the syntax of a crontab file, let us examine the default *root* crontab.

```
# crontab -l
# If you don’t want the output of a cron job mailed to you, you have to direct
# any output to /dev/null.  We’ll do this here since these jobs should run
# properly on a newly installed system, but if they don’t the average newbie
# might get quite perplexed about getting strange mail every 5 minutes. :^)
#
# Run the hourly, daily, weekly, and monthly cron jobs.
# Jobs that need different timing may be entered into the crontab as before,
# but most really don’t need greater granularity than this.  If the exact
# times of the hourly, daily, weekly, and monthly cron jobs do not suit your
# needs, feel free to adjust them.
#
# Run hourly cron jobs at 47 minutes after the hour:
47❶ *❷ *❸ *❹ *❺ /usr/bin/run-parts /etc/cron.hourly 1> /dev/null❻
#
# Run daily cron jobs at 4:40 every day:
40 4 * * * /usr/bin/run-parts /etc/cron.daily 1> /dev/null
#
# Run weekly cron jobs at 4:30 on the first day of the week:
30 4 * * 0 /usr/bin/run-parts /etc/cron.weekly 1> /dev/null
#
# Run monthly cron jobs at 4:20 on the first day of the month:
20 4 1 * * /usr/bin/run-parts /etc/cron.monthly 1> /dev/null
```

❶ The first field, `47`, specifies that this job should occur at 47 minutes after the hour.

❷ The second field, `*`, is a wildcard, so this job should occur *every* hour.

❸ The third field, `*`, is a wildcard, so this job should occur *every* day.

❹ The fourth field, `*`, is a wildcard, so this job should occur *every* month.

❺ The fifth field, `*`, is a wildcard, so this job should occur *every* day of the week.

❻ The sixth field, **/usr/bin/run-parts /etc/cron.hourly 1> /dev/null**, specifies the command that should be run at the time specification defined in the first five fields.

The default root crontab is setup to run scripts in `/etc/cron.monthly` on a monthly basis, the scripts in `/etc/cron.weekly` on a weekly basis, the scripts in `/etc/cron.daily` on a daily basis, and the scripts in `/etc/cron.hourly` on an hourly basis. For this reason it is not strictly necessary for an administrator to understand the inner workings of cron at all. With Slackware Linux, you can simply add a new script to one of the above directories to schedule a new periodic task. Indeed, perusing those directories will give you a good idea of the work that Slackware Linux does behind the scenes on a regular basis to keep things like the **slocate** database updated.

Hard disk parameters

Many modern disks offer various features for increasing disk performance and improving integrity. Many of these features can be tuned with the **hdparm** command. Be careful with changing disk settings with this utility, because some changes can damage data on your disk.

You can get an overview of the active settings for a disk by providing the device node of a disk as a parameter to **hdparm**:

```
# hdparm /dev/hda

/dev/hda:
 multcount    =  0 (off)
 IO_support   =  1 (32-bit)
 unmaskirq    =  1 (on)
 using_dma    =  1 (on)
 keepsettings =  0 (off)
 readonly     =  0 (off)
 readahead    = 256 (on)
 geometry     = 65535/16/63, sectors = 78165360, start = 0
```

A common cause for bad disk performance is that DMA was not automatically used by the kernel for a disk. DMA will speed up I/O throughput and offload CPU usage, by making it possible for the disk to directly transfer data from the disk to the system memory. If DMA is turned off, the overview of settings would shows this line:

```
using_dma    =  0 (off)
```

You can easily turn on DMA for this disk with the *-d* parameter of **hdparm**:

```
# hdparm -d 1 /dev/hda

/dev/hda:
 setting using_dma to 1 (on)
 using_dma    =  1 (on)
```

You can do this during every boot by adding the hdparm command to `/etc/rc.d/rc.local`.

The *-i* parameter of **hdparm** is also very useful, because it gives detailed information about a disk:

```
# hdparm -i /dev/hda
/dev/hda:

 Model=WDC WD400EB-00CPF0, FwRev=06.04G06, SerialNo=WD-WCAAT6022342
 Config={ HardSect NotMFM HdSw>15uSec SpinMotCtl Fixed DTR>5Mbs FmtGapReq }
 RawCHS=16383/16/63, TrkSize=57600, SectSize=600, ECCbytes=40
 BuffType=DualPortCache, BuffSize=2048kB, MaxMultSect=16, MultSect=off
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=78163247
 IORDY=on/off, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
 PIO modes:  pio0 pio1 pio2 pio3 pio4
 DMA modes:  mdma0 mdma1 mdma2
 UDMA modes: udma0 udma1 udma2 udma3 udma4 *udma5
 AdvancedPM=no WriteCache=enabled
 Drive conforms to: device does not report version:

 * signifies the current active mode
```

Monitoring memory usage

In some situations it is handy to diagnose information about how memory is used. For example, on a badly performing server you may want to look whether RAM shortage is causing the system to swap pages, or maybe you are setting up a network service, and want to find the optimum caching parameters. Slackware Linux provides some tools that help you analyse how memory is used.

vmstat

vmstat is a command that can provide statistics about various parts of the virtual memory system. Without any extra parameters **vmstat** provides a summary of some relevant statistics:

```
# vmstat
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  0      0 286804   7912  98632    0    0   198     9 1189   783  5  1 93  1
```

Since we are only looking at memory usage in this section, we will only have a look at the *memory* and *swap* fields.

- *swpd*: The amount of virtual memory being used.
- *free*: The amount of memory that is not used at the moment.
- *buff*: The amount of memory used as buffers.
- *cache*: The amount of memory used as cached.
- *si*: The amount of memory that is swapped in from disk per second.
- *si*: The amount of memory that is swapped to disk per second.

It is often useful to see how memory usage changes over time. You can add an interval as a parameter to **vmstat**, to run **vmstat** continuously, printing the current statistics. This interval is in seconds. So, if you want to get updated statistics every second, you can execute:

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 2  0      0 315132   8832  99324    0    0   189    10 1185   767  5  1 93  1
 1  0      0 304812   8832  99324    0    0     0     0 1222  6881 24  8 68  0
 0  0      0 299948   8836  99312    0    0     0     0 1171  1824 41  9 49  0
 [...]
```

Additionally, you can tell **vmstat** how many times it should output these statistics (rather than doing this infinitely). For example, if you would like to print these statistics every two seconds, and five times in total, you could execute **vmstat** in the following manner:

```
# vmstat 2 5
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 2  0      0 300996   9172  99360    0    0   186    10 1184   756  5  1 93  1
 0  1      0 299012   9848  99368    0    0   336     0 1293  8167 20  8 21 51
 1  0      0 294788  11976  99368    0    0  1054     0 1341 12749 14 11  0 76
 2  0      0 289996  13916  99368    0    0   960   176 1320 17636 22 14  0 64
 2  0      0 284620  16112  99368    0    0  1086   426 1351 21217 25 18  0 58
```

VI. Network administration

Chapter 23. Networking configuration

Hardware

Network cards (NICs)

Drivers for NICs are installed as kernel modules. The module for your NIC has to be loaded during the initialization of Slackware Linux. On most systems the NIC is automatically detected and configured during the installation of Slackware Linux. You can reconfigure your NIC with the **netconfig** command. **netconfig** adds the driver (module) for the detected card to `/etc/rc.d/rc.netdevice`.

It is also possible to manually configure which modules should be loaded during the initialization of the system. This can be done by adding a **modprobe** line to `/etc/rc.d/rc.modules`. For example, if you want to load the module for 3Com 59x NICs (3c59x.o), add the following line to `/etc/rc.d/rc.modules`

```
/sbin/modprobe 3c59x
```

PCMCIA cards

Supported PCMCIA cards are detected automatically by the PCMCIA software. The pcmcia-cs packages from the “a” disk set provides PCMCIA functionality for Slackware Linux.

Configuration of interfaces

Network cards are available under Linux through so-called “interfaces”. The **ifconfig**(8) command can be used to display the available interfaces:

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:20:AF:F6:D4:AD
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1301 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1529 errors:0 dropped:0 overruns:0 carrier:0
          collisions:1 txqueuelen:100
          RX bytes:472116 (461.0 Kb)  TX bytes:280355 (273.7 Kb)
          Interrupt:10 Base address:0xdc00

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8482 (8.2 Kb)  TX bytes:8482 (8.2 Kb)
```

Network cards get the name ethn, in which n is a number, starting with 0. In the example above, the first network card (eth0) already has an IP address. But unconfigured interfaces have no IP address, the **ifconfig** will not show IP addresses for unconfigured interfaces. Interfaces can be configured in the `/etc/rc.d/rc.inet1.conf` file. You can simply read the comments, and fill in the required information. For example:

```
# Config information for eth0:
IPADDR[0]="192.168.1.1"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
```

In this example the IP address 192.168.1.1 with the 255.255.255.0 netmask is assigned to the first ethernet interface (eth0). If you are using a DHCP server you can change the *USE_DHCP=""* line to *USE_DHP[n]="yes"* (swap “n” with the interface number). Other variables, except *DHCP_HOSTNAME* are ignored when using DHCP. For example:

```
IPADDR[1]=""
NETMASK[1]=""
USE_DHCP[1]="yes"
DHCP_HOSTNAME[1]=""
```

The same applies to other interfaces. You can activate the settings by rebooting the system or by executing **/etc/rc.d/rc.inet1**. It is also possible to reconfigure only one interface with **/etc/rc.d/rc.inet1 ethX_restart**, in which *ethX* should be replaced by the name of the interface that you would like to reconfigure.

Configuration of interfaces (IPv6)

Introduction

IPv6 is the next generation internet protocol. One of the advantages is that it has a much larger address space. In IPv4 (the internet protocol that is commonly used today) addresses are 32-bit, this address space is almost completely used right now, and there is a lack of IPv4 addresses. IPv6 uses 128-bit addresses, which provides an unimaginable huge address space (2^128 addresses). IPv6 uses another address notation, first of all hex numbers are used instead of decimal numbers, and the address is noted in pairs of 16-bits, separated by a colon (“:”). Let’s have a look at an example address:

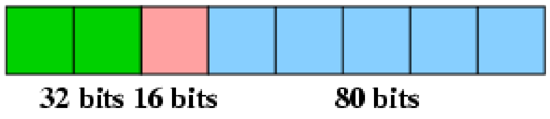
```
fec0:ffff:a300:2312:0:0:0:1
```

A block of zeroes can be replaced by two colons (“::”). Thus, thee address above can be written as:

```
fec0:ffff:a300:2312::1
```

Each IPv6 address has a prefix. Normally this consists of two elements: 32 bits identifying the address space the provider provides you, and a 16-bit number that specifies the network. These two elements form the prefix, and in this case the prefixlength is 32 + 16 = 48 bits. Thus, if you have a /48 prefix you can make 2^16 subnets and have 2^80 hosts on each subnet. The image below shows the structure of an IPv6 address with a 48-bit prefix.

Figure 23-1. The anatomy of an IPv6 address



There are a some specially reserved prefixes, most notable include:

Table 23-1. Important IPv6 Prefixes

Prefix	Description
fe80::	Link local addresses, which are not routed.
fec0::	Site local addresses, which are locally routed, but not on or to the internet.
2002::	6to4 addresses, which are used for the transition from IPv4 to IPv6.

Slackware Linux IPv6 support

The Linux kernel binaries included in Slackware Linux do not support IPv6 by default, but support is included as a kernel module. This module can be loaded using **modprobe**:

```
# modprobe ipv6
```

You can verify if IPv6 support is loaded correctly by looking at the kernel output using the **dmesg**:

```
$ dmesg
[..]
IPv6 v0.8 for NET4.0
```

IPv6 support can be enabled permanently by adding the following line to `/etc/rc.d/rc.modules`:

```
/sbin/modprobe ipv6
```

Interfaces can be configured using **ifconfig**. But it is recommended to make IPv6 settings using the **ip** command, which is part of the “iputils” package that can be found in the `extra/` directory of the Slackware Linux tree.

Adding an IPv6 address to an interface

If there are any router advertisers on a network there is a chance that the interfaces on that network already received an IPv6 address when the IPv6 kernel support was loaded. If this is not the case an IPv6 address can be added to an interface using the **ip** utility. Suppose we want to add the address “fec0:0:0:bebe::1” with a prefix length of 64 (meaning “fec0:0:0:bebe” is the prefix). This can be done with the following command syntax:

```
# ip -6 addr add <ip6addr>/<prefixlen> dev <device>
```

For example:

```
# ip -6 addr add fec0:0:0:bebe::1/64 dev eth0
```

Wireless interfaces

Wireless interfaces usually require some additional configuration, like setting the ESSID, WEP keys and the wireless mode. Interface settings that are specific to wireless interfaces can be set in the `/etc/rc.d/rc.wireless.conf` file. The **/etc/rc.d/rc.wireless** script configures wireless interfaces based on descriptions from `/etc/rc.d/rc.wireless.conf`. In `rc.wireless.conf` settings are made per interface MAC address. By default this file has a section that matches any interface:

```
## NOTE : Comment out the following five lines to activate the samples below ...
## --------- START SECTION TO REMOVE -----------
## Pick up any Access Point, should work on most 802.11 cards
*)
    INFO="Any ESSID"
    ESSID="any"
    ;;
## ---------- END SECTION TO REMOVE ------------
```

It is generally a good idea to remove this section to make per-card settings. If you are lazy and only have one wireless card, you can leave this section in and add any configuration parameters you need. Since this section matches any wireless interface the wireless card you have will be matched and configured. You can now add a sections for your wireless interfaces. Each section has the following format:

```
<MAC address>)
    <settings>
;;
```

You can find the MAC address of an interface by looking at the **ifconfig** output for the interface. For example, if a wireless card has the *eth1* interface name, you can find the MAC address the following way:

```
# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:01:F4:EC:A5:32
          inet addr:192.168.2.2  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:1 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:504 (504.0 b)
          Interrupt:5 Base address:0x100
```

The hexadecimal address that is printed after *HWaddr* is the MAC address, in this case *00:01:F4:EC:A5:32*. When you have found the MAC address of the interface you can add a section for the device to `/etc/rc.d/rc.wireless.conf`. For example:

```
00:01:F4:EC:A5:32)
    INFO="Cabletron Roamabout WLAN NIC"
    ESSID="home"
    CHANNEL="8"
    MODE="Managed"
    KEY="1234-5678-AB"
    ;;
```

This will set the interface with MAC address *00:01:F4:EC:A5:32* to use the ESSID *home*, work in *Managed* mode on channel *8*. The key used for WEP encryption is *1234-5678-AB*. There are many other parameters that can be set. For an overview of all parameters, refer to the last example in `rc.wireless.conf`.

After configuring a wireless interface, you can activate the changes by executing the network initialization script **/etc/rc.d/rc.inet1**. You can see the current wireless settings with the **iwconfig** command:

```
eth1      IEEE 802.11-DS  ESSID:"home"  Nickname:"HERMES I"
          Mode:Managed  Frequency:2.447 GHz  Access Point: 02:20:6B:75:0C:56
          Bit Rate:2 Mb/s   Tx-Power=15 dBm   Sensitivity:1/3
          Retry limit:4   RTS thr:off   Fragment thr:off
          Encryption key:1234-5678-AB
          Power Management:off
          Link Quality=0/92  Signal level=134/153  Noise level=134/153
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:27  Invalid misc:0   Missed beacon:0
```

Resolving

Hostname

Each computer on the internet has a hostname. If you do not have a hostname that is resolvable with DNS, it is still a good idea to configure your hostname, because some software uses it. You can configure the hostname in `/etc/HOSTNAME`. A single line with the hostname of the machine will suffice. Normally a hostname has the following form: host.domain.tld, for example darkstar.slackfans.org. Be aware that the hostname has to be resolvable, meaning that GNU/Linux should be able to convert the hostname to an IP address. You can make sure the hostname is resolvable by adding it to `/etc/hosts`. Read the following section for more information about this file.

/etc/hosts

`/etc/hosts` is a table of IP addresses with associated hostnames. This file can be used to name computers in a small network. Let's look at an example of the `/etc/hosts` file:

```
127.0.0.1               localhost
192.168.1.1             tazzy.slackfans.org tazzy
192.168.1.169           flux.slackfans.org
```

The *localhost* line should always be present. It assigns the name *localhost* to a special interface, the loopback. In this example the names *tazzy.slackfans.org* and *tazzy* are assigned to the IP address 192.168.1.1, and the name *flux.slackfans.org* is assigned to the IP address 192.168.1.169. On the system with this file both computers are available via the mentioned hostnames.

It is also possible to add IPv6 addresses, which will be used if your system is configured for IPv6. This is an example of a `/etc/hosts` file with IPv4 and IPv6 entries:

```
# IPv4 entries
127.0.0.1               localhost
192.168.1.1             tazzy.slackfans.org tazzy
192.168.1.169           gideon.slackfans.org
```

```
# IPv6 entries
::1   localhost
fec0:0:0:bebe::2 flux.slackfans.org
```

Please note that “::1” is the default IPv6 loopback.

/etc/resolv.conf

The `/etc/resolv.conf` file is used to specify which nameservers the system should use. A nameserver converts hostnames to IP addresses. Your provider should have given you at least two name name server addresses (DNS servers). You can add these nameservers to `/etc/resolv.conf` by adding the line *nameserver ipaddress* for each nameserver. For example:

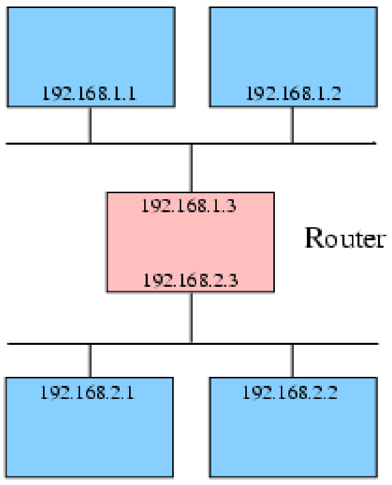
```
nameserver 192.168.1.1
nameserver 192.168.1.169
```

You can check wether the hostnames are tranlated correctly or not with the **host hostname** command. Swap *hostname* with an existing hostname, for example the website of your internet service provider.

IPv4 Forwarding

IPv4 forwarding connects two or more networks by sending packets which arrive on one interface to another interface. This makes it possible to let a GNU/Linux machine act as a router. For example, you can connect multiple networks, or your home network with the internet. Let’s have a look at an example:

Figure 23-2. Router example



In dit example there are two networks, 192.168.1.0 and 192.168.2.0. Three hosts are connected to both network. One of these hosts is connected to both networks with interfaces. The interface on the 192.168.1.0 network has IP address 192.168.1.3, the interface on the 192.168.2.0 network has IP address 192.168.2.3. If the host acts as a router between both networks it forwards packets from the 192.168.1.0 network to the 192.168.2.0 network and vise versa. Routing of normal IPv4 TCP/IP packages can be enabled by enabling IPv4 forwarding.

IPv4 forwarding can be enabled or disabled under Slackware Linux by changing the executable bit of the `/etc/rc.d/rc.ip_forward` file. If the executable bit is set on this file, IP forwarding will be enabled during the system boot, otherwise it will not. You can check whether the executable bit is enabled with **ls -l** (a description of the **ls** command can be found in the section called *ls* in Chapter 8).

It is also possible to enable IPv4 forwarding on a running system with the following command (0 disables forwarding, 1 enables forwarding):

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Be cautious! By default there are no active packet filters. This means that anyone can access other networks. Traffic can be filtered and logged with the iptables kernel packet filter. Iptables can be administrated through the **iptables** command. NAT (Network Address Translation) is also a subset of iptables, and can be controlled and enabled through the **iptables** command. NAT makes it possible to “hide” a network behind one IP address. This allows you to use the internet on a complete network with only one IP address.

Chapter 24. IPsec

Theory

IPsec is a standard for securing IP communication through authentication, and encryption. Besides that it can compress packets, reducing traffic. The following protocols are part of the IPsec standard:

- AH (Authentication Header) provides authenticity guarantee for transported packets. This is done by checksumming the packages using a cryptographic algorithm. If the checksum is found to be correct by the receiver, the receiver can be assured that the packet is not modified, and that the packet really originated from the reported sender (provided that the keys are only known by the sender and receiver).
- ESP (Encapsulating Security Payload) is used to encrypt packets. This makes the data of the packet confident, and only readable by the host with the right decryption key.
- IPcomp (IP payload compression) provides compression before a packet is encrypted. This is useful, because encrypted data generally compresses worse than unencrypted data.
- IKE (Internet Key Exchange) provides the means to negotiate keys in secrecy. Please note that IKE is optional, keys can be configured manually.

There are actually two modes of operation: *transport mode* is used to encrypt normal connections between two hosts, *tunnel mode* encapsulates the original package in a new header. In this chapter we are going to look at the transport mode, because the primary goal of this chapter is to show how to set up a secure connection between two hosts.

There are also two major methods of authentication. You can use manual keys, or an Internet Key Exchange (IKE) daemon, like racoon, that automatically exchanges keys securely betwoon two hosts. In both cases you need to set up a policy in the Security Policy Database (SPD). This database is used by the kernel to decide what kind of security policy is needed to communicate with another host. If you use manual keying you also have to set up Security Association Database (SAD) entries, which specifies what encryption algorithmn and key should be used for secure communication with another host. If you use an IKE daemon the security associations are automatically established.

Kernel configuration

Native IPsec support is only available in Linux 2.6.x kernels. Earlier kernels have no native IPsec support. So, make sure that you have a 2.6.x kernel. The 2.6 kernel is available in Slackware Linux 10.0 and 10.1 from the `testing` directory on CD2 of the Slackware Linux CD sets, or any of the official Slackware Linux mirrors. The default Slackware Linux 2.6 kernel has support for AH, ESP and IPcomp in for both IPv4 and IPv6. If you are compiling a custom kernel enable use at least the following options in your kernel configuration:

```
CONFIG_INET_AH=y
CONFIG_INET_ESP=y
CONFIG_INET_IPCOMP=y
```

Or you can compile support for IPsec protocols as a module:

```
CONFIG_INET_AH=m
```

```
CONFIG_INET_ESP=m
CONFIG_INET_IPCOMP=m
```

In this chapter we are only going to use AH and ESP transformations, but it is not a bad idea to enable IPComp transformation for further configuration of IPsec.

When you choose to compile IPsec support as a module, make sure that the required modules are loaded. For example, if you are going to use ESP for IPv4 connections, load the *esp4* module.

Compile the kernel as usual and boot it.

Installing IPsec-Tools

The next step is to install the IPsec-Tools (http://ipsec-tools.sourceforge.net). These tools are ports of the KAME (http://www.kame.net) IPsec utilities. Download the latest sources and unpack, configure and install them:

```
# tar jxf ipsec-tools-x.y.z.tar.bz2
# cd ipsec-tools-x.y.z
# CFLAGS="-O2 -march=i486 -mcpu=i686" \
  ./configure --prefix=/usr \
              --sysconfdir=/etc \
              --localstatedir=/var \
              --enable-hybrid \
              --enable-natt \
              --enable-dpd \
              --enable-frag \
              i486-slackware-linux
# make
# make install
```

Replace *x.y.z* with the version of the downloaded sources. The most notable flags that we specify during the configuration of the sources are:

- *--enable-dpd*: enables dead peer detection (DPD). DPD is a method for detecting wether any of the hosts for which security associations are set up is unreachable. When this is the case the security associations to that host can be removed.
- *--enable-natt*: enables NAT traversal (NAT-T). Since NAT alters the IP headers, this causes problems for guaranteeing authenticity of a packet. NAT-T is a method that helps overcoming this problem. Configuring NAT-T is beyond the scope of this article.

Setting up IPsec with manual keying

Introduction

We will use an example as the guideline for setting up an encrypted connection between to hosts. The hosts have the IP addresses *192.168.1.1* and *192.168.1.169*. The “transport mode” of operation will be used with AH and ESP transformations and manual keys.

Writing the configuration file

The first step is to write a configuration file we will name `/etc/setkey.conf`. On the first host (192.168.1.1) the following `/etc/setkey.conf` configuration will be used:

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

add 192.168.1.1 192.168.1.169 ah 0x200 -A hmac-md5
0xa731649644c5dee92cbd9c2e7e188ee6;
add 192.168.1.169 192.168.1.1 ah 0x300 -A hmac-md5
0x27f6d123d7077b361662fc6e451f65d8;

add 192.168.1.1 192.168.1.169 esp 0x201 -E 3des-cbc
0x656c8523255ccc23a66c1917aa0cf30991fce83532a4b224;
add 192.168.1.169 192.168.1.1 esp 0x301 -E 3des-cbc
0xc966199f24d095f3990a320d749056401e82b26570320292

spdadd 192.168.1.1 192.168.1.169 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

The first line (a line ends with a ";") adds a key for the header checksumming for packets coming from 192.168.1.1 going to 192.168.1.169. The second line does the same for packets coming from 192.168.1.169 to 192.168.1.1. The third and the fourth line define the keys for the data encryption the same way as the first two lines. Finally the "spadd" lines define two policies, namely packets going out from 192.168.1.1 to 192.168.1.169 should be (require) encoded (esp) and "signed" with the authorization header. The second policy is for incoming packets and it is the same as outgoing packages.

Please be aware that you should not use these keys, but your own secretly kept unique keys. You can generate keys using the `/dev/random` device:

```
# dd if=/dev/random count=16 bs=1 | xxd -ps
```

This command uses dd to output 16 bytes from `/dev/random`. Don't forget to add "0x" at the beginning of the line in the configuration files. You can use the 16 byte (128 bits) for the hmac-md5 algorithm that is used for AH. The 3des-cbc algorithm that is used for ESP in the example should be fed with a 24 byte (192 bits) key. These keys can be generated with:

```
# dd if=/dev/random count=24 bs=1 | xxd -ps
```

Make sure that the `/etc/setkey.conf` file can only be read by the root user. If normal users can read the keys IPsec provides no security at all. You can do this with:

```
# chmod 600 /etc/setkey.conf
```

The same `/etc/setkey.conf` can be created on the 192.168.1.169 host, with inverted *`-P in`* and *`-P out`* options. So, the `/etc/setkey.conf` will look like this:

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

add 192.168.1.1 192.168.1.169 ah 0x200 -A hmac-md5
0xa731649644c5dee92cbd9c2e7e188ee6;
add 192.168.1.169 192.168.1.1 ah 0x300 -A hmac-md5
0x27f6d123d7077b361662fc6e451f65d8;

add 192.168.1.1 192.168.1.169 esp 0x201 -E 3des-cbc
0x656c8523255ccc23a66c1917aa0cf30991fce83532a4b224;
add 192.168.1.169 192.168.1.1 esp 0x301 -E 3des-cbc
0xc966199f24d095f3990a320d749056401e82b26570320292

spdadd 192.168.1.1 192.168.1.169 any -P in ipsec
           esp/transport//require
           ah/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P out ipsec
           esp/transport//require
           ah/transport//require;
```

Activating the IPsec configuration

The IPsec configuration can be activated with the **setkey** command:

```
# setkey -f /etc/setkey.conf
```

If you want to enable IPsec permanently you can add the following line to `/etc/rc.d/rc.local` on both hosts:

```
/usr/sbin/setkey -f /etc/setkey.conf
```

After configuring IPsec you can test the connection by running **tcpdump** and simultaneously pinging the other host. You can see if AH and ESP are actually used in the **tcpdump** output:

```
# tcpdump -i eth0
tcpdump: listening on eth0
11:29:58.869988 terrapin.taickim.net > 192.168.1.169: AH(spi=0x00000200,seq=0x40f): ESP(
11:29:58.870786 192.168.1.169 > terrapin.taickim.net: AH(spi=0x00000300,seq=0x33d7): ESP
```

Setting up IPsec with automatic key exchanging

Introduction

The subject of automatical key exchange is already touched shortly in the introduction of this chapter. Put simply, IPsec with IKE works in the following steps.

1. Some process on the host wants to connect to another host. The kernel checks whether there is a security policy set up for the other host. If there already is a security association corresponding with the policy the connection can be made, and will be authenticated, encrypted and/or compressed as defined in the security association. If there is no security association, the kernel will request a user-land IKE daemon to set up the necessary security association(s).
2. During the first phase of the key exchange the IKE daemon will try to verify the authenticity of the other host. This is usually done with a preshared key or certificate. If the authentication is successful a secure channel is set up between the two hosts, usually called a IKE security association, to continue the key exchange.
3. During the second phase of the key exchange the security associations for communication with the other host are set up. This involves choosing the encryption algorithm to be used, and generating keys that are used for encryption of the communication.
4. At this point the first step is repeated again, but since there are now security associations the communication can proceed.

The racoon IKE daemon is included with the KAME IPsec tools, the sections that follow explain how to set up racoon.

Using racoon with a preshared key

As usual the first step to set up IPsec is to define security policies. In contrast to the manual keying example you should not set up security associations, because racoon will make them for you. We will use the same host IPs as in the example above. The security policy rules look like this:

```
#!/usr/sbin/setkey -f

# Flush the SAD and SPD
flush;
spdflush;

spdadd 192.168.1.1 192.168.1.169 any -P out ipsec
 esp/transport//require;

spdadd 192.168.1.169 192.168.1.1 any -P in ipsec
 esp/transport//require;
```

Cautious souls have probably noticed that AH policies are missing in this example. In most situations this is no problem, ESP can provide authentication. But you should be aware that the authentication is more narrow; it does not protect information outside the ESP headers. But it is more efficient than encapsulating ESP packets in AH.

With the security policies set up you can configure racoon. Since the connection-specific information, like the authentication method is specified in the phase one configuration. We can use a general phase two configuration. It is also possible to make specific phase two settings for certain hosts. But generally speaking a general configuration will often suffice in simple setups. We will also add paths for the preshared key file, and certification directory. This is an example of `/etc/racoon/racoon.conf` with the paths and a general phase two policy set up:

```
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

sainfo anonymous {
{
 pfs_group 2;
 lifetime time 1 hour;
 encryption_algorithm 3des, blowfish 448, rijndael;
 authentication_algorithm hmac_sha1, hmac_md5;
 compression_algorithm deflate;
}
```

The *`sainfo`* identifier is used to make a block that specifies the settings for security associations. Instead of setting this for a specific host, the *`anonymous`* parameter is used to specify that these settings should be used for all hosts that do not have a specific configuration. The *`pfs_group`* specifies which group of Diffie-Hellman exponentiations should be used. The different groups provide different lengths of base prime numbers that are used for the authentication process. Group 2 provides a 1024 bit length if you would like to use a greater length, for increased security, you can use another group (like 14 for a 2048 bit length). The *`encryption_algorithm`* specifies which encryption algorithms this host is willing to use for ESP encryption. The *`authentication_algorithm`* specifies the algorithm to be used for ESP Authentication or AH. Finally, the *`compression_algorithm`* is used to specify which compression algorithm should be used when IPcomp is specified in an association.

The next step is to add a phase one configuration for the key exchange with the other host to the racoon configuration. For example:

```
remote 192.168.1.169
{
 exchange_mode aggressive, main;
 my_identifier address;
 proposal {
  encryption_algorithm 3des;
  hash_algorithm sha1;
  authentication_method pre_shared_key;
  dh_group 2;
 }
}
```

The *`remote`* block specifies a phase one configuration. The *`exchange_mode`* is used to configure what exchange mode should be used for phase. You can specify more than one exchange mode, but the first method is used if this host is the initiator of the key exchange. The *`my_identifier`* option specifies what identifier should be sent to the remote host. If this option committed *`address`* is used, which sends the IP address as the identifier. The *`proposal`* block specifies parameter that will be proposed to the other host during phase one authentication. The *`encryption_algorithm`*, and *`dh_group`* are explained above. The *`hash_algorithm`* option is mandatory, and configures the hash algorithm that should be used. This can be *`md5`*, or *`sha1`*. The *`authentication_method`* is crucial for this configuration, as this parameter is used to specify that a preshared key should be used, with *`pre_shared_key`*.

With racoon set up there is one thing left to do, the preshared key has to be added to `/etc/racoon/psk.txt`. The syntax is very simple, each line contains a host IP address and a key. These parameters are separated with a tab. For example:

```
192.168.1.169 somekey
```

Activating the IPsec configuration

At this point the configuration of the security policies and racoon is complete, and you can start to test the configuration. It is a good idea to start **racoon** with the *-F* parameter. This will run racoon in the foreground, making it easier to catch error messages. To wrap it up:

```
# setkey -f /etc/setkey.conf
# racoon -F
```

Now that you have added the security policies to the security policy database, and started Racoon, you can test your IPsec configuration. For instance, you could ping the other host to start with. The first time you ping the other host, this will fail:

```
$ ping 192.168.1.169
connect: Resource temporarily unavailable
```

The reason for this is that the security associations still have to be set up. But the ICMP packet will trigger the key exchange. ping will trigger the key exchange. You can see whether the exchange was succesful or not by looking at the Racoon log messages in `/var/log/messages`, or the racoon output if you started racoon in the foreground. A succesful key exhange looks like this:

```
 Apr  4 17:14:58 terrapin racoon: INFO: IPsec-SA request for 192.168.1.169 queued due to
 Apr  4 17:14:58 terrapin racoon: INFO: initiate new phase 1 negotiation: 192.168.1.1[50
 Apr  4 17:14:58 terrapin racoon: INFO: begin Aggressive mode.
 Apr  4 17:14:58 terrapin racoon: INFO: received Vendor ID: DPD
 Apr  4 17:14:58 terrapin racoon: NOTIFY: couldn't find the proper pskey, try to get one
 Apr  4 17:14:58 terrapin racoon: INFO: ISAKMP-SA established 192.168.1.1[500]-192.168.1
 Apr  4 17:14:59 terrapin racoon: INFO: initiate new phase 2 negotiation: 192.168.1.1[0]
 Apr  4 17:14:59 terrapin racoon: INFO: IPsec-SA established: ESP/Transport 192.168.1.16
 Apr  4 17:14:59 terrapin racoon: INFO: IPsec-SA established: ESP/Transport 192.168.1.1-
```

After the key exchange, you can verify that IPsec is set up correctly by analyzing the packets that go in and out with **tcpdump**. tcpdump is available in the *n* diskset. Suppose that the outgoing connection to the other host goes through the *eth0* interface, you can analyze the packats that go though the *eth0* interface with **tcpdump -i eth0**. If the outgoing packets are encrypted with ESP, you can see this in the **tcpdump** output. For example:

```
# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
17:27:50.241067 IP terrapin.taickim.net > 192.168.1.169: ESP(spi=0x059950e8,seq=0x9)
17:27:50.241221 IP 192.168.1.169 > terrapin.taickim.net: ESP(spi=0x0ddff7e7,seq=0x9)
```

Chapter 25. The internet super server

Introduction

There are two ways to offer TCP/IP services: by running server applications standalone as a daemon or by using the internet super server, **inetd**(8). **inetd** is a daemon which monitors a range of ports. If a client attempts to connect to a port **inetd** handles the connection and forwards the connection to the server software which handles that kind of connection. The advantage of this approach is that it adds an extra layer of security and it makes it easier to log incoming connections. The disadvantage is that it is somewhat slower than using a standalone daemon. It is thus a good idea to run a standalone daemon on, for example, a heavily loaded FTP server.

Configuration

inetd can be configured using the `/etc/inetd.conf` file. Let's have a look at an example line from `inetd.conf`:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

This line specifies that **inetd** should accept FTP connections and pass them to **tcpd**. This may seem a bit odd, because **proftpd** normally handles FTP connections. You can also specify to use **proftpd** directly in `inetd.conf`, but Slackware Linux normally passes the connection to **tcpd**. This program passes the connection to **proftpd** in turn, as specified. **tcpd** is used to monitor services and to provide host based access control.

Services can be disabled by adding the comment character (#) at the beginning of the line. It is a good idea to disable all services and enable services you need one at a time. After changing `/etc/inetd.conf` **inetd** needs to be restarted to activate the changes. This can be done by sending the HUP signal to the inetd process:

```
# ps ax | grep 'inetd'
   64 ?        S      0:00 /usr/sbin/inetd
# kill -HUP 64
```

Or you can use the `rc.inetd` init script to restart **inetd**:

```
# /etc/rc.d/rc.inetd restart
```

TCP wrappers

As you can see in `/etc/inetd.conf` connections for most protocols are made through **tcpd**, instead of directly passing the connection to a service program. For example:

```
# File Transfer Protocol (FTP) server:
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  proftpd
```

In this example ftp connections are passed through **tcpd**. **tcpd** logs the connection through syslog and allows for additional checks. One of the most used features of **tcpd** is host-based access control.

Hosts that should be denied are controlled via `/etc/hosts.deny`, hosts that should be allowed via `/etc/hosts.allow`. Both files have one rule on each line of the following form:

```
service: hosts
```

Hosts can be specified by hostname or IP address. The ALL keyword specifies all hosts or all services.

Suppose we want to block access to all services managed through **tcpd**, except for host “trusted.example.org”. To do this the following `hosts.deny` and `hosts.allow` files should be created.

`/etc/hosts.deny`:

```
ALL: ALL
```

`/etc/hosts.allow`:

```
ALL: trusted.example.org
```

In the `hosts.deny` access is blocked to all (ALL) services for all (ALL) hosts. But `hosts.allow` specifies that all (ALL) services should be available to “trusted.example.org”.

Chapter 26. Apache

Introduction

Apache is the most popular web server since April 1996. It was originally based on NCSA httpd, and has grown into a full-featured HTTP server. Slackware Linux currently uses the 1.3.x branch of Apache. This chapter is based on Apache 1.3.x.

Installation

Apache can be installed by adding the `apache` pacakge from the "n" disk set. If you also want to use PHP, the `php` ("n" disk set) and `mysql` ("ap" disk set) are also required. MySQL is required, because the precompiled PHP depends on MySQL shared libraries. You do not have to run MySQL itself. After installing Apache it can be started automatically while booting the system by making the `/etc/rc.d/rc.httpd` file executable. You can do this by executing:

```
# chmod a+x /etc/rc.d/rc.httpd
```

The Apache configuration can be altered in the `/etc/apache/httpd.conf` file. Apache can be stopped/started/restarted every moment with the **apachectl** command, and the *`stop`*, *`start`* and *`restart`* parameters. For example, execute the following command to restart Apache:

```
# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
```

User directories

Apache provides support for so-call user directories. This means every user gets webspace in the form of *http://host/~user/*. The contents of "~user/" is stored in a subdirectory in the home directory of the user. This directory can be specified using the "UserDir" option in `httpd.conf`, for example:

```
UserDir public_html
```

This specifies that the `public_html` directory should be used for storing the webpages. For example, the webpages at URL *http://host/~snail/* are stored in `/home/snail/public_html`.

Virtual hosts

The default documentroot for apache under Slackware Linux is `/var/www/htdocs`. Without using virtual hosts every client connecting to the Apache server will receive the website in this directory. So, if we have two hostnames pointing to the server, "www.example.org" and "forum.example.org", both will display the same website. You can make seperate sites for different hostnames by using virtual hosts.

In this example we are going to look how you can make two virtual hosts, one for "www.example.org", with the documentroot `/var/www/htdocs-www`, and "forum.example.org", with the documentroot `/var/www/htdocs-forum`. First of all we have to specify which IP addresses Apache should listen to. Somewhere in the `/etc/apache/httpd.conf` configuration file you will find the following line:

```
#NameVirtualHost *:80
```

This line has to be commented out to use name-based virtual hosts. Remove the comment character (#) and change the parameter to “BindAddress IP:port”, or “BindAddress *:port” if you want Apache to bind to all IP addresses the host has. Suppose we want to provide virtual hosts for IP address 192.168.1.201 port 80 (the default Apache port), we would change the line to:

```
NameVirtualHost 192.168.1.201:80
```

Somewhere below the NameVirtualHost line you can find a commented example of a virtualhost:

```
#<VirtualHost *:80>
#    ServerAdmin webmaster@dummy-host.example.com
#    DocumentRoot /www/docs/dummy-host.example.com
#    ServerName dummy-host.example.com
#    ErrorLog logs/dummy-host.example.com-error_log
#    CustomLog logs/dummy-host.example.com-access_log common
#</VirtualHost>
```

You can use this example as a guideline. For example, if we want to use all the default values, and we want to write the logs for both virtual hosts to the default Apache logs, we would add these lines:

```
<VirtualHost 192.168.1.201:80>
 DocumentRoot /var/www/htdocs-www
 ServerName www.example.org
</VirtualHost>

<VirtualHost 192.168.1.201:80>
 DocumentRoot /var/www/htdocs-forum
 ServerName forum.example.org
</VirtualHost>
```

Chapter 27. BIND

Introduction

The domain name system (DNS) is used to convert human-friendly host names (for example www.slackware.com) to IP addresses. BIND (Berkeley Internet Name Domain) is the most widely used DNS daemon, and will be covered in this chapter.

Delegation

One of the main features is that DNS requests can be delegated. For example, suppose that you own the “linuxcorp.com” domain. You can provide the authorized nameservers for this domain, you nameservers are authorative for the “linuxcorp.com”. Suppose that there are different branches within your company, and you want to give each branch authority over their own zone, that is no problem with DNS. You can delegate DNS for e.g. “sales.linuxcorp.com” to another nameserver within the DNS configuration for the “linuxcorp.com” zone.

The DNS system has so-called root servers, which delegate the DNS for millions of domain names and extensions (for example, country specific extensions, like “.nl” or “.uk”) to authorized DNS servers. This system allows a branched tree of delegation, which is very flexible, and distributes DNS traffic.

DNS records

The following types are common DNS records:

Table 27-1. DNS records

Prefix	Description
A	An A records points to an IP address.
CNAME	A CNAME record points to another DNS entry.
MX	A MX record specifies which should handle mail for the domain.

Masters and slaves

Two kinds of nameservers can be provided for a domain name: a master and slaves. The master server DNS records are authorative. Slave servers download their DNS record from the master servers. Using slave servers besides a master server is recommended for high availability and can be used for load-balancing.

Making a caching nameserver

A caching nameserver provides DNS services for a machine or a network, but does not provide DNS for a domain. That means it can only be used to convert hostnames to IP addresses. Setting up a nameserver with Slackware Linux is fairly easy, because BIND is configured as a caching nameserver by default. Enabling the caching nameserver takes just two steps: you have to install BIND and alter the initialization scripts. BIND can be installed by adding the bind package from the "n" disk set. After that bind can be started by executing the **named(8)** command. If want to start BIND by default, make the `/etc/rc.d/rc.bind` file executable. This can be done by executing the following command as root:

```
# chmod a+x /etc/rc.d/rc.bind
```

If you want to use the nameserver on the machine that runs BIND, you also have to alter `/etc/resolv.conf`.