Slackware Linux Basics

For Slackware Linux 10.0

Daniël de Kok
Slackware Linux Basics: For Slackware Linux 10.0
by Daniël de Kok

Copyright © 2002, 2003, 2004 by Daniël de Kok

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of this book must retain the above copyright notice, this list of conditions and the following disclaimer.

2. All advertising materials mentioning features or use of this book must display the following acknowledgement: This product includes content written by Daniël de Kok.

3. The name of the author may not be used to endorse or promote products derived from this book without specific prior written permission.

THIS BOOK IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS BOOK, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Linux is a registered trademark of Linus Torvalds. Slackware Linux is a registered trademark of Patrick Volkerding and Slackware Linux, Inc.
Table of Contents

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
   Slackware Linux 10.0 features .............................................................................................. 2
   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 ....................................................................................................................... 4

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 .................................................................................................. 12

II. GNU/Linux Basics ..................................................................................................................... 20

6. The Bash shell ............................................................................................................................. 21
   Introduction ............................................................................................................................. 21
   Starting the shell .................................................................................................................... 21
   Shell basics ............................................................................................................................ 21
       Executing commands ....................................................................................................... 21
       Browsing through shell commands ............................................................................... 21
       Completion ....................................................................................................................... 22
       Wildcards ........................................................................................................................ 22
           Matching a string of characters .................................................................................. 22
           Matching single characters ...................................................................................... 23
           Matching characters from a set .................................................................................. 23
       Redirections and pipes ................................................................................................. 23
           Redirection .................................................................................................................. 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>24</td>
</tr>
<tr>
<td>The basics</td>
<td>25</td>
</tr>
<tr>
<td>Permissions</td>
<td>27</td>
</tr>
<tr>
<td>Archives</td>
<td>29</td>
</tr>
<tr>
<td>Extracting archives</td>
<td>29</td>
</tr>
<tr>
<td>Creating archives</td>
<td>30</td>
</tr>
<tr>
<td>Extended attributes</td>
<td>30</td>
</tr>
<tr>
<td>Installing the necessary utilities</td>
<td>31</td>
</tr>
<tr>
<td>Showing extended attributes</td>
<td>31</td>
</tr>
<tr>
<td>Setting extended attributes</td>
<td>31</td>
</tr>
<tr>
<td>Mounting filesystems</td>
<td>32</td>
</tr>
<tr>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td>mount</td>
<td>32</td>
</tr>
<tr>
<td>umount</td>
<td>32</td>
</tr>
<tr>
<td>The fstab file</td>
<td>33</td>
</tr>
<tr>
<td>fs_spec</td>
<td>33</td>
</tr>
<tr>
<td>fs_file</td>
<td>33</td>
</tr>
<tr>
<td>fs_vftype</td>
<td>33</td>
</tr>
<tr>
<td>fs_mntops</td>
<td>33</td>
</tr>
<tr>
<td>fs_freq</td>
<td>34</td>
</tr>
<tr>
<td>fs_passno</td>
<td>34</td>
</tr>
<tr>
<td>8. Text Utilities</td>
<td>35</td>
</tr>
<tr>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>The basics</td>
<td>35</td>
</tr>
<tr>
<td>cat</td>
<td>35</td>
</tr>
<tr>
<td>echo</td>
<td>36</td>
</tr>
<tr>
<td>wc</td>
<td>36</td>
</tr>
<tr>
<td>tr</td>
<td>36</td>
</tr>
<tr>
<td>sort</td>
<td>37</td>
</tr>
<tr>
<td>uniq</td>
<td>38</td>
</tr>
<tr>
<td>9. Process management</td>
<td>39</td>
</tr>
<tr>
<td>Introduction</td>
<td>39</td>
</tr>
<tr>
<td>Process basics</td>
<td>39</td>
</tr>
<tr>
<td>ps</td>
<td>39</td>
</tr>
<tr>
<td>kill</td>
<td>40</td>
</tr>
<tr>
<td>Advanced process management</td>
<td>40</td>
</tr>
<tr>
<td>Background processes</td>
<td>41</td>
</tr>
<tr>
<td>Stopping processes</td>
<td>41</td>
</tr>
<tr>
<td>Altering priorities</td>
<td>41</td>
</tr>
<tr>
<td>III. System administration</td>
<td>43</td>
</tr>
<tr>
<td>10. User management</td>
<td>44</td>
</tr>
<tr>
<td>Introduction</td>
<td>44</td>
</tr>
<tr>
<td>Adding and removing users</td>
<td>44</td>
</tr>
<tr>
<td>useradd</td>
<td>44</td>
</tr>
<tr>
<td>passwd</td>
<td>45</td>
</tr>
<tr>
<td>adduser</td>
<td>45</td>
</tr>
<tr>
<td>userdel</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>46</td>
</tr>
</tbody>
</table>
Slackware Linux IPv6 support ......................................................... 71
Adding an IPv6 address to an interface ......................................... 71
Resolving ...................................................................................... 71
IPv4 Forwarding ........................................................................... 73

17. IPsec ....................................................................................... 75
   Theory ...................................................................................... 75
   Kernel configuration ................................................................... 75
   Installing IPsec-Tools ............................................................... 75
   The configuration of ipsec ......................................................... 76
      Introduction ........................................................................... 76
      Writing the configuration file ............................................... 76
      Activating the IPsec configuration ......................................... 77

18. The internet super server .......................................................... 79
   Introduction .............................................................................. 79
   Configuration ........................................................................... 79
   TCP wrappers ........................................................................... 79

19. Apache ..................................................................................... 81
   Introduction .............................................................................. 81
   Installation ............................................................................... 81
   User directories ........................................................................ 81
   Virtual hosts ............................................................................. 81

20. BIND ....................................................................................... 83
   Introduction .............................................................................. 83
      Delegation ............................................................................. 83
      DNS records ........................................................................... 83
      Masters and slaves ............................................................... 83
      Making a caching nameserver ............................................... 83
List of Tables

6-1. Bash wildcards .................................................................................................................. 22
7-1. Archive file extensions ....................................................................................................... 29
8-1. Special tr character sequences ............................................................................................. 36
16-1. Important IPv6 Prefixes .................................................................................................... 71
20-1. DNS records ....................................................................................................................... 83

List of Figures

4-1. Forking a process .................................................................................................................. 6
4-2. The filesystem structure ..................................................................................................... 7
5-1. The cfdisk partition tool ...................................................................................................... 11
5-2. The setup tool ..................................................................................................................... 12
5-3. Setting up the swap partition .................................................................................................. 13
5-4. Selecting a partition to initialize ........................................................................................... 13
5-5. Formatting the partition ....................................................................................................... 14
5-6. Selecting a filesystem type .................................................................................................... 14
5-7. Selecting the source medium ............................................................................................... 15
5-8. Selecting the disk sets .......................................................................................................... 15
5-9. Installing the kernel ............................................................................................................. 16
5-10. Installing the kernel ............................................................................................................ 16
5-11. Selecting the default modem ............................................................................................. 17
5-12. Enabling hotplugging ........................................................................................................ 17
5-13. Selecting the kind of LILO installation ............................................................................. 18
5-14. Choosing the framebuffer resolution .................................................................................. 18
6-1. Standard input and output ................................................................................................... 23
6-2. A pipeline ............................................................................................................................ 24
16-1. The anatomy of an IPv6 address ......................................................................................... 70
16-2. Router example ................................................................................................................. 73
Preface

This book aims to provide an introduction to Slackware Linux. It addresses people who have little or no GNU/Linux experience. It aims to cover the Slackware Linux installation, basic GNU/Linux commands and the configuration of Slackware Linux. As you can see the book is still work in progress, but the first bits are released in the "release early, release often" spirit.

This book was written by Daniël de Kok and is freely available under a BSDL-like license. It is continually under development, not just to keep up with the latest Slackware Linux versions, but also to refine the documentation, and extend it where it is deemed necessary.

I wish everybody a good time with Slackware Linux, and I hope this book is useful for you.
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://daniel.taickim.net/slackware-basics/.

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 operating 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 somewhat harsh, but it provides more transparency 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 principle 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 instead of a script for each 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.
Slackware Linux 10.0 features

- **Linux 2.4.26** - Slackware Linux uses the proven 2.4 Linux kernel as the default kernel. Slackware Linux 10.0 provides Linux 2.6.7 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.

- **X11R6.7.0** - This is the first Slackware Linux to provide the X Window System provided by the X.org Foundation. X11R6.7.0 is based on XFree86 4.4.0RC2, and is expanded with additional hardware support, enhancements, and bug fixes.

- **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) 3.2.3** - 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.

- **Improved wireless support** - Slackware Linux 10.0 features a new initialization script for WLAN network cards. These cards can now be configured through `/etc/rc.d/rc.wireless`.

**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/](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 statistics, masquerade connections, and multicast memberships
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 conventions), e.g. man(7), groff(7)
8 System administration commands (usually only for root)
9 Kernel routines [Non standard]
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 9).

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.
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 7.

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:
Chapter 4. General concepts

- /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.
- 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 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. 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.

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).
Chapter 5. Installing Slackware Linux

Figure 5-1. The cfdisk partition tool

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.
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”.

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.
Chapter 5. Installing Slackware Linux

Figure 5-4. Selecting a partition to initialize

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

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.
Chapter 5. Installing Slackware Linux

Figure 5-6. Selecting a filesystem type

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

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.
Chapter 5. Installing Slackware Linux

Figure 5-8. Selecting the disk sets

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

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.
The following dialog can be used to make a link, /dev/modem, that points to your modem device. If you do not have a modem you can select no modem.

The next step is to select whether you would like to use hotplug. 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.
Chapter 5. Installing Slackware Linux

Figure 5-12. Enabling hotplugging

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

After selecting the simple option the LILO configuration utility will ask you whether you would like to use a framebuffer or not. 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.
After setting the framebuffer you can pass extra parameters to the kernel. This is normally not necessary, if you do not want to pass extra parameters you can just press the <Enter> key.

The last step of the LILO configuration is selecting where LILO should be installed. 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.
II. GNU/Linux Basics
Chapter 6. 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@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 practice 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 6-1. Bash wildcards

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>A string of characters</td>
</tr>
<tr>
<td>?</td>
<td>A single character</td>
</tr>
<tr>
<td>[]</td>
<td>A character in an array of characters</td>
</tr>
</tbody>
</table>

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.
Chapter 6. The Bash shell

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 6-1

**Figure 6-1. 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 6-2 shows a schematic overview of a pipeline consisting of two commands.

Figure 6-2. A pipeline

```
stdin  Command 1  Command 2  stdout
```

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 7. 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 to 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
Chapter 7. Files and directories

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

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

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

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

### mkdir

As you might have guessed, the `mkdir` command can be used to create directories. For example:

```bash
$ 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:

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

### cp

Files can be copied with the `cp` 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:

```bash
$ 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:

```bash
$ 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:

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

The `mv` 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` command is used to remove files and directories. Let’s look at a simple example:

```
$ rm hello.c
```

This command 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 be 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 dot 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 notes
-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 momo
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:

```bash
$ 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 7-1 lists frequently used archive extensions, and what they mean.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.tar</td>
<td>An uncompressed tar archive</td>
</tr>
<tr>
<td>.tar.gz</td>
<td>A tar archive compressed with gzip</td>
</tr>
<tr>
<td>.tgz</td>
<td>A tar archive compressed with gzip</td>
</tr>
<tr>
<td>.tar.bz2</td>
<td>A tar archive compressed with bzip2</td>
</tr>
<tr>
<td>.tbz</td>
<td>A tar archive compressed with bzip2</td>
</tr>
</tbody>
</table>

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:

```bash
$ 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 6. Let’s see what happens if we redirect our tar archive to tar:

```bash
$ 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 directory tree

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/
  - Test directory contents

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 directory tree

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
```
Chapter 7. Files and directories

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:

```bash
$ 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:

```bash
$ 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:

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

The value of the attribute can be set using the -v (value) parameter:
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
Chapter 7. Files and directories

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).
Chapter 7. Files and directories

- **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.
Chapter 8. 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 6 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 possible 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 8-1 describes some of the frequently used special character sequences.
Table 8-1. Special tr character sequences

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\</code></td>
<td>backslash ()</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>new line</td>
</tr>
<tr>
<td><code>char1-char2</code></td>
<td>All characters from <code>char1</code> to <code>char2</code> (e.g. “a-z”)</td>
</tr>
<tr>
<td><code>[:alnum:]</code></td>
<td>All alphanumeric characters</td>
</tr>
<tr>
<td><code>[:alpha:]</code></td>
<td>All letters in the alphabet</td>
</tr>
<tr>
<td><code>[:punct:]</code></td>
<td>Punctuation characters</td>
</tr>
</tbody>
</table>

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:

```bash
$ 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:

```bash
$ 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:

```bash
$ 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:

```bash
$ 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 occurrences 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 a 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_sorted
Chapter 9. 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 /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
```
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) SIGWIN 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 \texttt{-n} parameter is used to specify the priority value. In this case the \texttt{/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:

\begin{verbatim}
$ nice -n -1 nice
nice: cannot set priority: Permission denied
\end{verbatim}

But it will work as the \texttt{root} user:

\begin{verbatim}
$ su -c "nice -n -1 nice"
Password: 
-1
\end{verbatim}

The \texttt{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 \texttt{renice} command, as is shown in the following example:

\begin{verbatim}
$ renice +5 5811
5811: old priority 0, new priority 5
\end{verbatim}

As with \texttt{nice} a non-\texttt{root} user cannot set negative priority values:

\begin{verbatim}
$ renice -5 5811
renice: 5811: setpriority: Permission denied
\end{verbatim}

\textbf{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 \texttt{nice} is derived from being nice to the other users on the system ;^).
III. System administration
Chapter 10. 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 \texttt{useradd} only adds users, it does not set a password for the added user. Passwords can be set using the \texttt{passwd} command.

\textbf{passwd}

As you probably guessed the \texttt{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:

\begin{verbatim}
$ passwd
Changing password for bob
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
\end{verbatim}

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

\begin{verbatim}
# passwd bob
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
\end{verbatim}

\textbf{adduser}

The \texttt{adduser} command combines \texttt{useradd} and \texttt{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.

\begin{verbatim}
# 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
\end{verbatim}
Chapter 10. User management

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.

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)
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

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.

## 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:
Chapter 10. User management

/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 `quotamon` command:

```
# mount -o remount /home
# quotamon -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 file system 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`. 
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”.


Chapter 11. Printer configuration

- **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.
Chapter 12. 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 consume 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 13. 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-10.0/
```

This will use the Slackware Linux 10.0 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:
Chapter 13. Package Management

```
# gpg --keyserver pgp.mit.edu --search security@slackware.com

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>" imported

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
```

Public key fingerprint = EC56 49DA 401E 22AB FA67 36EF 6A44 63C0 4010 2233
Subkey fingerprint = 5E523569 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 10.0) 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.
Chapter 14. 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 10.0, 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:

```bash
# cp /mnt/cdrom/kernels/bare.i/config /usr/src/linux/.config
```
Chapter 14. Building a kernel

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:

```bash
# 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:

```bash
# 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:

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

```bash
# 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:

```bash
$ lilo
Added Slack *
Added SlackOld
```

The next time you boot both entries will be available, and the “Slack” entry will be booted by default.
Chapter 15. 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.

E-Mail security

Introduction

GNU/Linux supports two major methods of securing e-mails, but before we will look into these specifically, we are going to look what e-mail security provides. There are two ways in which you can secure e-mails: signing e-mail and encrypting e-mail. Signing e-mail means that a special digital signature is added to the e-mail by the mail user agent. The recipient can use the signature to check whether an e-mail is really from the sender it claims to be from or not, and that the message is not in any way changed during the transmission. E-Mail encryption codes the e-mail in a way that only the intended recipient can decode it.

This system relies on two keys: the private and the public key. Public keys are used to encrypt messages, and messages can only be decrypted with the private key. This means that one can send his public key out to people. People can use this key to send encrypted e-mails, 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.

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?

GnuPG or S/MIME

At the moment, there are two major standards that are supported by GNU/Linux, the OpenPGP standard, by the GnuPG tool, and S/MIME. Both are about equally good concerning the strength of the used encryption algorithms. The big difference is that S/MIME relies on a certificate authority (CA), GnuPG does not. A certificate authority is an organization that is authorized to hand out keys. This
Chapter 15. Security

means that the certificate authority validates that the e-mail address and/or name in the certificate is authentic. This means that either:

- The certificate (aka private and public key) belongs to the owner of the e-mail address. E-Mail verification is used by the certificate authority to check this.
- The certificate belongs to the owner of the e-mail address, and to the name in the certificate. This involves an extra step than e-mail verification. To have a certificate with a name, your identity has to be checked by the certificate authority.

As you can see this system is quite bullet-proof, and you can reasonably expect that a certificate is authentically representing the e-mail address and/or person. GnuPG does not use a certificate authority. This means that anybody can make a key with your name and e-mail address. Trust is established by signing keys. For example, person A verifies that person B’s key really belongs to person B, by meeting him in real life, and he signs his key. The authenticity of a key is more certain if it is signed by people who you trust. As you can guess this system is not completely bullet proof, and it can take some time to get your key trusted by other people. Besides, if you use another e-mail address, and want to use a new key you will have to start all over again. The advantage of this approach is that your private key is not known to a certificate authority.

GnuPG used to be supported a lot better on GNU/Linux, but more and more e-mail clients support S/MIME these days. On Windows S/MIME is better supported, because Outlook Express and Mozilla Mail (two commonly used e-mail clients) both have S/MIME support. If you are communicating with many people that use Windows, it is a good choice to go for S/MIME, because it is worthless to use signed e-mails if the recipient can not verify them by default.

Configuring and using GnuPG

This section gives a short introduction on how to configure and use GnuPG. Make sure you have GnuPG installed. If you have not, you can install it 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 describes 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 been shown that 512-bit RSA keys can be broken in a relatively short time using common hardware. More information about these issues can be 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 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:

pub 2048R/8D080768 2004-07-16 John Doe (Work) <john@doe.com>

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:
Chapter 15. Security

$ 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 \texttt{--armor} parameter:

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

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

\section*{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 \texttt{--detach-sign} parameter. Let us look at an example. This command will make a signature for the \texttt{memo.txt} file. The signature will be stored in \texttt{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 (\texttt{memo.txt.sig}) will be created.

You can verify a file with its signature using the \texttt{--verify} parameter. Specify the signature file as a parameter to the \texttt{--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 \textit{John Doe (Work) <john@doe.com>}, with the key \texttt{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>"
Chapter 15. Security

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:
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
```
IV. Network administration
Chapter 16. 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

69
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_DHCP[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 new settings by rebooting the system or by executing `/etc/rc.d/rc.inet1`.

### 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, the 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 16-1. The anatomy of an IPv6 address](image)
There are a some specially reserved prefixes, most notable include:

Table 16-1. Important IPv6 Prefixes

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fe80::</td>
<td>Link local addresses, which are not routed.</td>
</tr>
<tr>
<td>fec0::</td>
<td>Site local addresses, which are locally routed, but not on or to the internet.</td>
</tr>
<tr>
<td>2002::</td>
<td>6to4 addresses, which are used for the transition from IPv4 to IPv6.</td>
</tr>
</tbody>
</table>

### 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
```
Chapter 16. Networking configuration

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
192.168.1.2 gideon.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 gideon.slackfans.org is assigned to the IP address 192.168.1.2. 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
192.168.1.2 gideon.slackfans.org

# IPv6 entries
::1 localhost
fec0::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 192.168.1.69
```
Chapter 16. Networking configuration

You can check whether the hostnames are translated 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 16-2. Router example

![Router example diagram]

In this example there are two networks, 192.168.1.0 and 192.168.2.0. Three hosts are connected to both networks. 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 vice 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 `IPV4_FORWARD` variable in `/etc/rc.d/rc.inet2`. The default setting is as follows:

```
IPV4_FORWARD=1
```

This means that IPv4 forwarding is enabled. You can disable forwarding by changing the `IPV4_FORWARD` value to 0. This setting can be enabled by rebooting the computer. 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 administered 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 17. 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.

Kernel configuration

This chapter only applies to Linux 2.6.x kernels. Earlier kernels have no native IPsec support. So, make sure that you have a 2.6.x kernel. Configure the kernel as you would do normally, and make sure the following options are enabled:

Device Drivers -> Networking support -> Networking options:

- <*> PF_Key sockets
- <*> IP: AH transformation
- <*> IP: ESP transformation
- <*> IP: IPComp transformation

In this article 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.

Compile the kernel as usual and boot it.

Installing IPsec-Tools

The next step is to install the IPsec-Tools. These tools are ports of the KAME (http://www.kame.net) IPsec utilities. Download the latest sources and unpack, configure and install them:

```
# tar zxf ipsec-tools-x.y.z.tar.gz
```
# cd ipsec-tools-x.y.z
# ./configure --prefix=/usr --sysconfdir=/etc
# make
# make install

Replace x.y.z with the version of the downloaded sources, for example 0.2.3.

## The configuration of ipsec

### Introduction

We will use an example as the guideline for setting up an encrypted connection between two 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/ipsec.conf`. On the first host (192.168.1.1) the following `/etc/ipsec.conf` configuration will be used:

```bash
#!/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
0xa731649644c5dee92cb9c27e188ee6;
add 192.168.1.1 192.168.1.169 ah 0x300 -A hmac-md5
0x27f6d123d707b361662fc6e451f65d8;
add 192.168.1.1 192.168.1.169 esp 0x201 -E 3des-cbc
0x666c5323255c323a066c197aa0cf30991fca83532a4b224;
add 192.168.1.1 192.168.1.169 esp 0x301 -E 3des-cbc
0xc966199f24d0955f3990a320d749056401e82b2657320292;

spdadd 192.168.1.1 192.168.1.169 any -P out ipsec
   esp/transport//require
   ah/transport//require;

spdadd 192.168.1.1 192.168.1.169 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 “spaddd” 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 `/etc/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/ipsec.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/ipsec.conf
```

The same `/etc/ipsec.conf` can be created on the 192.168.1.169 host, with inverted `-P in` and `-P out` options. So, the `/etc/ipsec.conf` will look like this:

```
#!/usr/sbin/setkey -f

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
    0xc966199f24d095f3990a320d749056401e82b26570320292;
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 192.168.1.1 192.168.1.169 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/ipsec.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/ipsec.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 tazzy.slackfiles.org > 192.168.1.169: AH(spi=0x00000200,seq=0x40f): ESP(spi=0x00000201,seq=0x40f) (DF)
11:29:58.870786 192.168.1.169 > tazzy.slackfiles.org: AH(spi=0x00000300,seq=0x33d7): ESP
```

Chapter 18. 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
```

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 19. 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 package 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:

```bash
# 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:

```bash
# 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 separate 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 20. 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 authoritative 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>
<thead>
<tr>
<th>Prefix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>An A records points to an IP address.</td>
</tr>
<tr>
<td>CNAME</td>
<td>A CNAME record points to another DNS entry.</td>
</tr>
<tr>
<td>MX</td>
<td>A MX record specifies which should handle mail for the domain.</td>
</tr>
</tbody>
</table>

Masters and slaves

Two kinds of nameservers can be provided for a domain name: a master and slaves. The master server DNS records are authoritative. 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.