Introduction to Linux

A Hands on Guide

Machtelt Garrels

Garrels.be

<tille_wants_no_spam_at_garrels_dot_be>

1.27 Edition

20080606

Table of Contents

Introduction **1**
1. Why this guide? 1
2. Who should read this book? 1
3. New versions and availability 1
4. Revision History 2
5. Contributions 3
6. Feedback 3
7. Copyright information 3
8. What do you need? 4
9. Conventions used in this document 4
10. Organization of this document 5

Chapter 1. What is Linux? **7**
1.1. History 7
1.1.1. UNIX 7
1.1.2. Linus and Linux 8
1.1.3. Current application of Linux systems 9
1.2. The user interface 9
1.2.1. Is Linux difficult? 9
1.2.2. Linux for non-experienced users 10
1.3. Does Linux have a future? 10
1.3.1. Open Source 10
1.3.2. Ten years of experience at your service 11
1.4. Properties of Linux 12
1.4.1. Linux Pros 12
1.4.2. Linux Cons 13
1.5. Linux Flavors 14
1.5.1. Linux and GNU 14
1.5.2. GNU/Linux 15
1.5.3. Which distribution should I install? 15
1.6. Summary 16
1.7. Exercises 16

Chapter 2. Quickstart **18**
2.1. Logging in, activating the user interface and logging out 18
2.1.1. Introduction 18
2.1.2. Graphical mode 18
2.1.3. Text mode 20
2.2. Absolute basics 21
2.2.1. The commands 21
2.2.2. General remarks 21
2.2.3. Using Bash features 22
2.3. Getting help 23
2.3.1. Be warned 23
2.3.2. The man pages 23
2.3.3. More info 25
2.4. Summary 28
2.5. Exercises 29

Table of Contents

Chapter 2. Quickstart
2.5.1. Connecting and disconnecting....29
2.5.2. Passwords....29
2.5.3. Directories....30
2.5.4. Files....30
2.5.5. Getting help....31

Chapter 3. About files and the file system....32
3.1. General overview of the Linux file system....32
3.1.1. Files....32
3.1.2. About partitioning....33
3.1.3. More file system layout....37
3.2. Orientation in the file system....40
3.2.1. The path....40
3.2.2. Absolute and relative paths....41
3.2.3. The most important files and directories....41
3.2.4. The most important configuration files....44
3.2.5. The most common devices....46
3.2.6. The most common variable files....47
3.3. Manipulating files....48
3.3.1. Viewing file properties....48
3.3.2. Creating and deleting files and directories....50
3.3.3. Finding files....53
3.3.4. More ways to view file content....57
3.3.5. Linking files....58
3.4. File security....60
3.4.1. Access rights: Linux's first line of defense....60
3.4.2. The tools....62
3.5. Summary....67
3.6. Exercises....68
3.6.1. Partitions....68
3.6.2. Paths....68
3.6.3. Tour of the system....69
3.6.4. Manipulating files....69
3.6.5. File permissions....69

Chapter 4. Processes....71
4.1. Processes inside out....71
4.1.1. Multi-user and multi-tasking....71
4.1.2. Process types....71
4.1.3. Process attributes....73
4.1.4. Displaying process information....74
4.1.5. Life and death of a process....76
4.1.6. SUID and SGID....78
4.2. Boot process, Init and shutdown....80
4.2.1. Introduction....80
4.2.2. The boot process....80
4.2.3. GRUB features....80

Table of Contents

Chapter 4. Processes
4.2.4. Init....81
4.2.5. Init run levels....83
4.2.6. Shutdown....84
4.3. Managing processes....84
4.3.1. Work for the system admin....84
4.3.2. How long does it take?....85
4.3.3. Performance....86
4.3.4. Load....86
4.3.5. Can I do anything as a user?....86
4.4. Scheduling processes....91
4.4.1. Use that idle time!....91
4.4.2. The sleep command....91
4.4.3. The at command....92
4.4.4. Cron and crontab....92
4.5. Summary....94
4.6. Exercises....95
4.6.1. General....95
4.6.2. Booting, init etc.....95
4.6.3. Scheduling....96

Chapter 5. I/O redirection....97
5.1. Simple redirections....97
5.1.1. What are standard input and standard output?....97
5.1.2. The redirection operators....97
5.2. Advanced redirection features....100
5.2.1. Use of file descriptors....100
5.2.2. Examples....101
5.3. Filters....101
5.3.1. More about grep....102
5.3.2. Filtering output....102
5.4. Summary....103
5.5. Exercises....103

Chapter 6. Text editors....105
6.1. Text editors....105
6.1.1. Why should I use an editor?....105
6.1.2. Which editor should I use?....105
6.2. Using the Vim editor....106
6.2.1. Two modes....106
6.2.2. Basic commands....107
6.2.3. The easy way....108
6.3. Linux in the office....108
6.3.1. History....108
6.3.2. Suites and programs....108
6.3.3. Remarks....109
6.4. Summary....109
6.5. Exercises....110

Table of Contents

Chapter 7. Home sweet /home **111**
7.1. General good housekeeping 111
7.1.1. Introduction 111
7.1.2. Make space 111
7.2. Your text environment 114
7.2.1. Environment variables 114
7.2.2. Shell setup files 116
7.2.3. A typical set of setup files 117
7.2.4. The Bash prompt 120
7.2.5. Shell scripts 121
7.3. The graphical environment 123
7.3.1. Introduction 123
7.3.2. The X Window System 124
7.3.3. X server configuration 125
7.4. Region specific settings 126
7.4.1. Keyboard setup 126
7.4.2. Fonts 126
7.4.3. Date and time zone 127
7.4.4. Language 127
7.4.5. Country-specific Information 128
7.5. Installing new software 128
7.5.1. General 128
7.5.2. Package formats 128
7.5.3. Automating package management and updates 131
7.5.4. Upgrading your kernel 132
7.5.5. Installing extra packages from the installation CDs 133
7.6. Summary 134
7.7. Exercises 135
7.7.1. Shell environment 135
7.7.2. Graphical environment 136

Chapter 8. Printers and printing **137**
8.1. Printing files 137
8.1.1. Command line printing 137
8.1.2. Formatting 138
8.2. The server side 139
8.2.1. General 139
8.2.2. Graphical printer configuration 140
8.2.3. Buying a printer for Linux 140
8.3. Print problems 140
8.3.1. Wrong file 140
8.3.2. My print hasn't come out 140
8.4. Summary 142
8.5. Exercises 142

Chapter 9. Fundamental Backup Techniques **144**
9.1. Introduction 144
9.1.1. Preparing your data 144

Table of Contents

Chapter 9. Fundamental Backup Techniques
9.2. Moving your data to a backup device....148
9.2.1. Making a copy on a floppy disk....148
9.2.2. Making a copy with a CD-writer....150
9.2.3. Backups on/from jazz drives, USB devices and other removables....151
9.2.4. Backing up data using a tape device....151
9.2.5. Tools from your distribution....151
9.3. Using rsync....152
9.3.1. Introduction....152
9.3.2. An example: rsync to a USB storage device....152
9.4. Encryption....152
9.4.1. General remarks....152
9.4.2. Generate a key....153
9.4.3. About your key....154
9.4.4. Encrypt data....154
9.4.5. Decrypting files....155
9.5. Summary....155
9.6. Exercises....156

Chapter 10. Networking....157
10.1. Networking Overview....157
10.1.1. The OSI Model....157
10.1.2. Some popular networking protocols....158
10.2. Network configuration and information....160
10.2.1. Configuration of network interfaces....160
10.2.2. Network configuration files....161
10.2.3. Network configuration commands....161
10.2.4. Network interface names....163
10.2.5. Checking the host configuration with netstat....164
10.2.6. Other hosts....164
10.3. Internet/Intranet applications....167
10.3.1. Server types....167
10.3.2. Mail....168
10.3.3. Web....170
10.3.4. File Transfer Protocol....171
10.3.5. Chatting and conferencing....172
10.3.6. News services....173
10.3.7. The Domain Name System....174
10.3.8. DHCP....174
10.3.9. Authentication services....174
10.4. Remote execution of applications....176
10.4.1. Introduction....176
10.4.2. Rsh, rlogin and telnet....176
10.4.3. The X Window System....177
10.4.4. The SSH suite....178
10.4.5. VNC....182
10.4.6. The rdesktop protocol....182
10.4.7. Cygwin....182

Table of Contents

Chapter 10. Networking
10.5. Security....183
10.5.1. Introduction....183
10.5.2. Services....183
10.5.3. Update regularly....184
10.5.4. Firewalls and access policies....184
10.5.5. Intrusion detection....185
10.5.6. More tips....186
10.5.7. Have I been hacked?....186
10.5.8. Recovering from intrusion....187
10.6. Summary....187
10.7. Exercises....188
10.7.1. General networking....188
10.7.2. Remote connections....188
10.7.3. Security....188

Chapter 11. Sound and Video....189
11.1. Audio Basics....189
11.1.1. Installation....189
11.1.2. Drivers and Architecture....189
11.2. Sound and video playing....190
11.2.1. CD playing and copying....190
11.2.2. Playing music files....190
11.2.3. Recording....192
11.3. Video playing, streams and television watching....192
11.4. Internet Telephony....193
11.4.1. What is it?....193
11.4.2. What do you need?....193
11.5. Summary....194
11.6. Exercises....195

Appendix A. Where to go from here?....196
A.1. Useful Books....196
A.1.1. General Linux....196
A.1.2. Editors....196
A.1.3. Shells....196
A.1.4. X Window....196
A.1.5. Networking....197
A.2. Useful sites....197
A.2.1. General information....197
A.2.2. Architecture Specific References....197
A.2.3. Distributions....197
A.2.4. Software....198

Appendix B. DOS versus Linux commands....199

Table of Contents

Appendix C. Shell Features **200**
C.1. Common features 200
C.2. Differing features 201

Glossary **204**
A 204
B 204
C 205
D 205
E 206
F 206
G 207
H 207
I 207
J 208
K 208
L 208
M 209
N 210
O 210
P 210
Q 211
R 211
S 212
T 212
U 213
V 214
W 214
X 214
Y 215
Z 215
Index 215

Introduction

1. Why this guide?

Many people still believe that learning Linux is difficult, or that only experts can understand how a Linux system works. Though there is a lot of free documentation available, the documentation is widely scattered on the Web, and often confusing, since it is usually oriented toward experienced UNIX or Linux users. Today, thanks to the advancements in development, Linux has grown in popularity both at home and at work. The goal of this guide is to show people of all ages that Linux can be simple and fun, and used for all kinds of purposes.

2. Who should read this book?

This guide was created as an overview of the Linux Operating System, geared toward new users as an exploration tour and getting started guide, with exercises at the end of each chapter. For more advanced trainees it can be a desktop reference, and a collection of the base knowledge needed to proceed with system and network administration. This book contains many real life examples derived from the author's experience as a Linux system and network administrator, trainer and consultant. We hope these examples will help you to get a better understanding of the Linux system and that you feel encouraged to try out things on your own.

Everybody who wants to get a "CLUE", a Command Line User Experience, with Linux (and UNIX in general) will find this book useful.

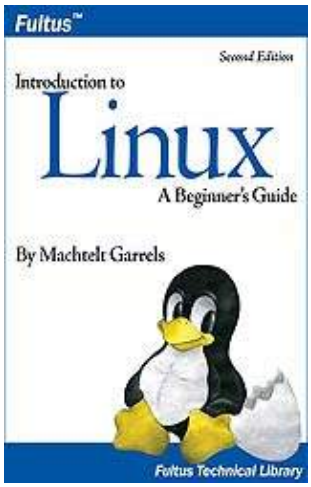
3. New versions and availability

This document is published in the Guides section of the Linux Documentation Project collection at http://www.tldp.org/guides.html; you can also download PDF and PostScript formatted versions here.

The most recent edition is available at http://tille.garrels.be/training/tldp/.

The second edition of this guide is available in print from Fultus.com Books as paperback Print On Demand (POD) book. Fultus distributes this document through Ingram and Baker & Taylor to many bookstores, including Amazon.com, Amazon.co.uk, BarnesAndNoble.com and Google's Froogle global shopping portal and Google Book Search.

Figure 1. Introduction to Linux front cover



The guide has been translated into Hindi by:

- Alok Kumar
- Dhananjay Sharma
- Kapil
- Puneet Goel
- Ravikant Yuyutsu

Andrea Montagner translated the guide into Italian.

4. Revision History

Revision History

Revision 1.27	20080606	Revised by: MG
updates.		
Revision 1.26	20070919	Revised by: MG
Comments from readers, license.		
Revision 1.25	20070511	Revised by: MG
Comments from readers, minor updates, E-mail etiquette, updated info about availability (thanks Oleg).		
Revision 1.24	2006-11-01	Revised by: MG
added index terms, prepared for second printed edition, added gpg and proxy info.		
Revision 1.23	2006-07-25	Revised by: MG and FK
Updates and corrections, removed app5 again, adapted license to enable inclusion in Debian docs.		
Revision 1.22	2006-04-06	Revised by: MG
chap8 revised completely, chap10: clarified examples, added ifconfig and cygwin info, revised network apps.		
Revision 1.21	2006-03-14	Revised by: MG
Added exercises in chap11, corrected newline errors, command overview completed for chapter 9, minor corrections in chap10.		
Revision 1.20	2006-01-06	Revised by: MG
Split chap7: audio stuff is now in separate chapter, chap11.xml. Small revisions, updates for commands like aptitude, more on USB storage, Internet telephony, corrections from readers.		
Revision 1.13	2004-04-27	Revised by: MG

Last read-through before sending everything to Fultus for printout. Added Fultus referrence in New Versions section, updated Conventions and Organization sections. Minor changes in chapters 4, 5, 6 and 8, added rdesktop info in chapter 10, updated glossary, replaced references to fileutils with coreutils, thankyou to Hindi translators.

5. Contributions

Many thanks to all the people who shared their experiences. And especially to the Belgian Linux users for hearing me out every day and always being generous in their comments.

Also a special thought for Tabatha Marshall for doing a really thorough revision, spell check and styling, and to Eugene Crosser for spotting the errors that we two overlooked.

And thanks to all the readers who notified me about missing topics and who helped to pick out the last errors, unclear definitions and typos by going through the trouble of mailing me all their remarks. These are also the people who help me keep this guide up to date, like Filipus Klutiero who did a complete review in 2005 and 2006 and helps me getting the guide into the Debian docs collection, and Alexey Eremenko who sent me the foundation for chapter 11.

In 2006, Suresh Rajashekara created a Debian package of this documentation.

Finally, a big thank you for the volunteers who are currently translating this document in French, Swedish, German, Farsi, Hindi and more. It is a big work that should not be underestimated; I admire your courage.

6. Feedback

Missing information, missing links, missing characters? Mail it to the maintainer of this document:

<tille wants no spam at garrels dot be>

Don't forget to check with the latest version first!

7. Copyright information

```
* 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 AUTHOR AND 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
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```


8. What do you need?

You will require a computer and a medium containing a Linux distribution. Most of this guide applies to all Linux distributions - and UNIX in general. Apart from time, there are no further specific requirements.

The Installation HOWTO contains helpful information on how to obtain Linux software and install it on your computer. Hardware requirements and coexistence with other operating systems are also discussed.

CD images can be downloaded from linux-iso.com and many other locations, see Appendix A.

An interesting alternative for those who don't dare to take the step of an actual Linux installation on their machine are the Linux distributions that you can run from a CD, such as the Knoppix distribution.

9. Conventions used in this document

The following typographic and usage conventions occur in this text:

Table 1. Typographic and usage conventions

Text type	Meaning
"Quoted text"	Quotes from people, quoted computer output.
`terminal view`	Literal computer input and output captured from the terminal, usually rendered with a light grey background.
command	Name of a command that can be entered on the command line.
`VARIABLE`	Name of a variable or pointer to content of a variable, as in `$VARNAME`.
`option`	Option to a command, as in "the `-a` option to the **ls** command".
`argument`	Argument to a command, as in "read **man** ***`ls`*** ".
`prompt`	User prompt, usually followed by a command that you type in a terminal window, like in `hilda@home>` **`ls -l`**
command `options` ***`arguments`***	Command synopsis or general usage, on a separated line.
`filename`	Name of a file or directory, for example "Change to the `/usr/bin` directory."
Key	Keys to hit on the keyboard, such as "type **Q** to quit".
Button	Graphical button to click, like the OK button.
Menu->Choice	Choice to select from a graphical menu, for instance: "SelectHelp->About Mozilla in your browser."

Terminology	Important term or concept: "The Linux *kernel* is the heart of the system."
\	The backslash in a terminal view or command synopsis indicates an unfinished line. In other words, if you see a long command that is cut into multiple lines, \ means "Don't press **Enter** yet!"
See Chapter 1	link to related subject within this guide.
The author	Clickable link to an external web resource.

The following images are used:

This is a note

It contains additional information or remarks.

This is a caution

It means be careful.

This is a warning

Be *very* careful.

This is a tip

Tips and tricks.

10. Organization of this document

This guide is part of the Linux Documentation Project and aims to be the foundation for all other materials that you can get from the Project. As such, it provides you with the fundamental knowledge needed by anyone who wants to start working with a Linux system, while at the same time it tries to consciously avoid re-inventing the hot water. Thus, you can expect this book to be incomplete and full of links to sources of additional information on your system, on the Internet and in your system documentation.

The first chapter is an introduction to the subject on Linux; the next two discuss absolute basic commands. Chapters 4 and 5 discuss some more advanced but still basic topics. Chapter 6 is needed for continuing with the rest, since it discusses editing files, an ability you need to pass from Linux newbie to Linux user. The following chapters discuss somewhat more advanced topics that you will have to deal with in everyday Linux use.

All chapters come with exercises that will test your preparedness for the next chapter.

- Chapter 1: What is Linux, how did it come into existence, advantages and disadvantages, what does the future hold for Linux, who should use it, installing your computer.
- Chapter 2: Getting started, connecting to the system, basic commands, where to find help.
- Chapter 3: The filesystem, important files and directories, managing files and directories, protecting your data.
- Chapter 4: Understanding and managing processes, boot and shutdown procedures, postponing tasks, repetitive tasks.
- Chapter 5: What are standard input, output and error and how are these features used from the command line.
- Chapter 6: Why you should learn to work with an editor, discussion of the most common editors.

- Chapter 7: Configuring your graphical, text and audio environment, settings for the non-native English speaking Linux user, tips for adding extra software.
- Chapter 8: Converting files to a printable format, getting them out of the printer, hints for solving print problems.
- Chapter 9: Preparing data to be backed up, discussion of various tools, remote backup.
- Chapter 10: Overview of Linux networking tools and user applications, with a short discussion of the underlying service daemon programs and secure networking.
- Chapter 11: Sound and video, including Voice over IP and sound recording is discussed in this chapter.
- Appendix A: Which books to read and sites to visit when you have finished reading this one.
- Appendix B: A comparison.
- Appendix C: If you ever get stuck, these tables might be an outcome. Also a good argument when your boss insists that YOU should use HIS favorite shell.

Chapter 1. What is Linux?

We will start with an overview of how Linux became the operating system it is today. We will discuss past and future development and take a closer look at the advantages and disadvantages of this system. We will talk about distributions, about Open Source in general and try to explain a little something about GNU.

This chapter answers questions like:

- What is Linux?
- Where and how did Linux start?
- Isn't Linux that system where everything is done in text mode?
- Does Linux have a future or is it just hype?
- What are the advantages of using Linux?
- What are the disadvantages?
- What kinds of Linux are there and how do I choose the one that fits me?
- What are the Open Source and GNU movements?

1.1. History

1.1.1. UNIX

In order to understand the popularity of Linux, we need to travel back in time, about 30 years ago...

Imagine computers as big as houses, even stadiums. While the sizes of those computers posed substantial problems, there was one thing that made this even worse: every computer had a different operating system. Software was always customized to serve a specific purpose, and software for one given system didn't run on another system. Being able to work with one system didn't automatically mean that you could work with another. It was difficult, both for the users and the system administrators.

Computers were extremely expensive then, and sacrifices had to be made even after the original purchase just to get the users to understand how they worked. The total cost per unit of computing power was enormous.

Technologically the world was not quite that advanced, so they had to live with the size for another decade. In 1969, a team of developers in the Bell Labs laboratories started working on a solution for the software problem, to address these compatibility issues. They developed a new operating system, which was

1. Simple and elegant.
2. Written in the C programming language instead of in assembly code.
3. Able to recycle code.

The Bell Labs developers named their project "UNIX."

The code recycling features were very important. Until then, all commercially available computer systems were written in a code specifically developed for one system. UNIX on the other hand needed only a small piece of that special code, which is now commonly named the kernel. This kernel is the only piece of code that needs to be adapted for every specific system and forms the base of the UNIX system. The operating system and all other functions were built around this kernel and written in a higher programming language, C.

This language was especially developed for creating the UNIX system. Using this new technique, it was much easier to develop an operating system that could run on many different types of hardware.

The software vendors were quick to adapt, since they could sell ten times more software almost effortlessly. Weird new situations came in existence: imagine for instance computers from different vendors communicating in the same network, or users working on different systems without the need for extra education to use another computer. UNIX did a great deal to help users become compatible with different systems.

Throughout the next couple of decades the development of UNIX continued. More things became possible to do and more hardware and software vendors added support for UNIX to their products.

UNIX was initially found only in very large environments with mainframes and minicomputers (note that a PC is a "micro" computer). You had to work at a university, for the government or for large financial corporations in order to get your hands on a UNIX system.

But smaller computers were being developed, and by the end of the 80's, many people had home computers. By that time, there were several versions of UNIX available for the PC architecture, but none of them were truly free and more important: they were all terribly slow, so most people ran MS DOS or Windows 3.1 on their home PCs.

1.1.2. Linus and Linux

By the beginning of the 90s home PCs were finally powerful enough to run a full blown UNIX. Linus Torvalds, a young man studying computer science at the university of Helsinki, thought it would be a good idea to have some sort of freely available academic version of UNIX, and promptly started to code.

He started to ask questions, looking for answers and solutions that would help him get UNIX on his PC. Below is one of his first posts in comp.os.minix, dating from 1991:

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: Gcc-1.40 and a posix-question
Message-ID: <1991Jul3.100050.9886@klaava.Helsinki.FI>
Date: 3 Jul 91 10:00:50 GMT
Hello netlanders,
Due to a project I'm working on (in minix), I'm interested in the posix
standard definition. Could somebody please point me to a (preferably)
machine-readable format of the latest posix rules? Ftp-sites would be
nice.
```

From the start, it was Linus' goal to have a free system that was completely compliant with the original UNIX. That is why he asked for POSIX standards, POSIX still being the standard for UNIX.

In those days plug-and-play wasn't invented yet, but so many people were interested in having a UNIX system of their own, that this was only a small obstacle. New drivers became available for all kinds of new hardware, at a continuously rising speed. Almost as soon as a new piece of hardware became available, someone bought it and submitted it to the Linux test, as the system was gradually being called, releasing more free code for an ever wider range of hardware. These coders didn't stop at their PC's; every piece of hardware they could find was useful for Linux.

Back then, those people were called "nerds" or "freaks", but it didn't matter to them, as long as the supported hardware list grew longer and longer. Thanks to these people, Linux is now not only ideal to run on new PC's,

but is also the system of choice for old and exotic hardware that would be useless if Linux didn't exist.

Two years after Linus' post, there were 12000 Linux users. The project, popular with hobbyists, grew steadily, all the while staying within the bounds of the POSIX standard. All the features of UNIX were added over the next couple of years, resulting in the mature operating system Linux has become today. Linux is a full UNIX clone, fit for use on workstations as well as on middle-range and high-end servers. Today, a lot of the important players on the hard- and software market each have their team of Linux developers; at your local dealer's you can even buy pre-installed Linux systems with official support - eventhough there is still a lot of hard- and software that is not supported, too.

1.1.3. Current application of Linux systems

Today Linux has joined the desktop market. Linux developers concentrated on networking and services in the beginning, and office applications have been the last barrier to be taken down. We don't like to admit that Microsoft is ruling this market, so plenty of alternatives have been started over the last couple of years to make Linux an acceptable choice as a workstation, providing an easy user interface and MS compatible office applications like word processors, spreadsheets, presentations and the like.

On the server side, Linux is well-known as a stable and reliable platform, providing database and trading services for companies like Amazon, the well-known online bookshop, US Post Office, the German army and many others. Especially Internet providers and Internet service providers have grown fond of Linux as firewall, proxy- and web server, and you will find a Linux box within reach of every UNIX system administrator who appreciates a comfortable management station. Clusters of Linux machines are used in the creation of movies such as "Titanic", "Shrek" and others. In post offices, they are the nerve centers that route mail and in large search engine, clusters are used to perform internet searches.These are only a few of the thousands of heavy-duty jobs that Linux is performing day-to-day across the world.

It is also worth to note that modern Linux not only runs on workstations, mid- and high-end servers, but also on "gadgets" like PDA's, mobiles, a shipload of embedded applications and even on experimental wristwatches. This makes Linux the only operating system in the world covering such a wide range of hardware.

1.2. The user interface

1.2.1. Is Linux difficult?

Whether Linux is difficult to learn depends on the person you're asking. Experienced UNIX users will say no, because Linux is an ideal operating system for power-users and programmers, because it has been and is being developed by such people.

Everything a good programmer can wish for is available: compilers, libraries, development and debugging tools. These packages come with every standard Linux distribution. The C-compiler is included for free - as opposed to many UNIX distributions demanding licensing fees for this tool. All the documentation and manuals are there, and examples are often included to help you get started in no time. It feels like UNIX and switching between UNIX and Linux is a natural thing.

In the early days of Linux, being an expert was kind of required to start using the system. Those who mastered Linux felt better than the rest of the "lusers" who hadn't seen the light yet. It was common practice to tell a beginning user to "RTFM" (read the manuals). While the manuals were on every system, it was difficult to

find the documentation, and even if someone did, explanations were in such technical terms that the new user became easily discouraged from learning the system.

The Linux-using community started to realize that if Linux was ever to be an important player on the operating system market, there had to be some serious changes in the accessibility of the system.

1.2.2. Linux for non-experienced users

Companies such as RedHat, SuSE and Mandriva have sprung up, providing packaged Linux distributions suitable for mass consumption. They integrated a great deal of graphical user interfaces (GUIs), developed by the community, in order to ease management of programs and services. As a Linux user today you have all the means of getting to know your system inside out, but it is no longer necessary to have that knowledge in order to make the system comply to your requests.

Nowadays you can log in graphically and start all required applications without even having to type a single character, while you still have the ability to access the core of the system if needed. Because of its structure, Linux allows a user to grow into the system: it equally fits new and experienced users. New users are not forced to do difficult things, while experienced users are not forced to work in the same way they did when they first started learning Linux.

While development in the service area continues, great things are being done for desktop users, generally considered as the group least likely to know how a system works. Developers of desktop applications are making incredible efforts to make the most beautiful desktops you've ever seen, or to make your Linux machine look just like your former MS Windows or an Apple workstation. The latest developments also include 3D acceleration support and support for USB devices, single-click updates of system and packages, and so on. Linux has these, and tries to present all available services in a logical form that ordinary people can understand. Below is a short list containing some great examples; these sites have a lot of screenshots that will give you a glimpse of what Linux on the desktop can be like:

- http://www.gnome.org
- http://kde.org/screenshots/
- http://www.openoffice.org
- http://www.mozilla.org

1.3. Does Linux have a future?

1.3.1. Open Source

The idea behind Open Source software is rather simple: when programmers can read, distribute and change code, the code will mature. People can adapt it, fix it, debug it, and they can do it at a speed that dwarfs the performance of software developers at conventional companies. This software will be more flexible and of a better quality than software that has been developed using the conventional channels, because more people have tested it in more different conditions than the closed software developer ever can.

The Open Source initiative started to make this clear to the commercial world, and very slowly, commercial vendors are starting to see the point. While lots of academics and technical people have already been convinced for 20 years now that this is the way to go, commercial vendors needed applications like the Internet to make them realize they can profit from Open Source. Now Linux has grown past the stage where it was almost exclusively an academic system, useful only to a handful of people with a technical background.

Now Linux provides more than the operating system: there is an entire infrastructure supporting the chain of effort of creating an operating system, of making and testing programs for it, of bringing everything to the users, of supplying maintenance, updates and support and customizations, etcetera. Today, Linux is ready to accept the challenge of a fast-changing world.

1.3.2. Ten years of experience at your service

While Linux is probably the most well-known Open Source initiative, there is another project that contributed enormously to the popularity of the Linux operating system. This project is called SAMBA, and its achievement is the reverse engineering of the Server Message Block (SMB)/Common Internet File System (CIFS) protocol used for file- and print-serving on PC-related machines, natively supported by MS Windows NT and OS/2, and Linux. Packages are now available for almost every system and provide interconnection solutions in mixed environments using MS Windows protocols: Windows-compatible (up to and includingWinXP) file- and print-servers.

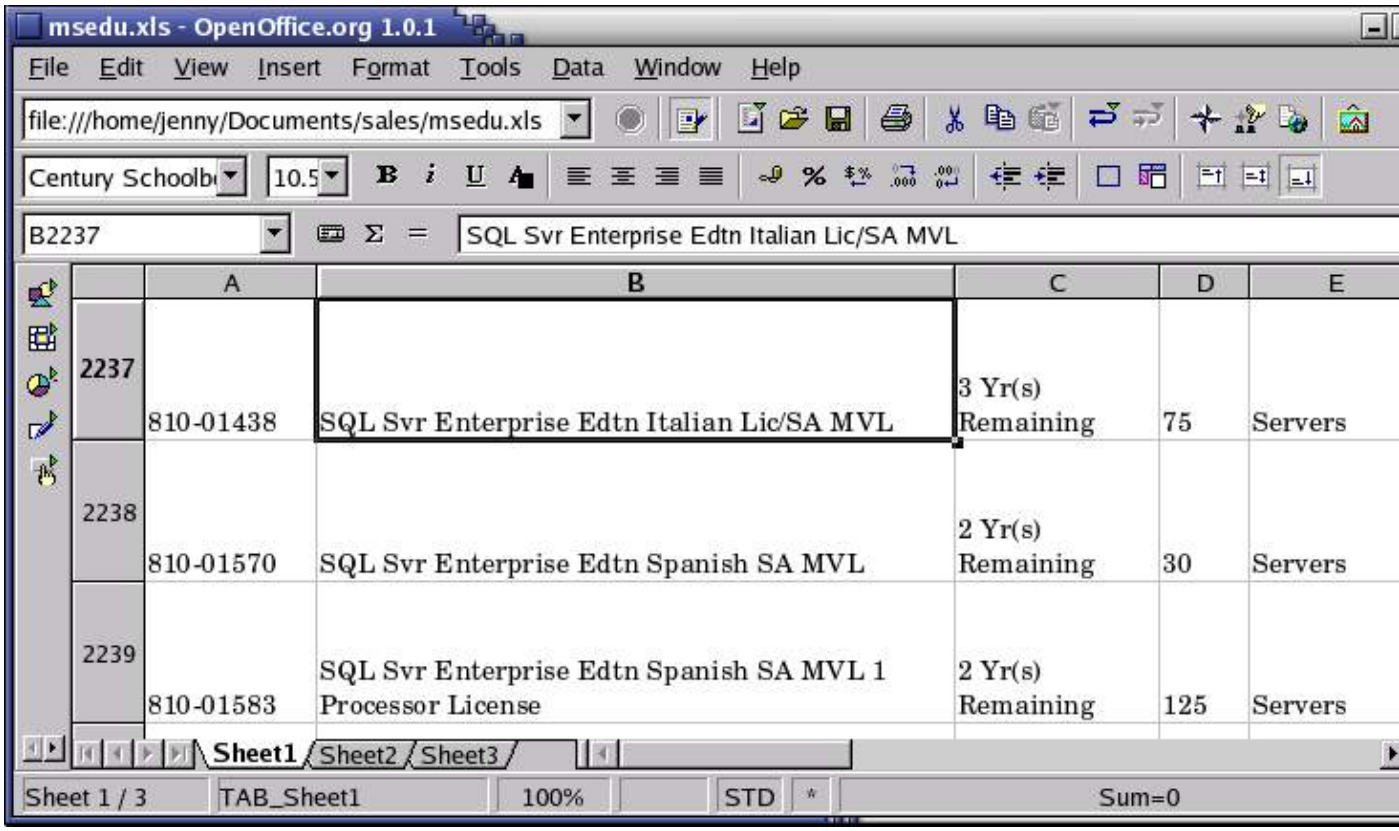
Maybe even more successful than the SAMBA project is the Apache HTTP server project. The server runs on UNIX, Windows NT and many other operating systems. Originally known as "A PAtCHy server", based on existing code and a series of "patch files", the name for the matured code deserves to be connoted with the native American tribe of the Apache, well-known for their superior skills in warfare strategy and inexhaustible endurance. Apache has been shown to be substantially faster, more stable and more feature-full than many other web servers. Apache is run on sites that get millions of visitors per day, and while no official support is provided by the developers, the Apache user community provides answers to all your questions. Commercial support is now being provided by a number of third parties.

In the category of office applications, a choice of MS Office suite clones is available, ranging from partial to full implementations of the applications available on MS Windows workstations. These initiatives helped a great deal to make Linux acceptable for the desktop market, because the users don't need extra training to learn how to work with new systems. With the desktop comes the praise of the common users, and not only their praise, but also their specific requirements, which are growing more intricate and demanding by the day.

The Open Source community, consisting largely of people who have been contributing for over half a decade, assures Linux' position as an important player on the desktop market as well as in general IT application. Paid employees and volunteers alike are working diligently so that Linux can maintain a position in the market. The more users, the more questions. The Open Source community makes sure answers keep coming, and watches the quality of the answers with a suspicious eye, resulting in ever more stability and accessibility.

Listing all the available Linux software is beyond the scope of this guide, as there are tens of thousands of packages. Throughout this course we will present you with the most common packages, which are almost all freely available. In order to take away some of the fear of the beginning user, here's a screenshot of one of your most-wanted programs. You can see for yourself that no effort has been spared to make users who are switching from Windows feel at home:

Figure 1-1. OpenOffice MS-compatible Spreadsheet



1.4. Properties of Linux

1.4.1. Linux Pros

A lot of the advantages of Linux are a consequence of Linux' origins, deeply rooted in UNIX, except for the first advantage, of course:

- Linux is free:

  As in free beer, they say. If you want to spend absolutely nothing, you don't even have to pay the price of a CD. Linux can be downloaded in its entirety from the Internet completely for free. No registration fees, no costs per user, free updates, and freely available source code in case you want to change the behavior of your system.

  Most of all, Linux is free as in free speech:

  The license commonly used is the GNU Public License (GPL). The license says that anybody who may want to do so, has the right to change Linux and eventually to redistribute a changed version, on the one condition that the code is still available after redistribution. In practice, you are free to grab a kernel image, for instance to add support for teletransportation machines or time travel and sell your new code, as long as your customers can still have a copy of that code.
- Linux is portable to any hardware platform:

  A vendor who wants to sell a new type of computer and who doesn't know what kind of OS his new machine will run (say the CPU in your car or washing machine), can take a Linux kernel and make it

work on his hardware, because documentation related to this activity is freely available.

- Linux was made to keep on running:

  As with UNIX, a Linux system expects to run without rebooting all the time. That is why a lot of tasks are being executed at night or scheduled automatically for other calm moments, resulting in higher availability during busier periods and a more balanced use of the hardware. This property allows for Linux to be applicable also in environments where people don't have the time or the possibility to control their systems night and day.
- Linux is secure and versatile:

  The security model used in Linux is based on the UNIX idea of security, which is known to be robust and of proven quality. But Linux is not only fit for use as a fort against enemy attacks from the Internet: it will adapt equally to other situations, utilizing the same high standards for security. Your development machine or control station will be as secure as your firewall.
- Linux is scalable:

  From a Palmtop with 2 MB of memory to a petabyte storage cluster with hundreds of nodes: add or remove the appropriate packages and Linux fits all. You don't need a supercomputer anymore, because you can use Linux to do big things using the building blocks provided with the system. If you want to do little things, such as making an operating system for an embedded processor or just recycling your old 486, Linux will do that as well.
- The Linux OS and most Linux applications have very short debug-times:

  Because Linux has been developed and tested by thousands of people, both errors and people to fix them are usually found rather quickly. It sometimes happens that there are only a couple of hours between discovery and fixing of a bug.

1.4.2. Linux Cons

- There are far too many different distributions:

  "Quot capites, tot rationes", as the Romans already said: the more people, the more opinions. At first glance, the amount of Linux distributions can be frightening, or ridiculous, depending on your point of view. But it also means that everyone will find what he or she needs. You don't need to be an expert to find a suitable release.

  When asked, generally every Linux user will say that the best distribution is the specific version he is using. So which one should you choose? Don't worry too much about that: all releases contain more or less the same set of basic packages. On top of the basics, special third party software is added making, for example, TurboLinux more suitable for the small and medium enterprise, RedHat for servers and SuSE for workstations. However, the differences are likely to be very superficial. The best strategy is to test a couple of distributions; unfortunately not everybody has the time for this. Luckily, there is plenty of advice on the subject of choosing your Linux. A quick search on Google, using the keywords "choosing your distribution" brings up tens of links to good advise. The Installation HOWTO also discusses choosing your distribution.
- Linux is not very user friendly and confusing for beginners:

  It must be said that Linux, at least the core system, is less userfriendly to use than MS Windows and certainly more difficult than MacOS, but... In light of its popularity, considerable effort has been made to make Linux even easier to use, especially for new users. More information is being released

daily, such as this guide, to help fill the gap for documentation available to users at all levels.

- Is an Open Source product trustworthy?

  How can something that is free also be reliable? Linux users have the choice whether to use Linux or not, which gives them an enormous advantage compared to users of proprietary software, who don't have that kind of freedom. After long periods of testing, most Linux users come to the conclusion that Linux is not only as good, but in many cases better and faster that the traditional solutions. If Linux were not trustworthy, it would have been long gone, never knowing the popularity it has now, with millions of users. Now users can influence their systems and share their remarks with the community, so the system gets better and better every day. It is a project that is never finished, that is true, but in an ever changing environment, Linux is also a project that continues to strive for perfection.

1.5. Linux Flavors

1.5.1. Linux and GNU

Although there are a large number of Linux implementations, you will find a lot of similarities in the different distributions, if only because every Linux machine is a box with building blocks that you may put together following your own needs and views. Installing the system is only the beginning of a longterm relationship. Just when you think you have a nice running system, Linux will stimulate your imagination and creativeness, and the more you realize what power the system can give you, the more you will try to redefine its limits.

Linux may appear different depending on the distribution, your hardware and personal taste, but the fundamentals on which all graphical and other interfaces are built, remain the same. The Linux system is based on GNU tools (Gnu's Not UNIX), which provide a set of standard ways to handle and use the system. All GNU tools are open source, so they can be installed on any system. Most distributions offer pre-compiled packages of most common tools, such as RPM packages on RedHat and Debian packages (also called deb or dpkg) on Debian, so you needn't be a programmer to install a package on your system. However, if you are and like doing things yourself, you will enjoy Linux all the better, since most distributions come with a complete set of development tools, allowing installation of new software purely from source code. This setup also allows you to install software even if it does not exist in a pre-packaged form suitable for your system.

A list of common GNU software:

- Bash: The GNU shell
- GCC: The GNU C Compiler
- GDB: The GNU Debugger
- Coreutils: a set of basic UNIX-style utilities, such as **ls**, **cat** and **chmod**
- Findutils: to search and find files
- Fontutils: to convert fonts from one format to another or make new fonts
- The Gimp: GNU Image Manipulation Program
- Gnome: the GNU desktop environment
- Emacs: a very powerful editor
- Ghostscript and Ghostview: interpreter and graphical frontend for PostScript files.
- GNU Photo: software for interaction with digital cameras
- Octave: a programming language, primarily intended to perform numerical computations and image processing.
- GNU SQL: relational database system
- Radius: a remote authentication and accounting server
- ...

Many commercial applications are available for Linux, and for more information about these packages we refer to their specific documentation. Throughout this guide we will only discuss freely available software, which comes (in most cases) with a GNU license.

To install missing or new packages, you will need some form of software management. The most common implementations include RPM and dpkg. RPM is the RedHat Package Manager, which is used on a variety of Linux systems, eventhough the name does not suggest this. Dpkg is the Debian package management system, which uses an interface called **apt-get**, that can manage RPM packages as well. Novell Ximian Red Carpet is a third party implementation of RPM with a graphical front-end. Other third party software vendors may have their own installation procedures, sometimes resembling the InstallShield and such, as known on MS Windows and other platforms. As you advance into Linux, you will likely get in touch with one or more of these programs.

1.5.2. GNU/Linux

The Linux kernel (the *bones* of your system, see Section 3.2.3.1) is not part of the GNU project but uses the same license as GNU software. A great majority of utilities and development tools (the *meat* of your system), which are not Linux-specific, are taken from the GNU project. Because any usable system must contain both the kernel and at least a minimal set of utilities, some people argue that such a system should be called a *GNU/Linux* system.

In order to obtain the highest possible degree of independence between distributions, this is the sort of Linux that we will discuss throughout this course. If we are not talking about a GNU/Linux system, the specific distribution, version or program name will be mentioned.

1.5.3. Which distribution should I install?

Prior to installation, the most important factor is your hardware. Since every Linux distribution contains the basic packages and can be built to meet almost any requirement (because they all use the Linux kernel), you only need to consider if the distribution will run on your hardware. LinuxPPC for example has been made to run on Apple and other PowerPCs and does not run on an ordinary x86 based PC. LinuxPPC does run on the new Macs, but you can't use it for some of the older ones with ancient bus technology. Another tricky case is Sun hardware, which could be an old SPARC CPU or a newer UltraSparc, both requiring different versions of Linux.

Some Linux distributions are optimized for certain processors, such as Athlon CPUs, while they will at the same time run decent enough on the standard 486, 586 and 686 Intel processors. Sometimes distributions for special CPUs are not as reliable, since they are tested by fewer people.

Most Linux distributions offer a set of programs for generic PCs with special packages containing optimized kernels for the x86 Intel based CPUs. These distributions are well-tested and maintained on a regular basis, focusing on reliant server implementation and easy installation and update procedures. Examples are Debian, Ubuntu, Fedora, SuSE and Mandriva, which are by far the most popular Linux systems and generally considered easy to handle for the beginning user, while not blocking professionals from getting the most out of their Linux machines. Linux also runs decently on laptops and middle-range servers. Drivers for new hardware are included only after extensive testing, which adds to the stability of a system.

While the standard desktop might be Gnome on one system, another might offer KDE by default. Generally, both Gnome and KDE are available for all major Linux distributions. Other window and desktop managers are available for more advanced users.

The standard installation process allows users to choose between different basic setups, such as a workstation, where all packages needed for everyday use and development are installed, or a server installation, where different network services can be selected. Expert users can install every combination of packages they want during the initial installation process.

The goal of this guide is to apply to all Linux distributions. For your own convenience, however, it is strongly advised that beginners stick to a mainstream distribution, supporting all common hardware and applications by default. The following are very good choices for novices:

- Fedora Core
- Debian
- SuSE Linux
- Mandriva (former MandrakeSoft)
- Knoppix: an operating system that runs from your CD-ROM, you don't need to install anything.

Downloadable ISO-images can be obtained from LinuxISO.org. The main distributions can be purchased in any decent computer shop.

1.6. Summary

In this chapter, we learned that:

- Linux is an implementation of UNIX.
- The Linux operating system is written in the C programming language.
- "De gustibus et coloribus non disputandum est": there's a Linux for everyone.
- Linux uses GNU tools, a set of freely available standard tools for handling the operating system.

1.7. Exercises

A practical exercise for starters: install Linux on your PC. Read the installation manual for your distribution and/or the Installation HOWTO and do it.

Read the docs!

Most errors stem from not reading the information provided during the install. Reading the installation messages carefully is the first step on the road to success.

Things you must know BEFORE starting a Linux installation:

- Will this distribution run on my hardware?

  Check with http://www.tldp.org/HOWTO/Hardware-HOWTO/index.html when in doubt about compatibility of your hardware.
- What kind of keyboard do I have (number of keys, layout)? What kind of mouse (serial/parallel, number of buttons)? How many MB of RAM?
- Will I install a basic workstation or a server, or will I need to select specific packages myself?
- Will I install from my hard disk, from a CD-ROM, or using the network? Should I adapt the BIOS for any of this? Does the installation method require a boot disk?
- Will Linux be the only system on this computer, or will it be a dual boot installation? Should I make a large partition in order to install virtual systems later on, or is this a virtual installation itself?

- Is this computer in a network? What is its hostname, IP address? Are there any gateway servers or other important networked machines my box should communicate with?

  Linux expects to be networked

  Not using the network or configuring it incorrectly may result in slow startup.
- Is this computer a gateway/router/firewall? (If you have to think about this question, it probably isn't.)
- Partitioning: let the installation program do it for you this time, we will discuss partitions in detail in Chapter 3. There is system-specific documentation available if you want to know everything about it. If your Linux distribution does not offer default partitioning, that probably means it is not suited for beginners.
- Will this machine start up in text mode or in graphical mode?
- Think of a good password for the administrator of this machine (root). Create a non-root user account (non-privileged access to the system).
- Do I need a rescue disk? (recommended)
- Which languages do I want?

The full checklist can be found at http://www.tldp.org/HOWTO/Installation-HOWTO/index.html.

In the following chapters we will find out if the installation has been successful.

Chapter 2. Quickstart

In order to get the most out of this guide, we will immediately start with a practical chapter on connecting to the Linux system and doing some basic things.

We will discuss:

- Connecting to the system
- Disconnecting from the system
- Text and graphic mode
- Changing your password
- Navigating through the file system
- Determining file type
- Looking at text files
- Finding help

2.1. Logging in, activating the user interface and logging out

2.1.1. Introduction

In order to work on a Linux system directly, you will need to provide a user name and password. You always need to authenticate to the system. As we already mentioned in the exercise from Chapter 1, most PC-based Linux systems have two basic modes for a system to run in: either quick and sober in text console mode, which looks like DOS with mouse, multitasking and multi-user features, or in graphical mode, which looks better but eats more system resources.

2.1.2. Graphical mode

This is the default nowadays on most desktop computers. You know you will connect to the system using graphical mode when you are first asked for your user name, and then, in a new window, to type your password.

To log in, make sure the mouse pointer is in the login window, provide your user name and password to the system and click OK or press **Enter**.

Careful with that root account!

It is generally considered a bad idea to connect (graphically) using the *root* user name, the system adminstrator's account, since the use of graphics includes running a lot of extra programs, in root's case with a lot of extra permissions. To keep all risks as low as possible, use a normal user account to connect graphically. But there are enough risks to keep this in mind as a general advice, for all use of the root account: only log in as root when extra privileges are required.

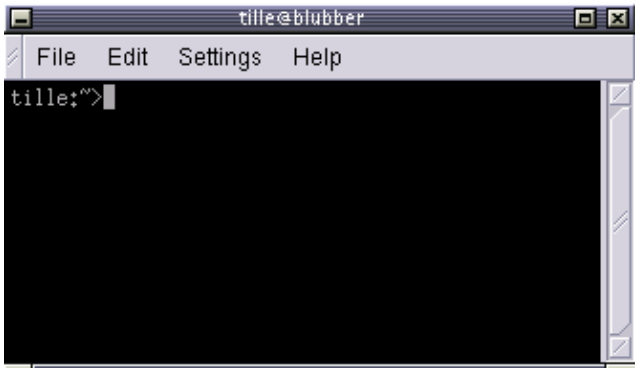
After entering your user name/password combination, it can take a little while before the graphical environment is started, depending on the CPU speed of your computer, on the software you use and on your personal settings.

To continue, you will need to open a *terminal window* or *xterm* for short (X being the name for the underlying software supporting the graphical environment). This program can be found in the Applications->Utilities, System Tools or Internet menu, depending on what window manager you are using. There might be icons that you can use as a shortcut to get an xterm window as well, and clicking the right mouse button on the desktop background will usually present you with a menu containing a terminal window application.

While browsing the menus, you will notice that a lot of things can be done without entering commands via the keyboard. For most users, the good old point-'n'-click method of dealing with the computer will do. But this guide is for future network and system administrators, who will need to meddle with the heart of the system. They need a stronger tool than a mouse to handle all the tasks they will face. This tool is the shell, and when in graphical mode, we activate our shell by opening a terminal window.

The terminal window is your control panel for the system. Almost everything that follows is done using this simple but powerful text tool. A terminal window should always show a command prompt when you open one. This terminal shows a standard prompt, which displays the user's login name, and the current working directory, represented by the twiddle (~):

Figure 2-1. Terminal window



Another common form for a prompt is this one:

```
[user@host dir]
```

In the above example, *user* will be your login name, *hosts* the name of the machine you are working on, and *dir* an indication of your current location in the file system.

Later we will discuss prompts and their behavior in detail. For now, it suffices to know that prompts can display all kinds of information, but that they are not part of the commands you are giving to your system.

To disconnect from the system in graphical mode, you need to close all terminal windows and other applications. After that, hit the logout icon or find Log Out in the menu. Closing everything is not really necessary, and the system can do this for you, but session management might put all currently open applications back on your screen when you connect again, which takes longer and is not always the desired effect. However, this behavior is configurable.

When you see the login screen again, asking to enter user name and password, logout was successful.

Gnome or KDE?

We mentioned both the Gnome and KDE desktops already a couple of times. These are the two most popular ways of managing your desktop, although there are many, many others. Whatever desktop you chose to work with is fine - as long as you know how to open a terminal window. However, we will continue to refer to both Gnome and KDE for the most popular ways of achieving certain tasks.

2.1.3. Text mode

You know you're in text mode when the whole screen is black, showing (in most cases white) characters. A text mode login screen typically shows some information about the machine you are working on, the name of the machine and a prompt waiting for you to log in:

```
RedHat Linux Release 8.0 (Psyche)

blast login: _
```

The login is different from a graphical login, in that you have to hit the **Enter** key after providing your user name, because there are no buttons on the screen that you can click with the mouse. Then you should type your password, followed by another **Enter**. You won't see any indication that you are entering something, not even an asterisk, and you won't see the cursor move. But this is normal on Linux and is done for security reasons.

When the system has accepted you as a valid user, you may get some more information, called the *message of the day*, which can be anything. Additionally, it is popular on UNIX systems to display a fortune cookie, which contains some general wise or unwise (this is up to you) thoughts. After that, you will be given a shell, indicated with the same prompt that you would get in graphical mode.

Don't log in as root

Also in text mode: log in as root only to do setup and configuration that absolutely requires administrator privileges, such as adding users, installing software packages, and performing network and other system configuration. Once you are finished, immediately leave the special account and resume your work as a non-privileged user. Alternatively, some systems, like Ubuntu, force you to use **sudo**, so that you do not need direct access to the administrative account.

Logging out is done by entering the **logout** command, followed by **Enter**. You are successfully disconnected from the system when you see the login screen again.

The power button

While Linux was not meant to be shut off without application of the proper procedures for halting the system, hitting the power button is equivalent to starting those procedures *on newer systems*. However, powering off an old system without going through the halting process might cause severe damage! If you want to be sure, always use the Shut down option when you log out from the graphical interface, or, when on the login screen (where you have to give your user name and password) look around for a shutdown button.

Now that we know how to connect to and disconnect from the system, we're ready for our first commands.

2.2. Absolute basics

2.2.1. The commands

These are the quickies, which we need to get started; we will discuss them later in more detail.

Table 2-1. Quickstart commands

Command	Meaning
ls	Displays a list of files in the current working directory, like the **dir** command in DOS
cd `directory`	change directories
passwd	change the password for the current user
file `filename`	display file type of file with name `filename`
cat `textfile`	throws content of `textfile` on the screen
pwd	display present working directory
exit or **logout**	leave this session
man ***command***	read man pages on **command**
info ***command***	read Info pages on **command**
apropos ***string***	search the *whatis* database for strings

2.2.2. General remarks

You type these commands after the prompt, in a terminal window in graphical mode or in text mode, followed by **Enter**.

Commands can be issued by themselves, such as **ls**. A command behaves different when you specify an *option*, usually preceded with a dash (-), as in **ls -a**. The same option character may have a different meaning for another command. GNU programs take long options, preceded by two dashes (--), like **ls --all**. Some commands have no options.

The argument(s) to a command are specifications for the object(s) on which you want the command to take effect. An example is **ls** ***/etc***, where the directory `/etc` is the argument to the **ls** command. This indicates that you want to see the content of that directory, instead of the default, which would be the content of the current directory, obtained by just typing **ls** followed by **Enter**. Some commands require arguments, sometimes arguments are optional.

You can find out whether a command takes options and arguments, and which ones are valid, by checking the online help for that command, see Section 2.3.

In Linux, like in UNIX, directories are separated using forward slashes, like the ones used in web addresses (URLs). We will discuss directory structure in-depth later.

The symbols . and .. have special meaning when directories are concerned. We will try to find out about those during the exercises, and more in the next chapter.

Try to avoid logging in with or using the system administrator's account, *root*. Besides doing your normal work, most tasks, including checking the system, collecting information etc., can be executed using a normal user account with no special permissions at all. If needed, for instance when creating a new user or installing new software, the preferred way of obtaining root access is by switching user IDs, see Section 3.2.1 for an example.

Almost all commands in this book can be executed without system administrator privileges. In most cases, when issuing a command or starting a program as a non-privileged user, the system will warn you or prompt you for the root password when root access is required. Once you're done, leave the application or session that gives you root privileges immediately.

Reading documentation should become your second nature. Especially in the beginning, it is important to read system documentation, manuals for basic commands, HOWTOs and so on. Since the amount of documentation is so enormous, it is impossible to include all related documentation. This book will try to guide you to the most appropriate documentation on every subject discussed, in order to stimulate the habit of reading the man pages.

2.2.3. Using Bash features

Several special key combinations allow you to do things easier and faster with the GNU shell, Bash, which is the default on almost any Linux system, see Section 3.2.3.2. Below is a list of the most commonly used features; you are strongly suggested to make a habit out of using them, so as to get the most out of your Linux experience from the very beginning.

Table 2-2. Key combinations in Bash

Key or key combination	Function
Ctrl+**A**	Move cursor to the beginning of the command line.
Ctrl+**C**	End a running program and return the prompt, see Chapter 4.
Ctrl+**D**	Log out of the current shell session, equal to typing **exit** or **logout**.
Ctrl+**E**	Move cursor to the end of the command line.
Ctrl+**H**	Generate backspace character.
Ctrl+**L**	Clear this terminal.
Ctrl+**R**	Search command history, see Section 3.3.3.4.
Ctrl+**Z**	Suspend a program, see Chapter 4.
ArrowLeft and **ArrowRight**	Move the cursor one place to the left or right on the command line, so that you can insert characters at other places than just at the beginning and the end.
ArrowUp and **ArrowDown**	Browse history. Go to the line that you want to repeat, edit details if necessary, and press **Enter** to save time.
Shift+**PageUp** and **Shift**+**PageDown**	Browse terminal buffer (to see text that has "scrolled off" the screen).
Tab	Command or filename completion; when multiple choices are possible, the system will either signal with an audio or visual bell, or, if too many choices are possible, ask you if you want to see them all.
Tab Tab	Shows file or command completion possibilities.

The last two items in the above table may need some extra explanations. For instance, if you want to change into the directory `directory_with_a_very_long_name`, you are not going to type that very long name, no. You just type on the command line **cd dir**, then you press **Tab** and the shell completes the name for you, if no other files are starting with the same three characters. Of course, if there are no other items starting with "d", then you might just as wel type **cd d** and then **Tab**. If more than one file starts with the same characters, the shell will signal this to you, upon which you can hit **Tab** twice with short interval, and the shell presents the choices you have:

```
your_prompt> cd st
starthere        stuff          stuffit
```

In the above example, if you type "a" after the first two characters and hit **Tab** again, no other possibilities are left, and the shell completes the directory name, without you having to type the string "rthere":

```
your_prompt> cd starthere
```

Of course, you'll still have to hit **Enter** to accept this choice.

In the same example, if you type "u", and then hit **Tab**, the shell will add the "ff" for you, but then it protests again, because multiple choices are possible. If you type **Tab Tab** again, you'll see the choices; if you type one or more characters that make the choice unambiguous to the system, and **Tab** again, or **Enter** when you've reach the end of the file name that you want to choose, the shell completes the file name and changes you into that directory - if indeed it is a directory name.

This works for all file names that are arguments to commands.

The same goes for command name completion. Typing **ls** and then hitting the **Tab** key twice, lists all the commands in your `PATH` (see Section 3.2.1) that start with these two characters:

```
your_prompt> ls
ls           lsdev        lspci        lsraid       lsw
lsattr       lsmod        lspgpot      lss16toppm
lsb_release  lsof         lspnp        lsusb
```

2.3. Getting help

2.3.1. Be warned

GNU/Linux is all about becoming more self-reliant. And as usual with this system, there are several ways to achieve the goal. A common way of getting help is finding someone who knows, and however patient and peace-loving the Linux-using community will be, almost everybody will expect you to have tried one or more of the methods in this section before asking them, and the ways in which this viewpoint is expressed may be rather harsh if you prove not to have followed this basic rule.

2.3.2. The man pages

A lot of beginning users fear the man (manual) pages, because they are an overwhelming source of documentation. They are, however, very structured, as you will see from the example below on: **man `man`**.

Reading man pages is usually done in a terminal window when in graphical mode, or just in text mode if you prefer it. Type the command like this at the prompt, followed by **Enter**:

```
yourname@yourcomp ~> man man
```

The documentation for **man** will be displayed on your screen after you press **Enter**:

```
man(1)                                                          man(1)



NAME
 man - format and display the on-line manual pages
 manpath - determine user's search path for man pages

SYNOPSIS
 man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
 [-M pathlist] [-P pager] [-S section_list] [section] name ...


DESCRIPTION
 man formats and displays the on-line manual pages.  If you specify
 section, man only looks in that section of the manual.
 name is normally the name of the manual page, which is typically the
 name of a  command, function, or file.  However, if name contains a
 slash (/) then man interprets it as a file specification, so that you
 can do man ./foo.5 or even man /cd/foo/bar.1.gz.

 See  below  for  a  description  of where man looks for the manual
 page files.

OPTIONS
 -C  config_file
lines 1-27
```

Browse to the next page using the space bar. You can go back to the previous page using the b-key. When you reach the end, **man** will usually quit and you get the prompt back. Type **q** if you want to leave the man page before reaching the end, or if the viewer does not quit automatically at the end of the page.

Pagers

The available key combinations for manipulating the man pages depend on the *pager* used in your distribution. Most distributions use **less** to view the man pages and to scroll around. See Section 3.3.4.2 for more info on pagers.

Each man page usually contains a couple of standard sections, as we can see from the **man `man`** example:

- The first line contains the name of the command you are reading about, and the id of the section in which this man page is located. The man pages are ordered in chapters. Commands are likely to have multiple man pages, for example the man page from the user section, the man page from the system admin section, and the man page from the programmer section.
- The name of the command and a short description are given, which is used for building an index of the man pages. You can look for any given search string in this index using the **apropos** command.
- The synopsis of the command provides a technical notation of all the options and/or arguments this command can take. You can think of an option as a way of executing the command. The argument is what you execute it on. Some commands have no options or no arguments. Optional options and arguments are put in between "[" and "]" to indicate that they can be left out.
- A longer description of the command is given.
- Options with their descriptions are listed. Options can usually be combined. If not so, this section will tell you about it.
- Environment describes the shell variables that influence the behavior of this command (not all commands have this).

- Sometimes sections specific to this command are provided.
- A reference to other man pages is given in the "SEE ALSO" section. In between parentheses is the number of the man page section in which to find this command. Experienced users often switch to the "SEE ALSO" part using the / command followed by the search string SEE and press **Enter**.
- Usually there is also information about known bugs (anomalies) and where to report new bugs you may find.
- There might also be author and copyright information.

Some commands have multiple man pages. For instance, the **passwd** command has a man page in section 1 and another in section 5. By default, the man page with the lowest number is shown. If you want to see another section than the default, specify it after the **man** command:

man 5 *passwd*

If you want to see all man pages about a command, one after the other, use the `-a` to man:

man `-a` *passwd*

This way, when you reach the end of the first man page and press **SPACE** again, the man page from the next section will be displayed.

2.3.3. More info

2.3.3.1. The Info pages

In addition to the man pages, you can read the Info pages about a command, using the **info** command. These usually contain more recent information and are somewhat easier to use. The man pages for some commands refer to the Info pages.

Get started by typing **info *info*** in a terminal window:

```
File: info.info,  Node: Top,  Next: Getting Started,  Up: (dir)

Info: An Introduction
*********************

   Info is a program, which you are using now, for reading
documentation of computer programs.  The GNU Project distributes most
of its on-line manuals in the Info format, so you need a program called
"Info reader" to read the manuals.  One of such programs you are using
now.

   If you are new to Info and want to learn how to use it, type the
command `h' now.  It brings you to a programmed instruction sequence.

   To learn advanced Info commands, type `n' twice.  This brings you to
`Info for Experts', skipping over the `Getting Started' chapter.

* Menu:

* Getting Started::             Getting started using an Info reader.
* Advanced Info::               Advanced commands within Info.
* Creating an Info File::       How to make your own Info file.
--zz-Info: (info.info.gz)Top, 24 lines --Top------------------------------
Welcome to Info version 4.2. Type C-h for help, m for menu item.
```

Use the arrow keys to browse through the text and move the cursor on a line starting with an asterisk, containing the keyword about which you want info, then hit **Enter**. Use the **P** and **N** keys to go to the previous or next subject. The space bar will move you one page further, no matter whether this starts a new subject or an Info page for another command. Use **Q** to quit. The **info** program has more information.

2.3.3.2. The whatis and apropos commands

A short index of explanations for commands is available using the **whatis** command, like in the examples below:

```
[your_prompt] whatis ls
ls                   (1)  - list directory contents
```

This displays short information about a command, and the first section in the collection of man pages that contains an appropriate page.

If you don't know where to get started and which man page to read, **apropos** gives more information. Say that you don't know how to start a browser, then you could enter the following command:

```
another prompt> apropos browser
Galeon [galeon](1)  - gecko-based GNOME web browser
lynx           (1)  - a general purpose distributed information browser
                      for the World Wide Web
ncftp          (1)  - Browser program for the File Transfer Protocol
opera          (1)  - a graphical web browser
pilot          (1)  - simple file system browser in the style of the
                      Pine Composer
pinfo          (1)  - curses based lynx-style info browser
pinfo [pman]   (1)  - curses based lynx-style info browser
viewres        (1x)  - graphical class browser for Xt
```

After pressing **Enter** you will see that a lot of browser related stuff is on your machine: not only web browsers, but also file and FTP browsers, and browsers for documentation. If you have development packages installed, you may also have the accompanying man pages dealing with writing programs having to do with browsers. Generally, a command with a man page in section one, so one marked with "(1)", is suitable for trying out as a user. The user who issued the above **apropos** might consequently try to start the commands **galeon**, **lynx** or **opera**, since these clearly have to do with browsing the world wide web.

2.3.3.3. The --help option

Most GNU commands support the `--help`, which gives a short explanation about how to use the command and a list of available options. Below is the output of this option with the **cat** command:

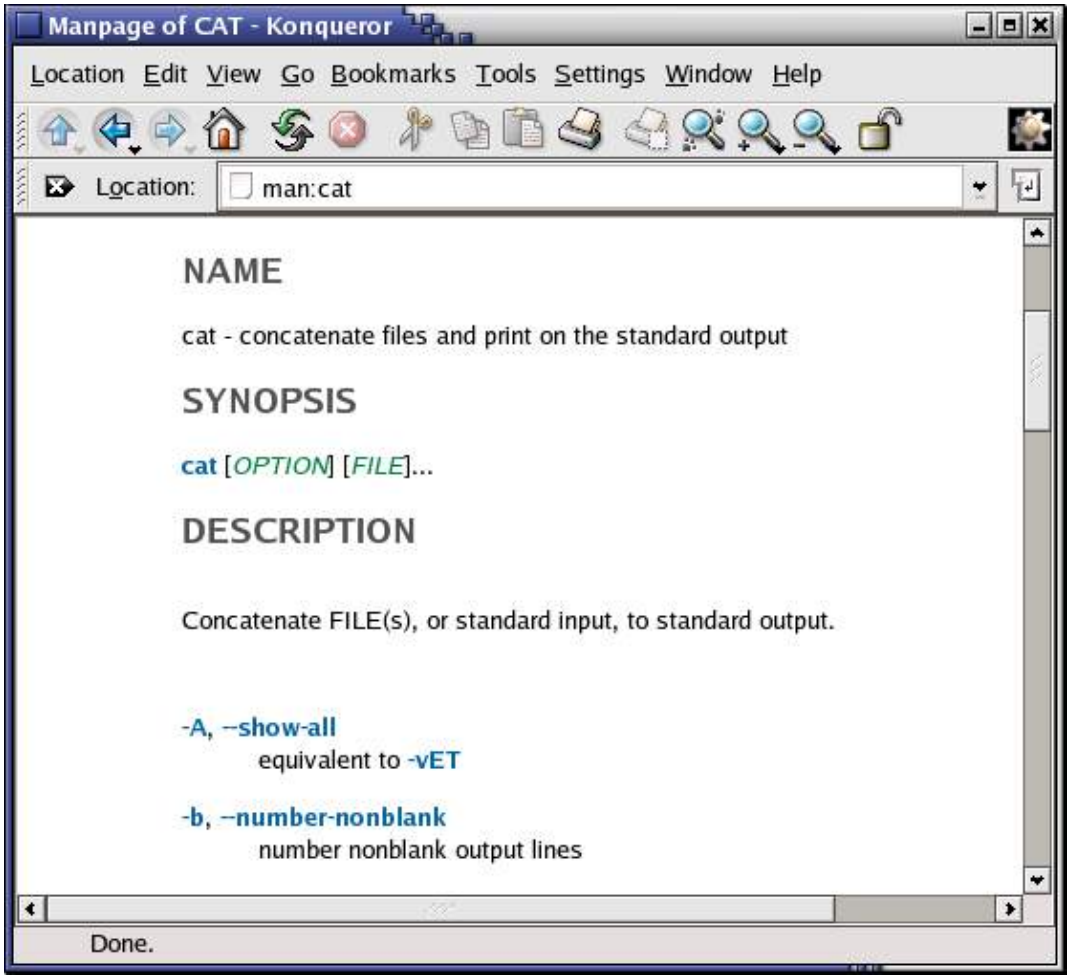
```
userprompt@host: cat --help
Usage: cat [OPTION] [FILE]...
Concatenate FILE(s), or standard input, to standard output.

  -A, --show-all           equivalent to -vET
  -b, --number-nonblank    number nonblank output lines
  -e                       equivalent to -vE
  -E, --show-ends          display $ at end of each line
  -n, --number             number all output lines
  -s, --squeeze-blank      never more than one single blank line
  -t                       equivalent to -vT
  -T, --show-tabs          display TAB characters as ^I
  -u                       (ignored)
  -v, --show-nonprinting   use ^ and M- notation,
```

```
                          except for LFD and TAB
      --help     display this help and exit
      --version  output version information and exit

With no FILE, or when FILE is -, read standard input.

Report bugs to <bug-textutils@gnu.org>.
```

2.3.3.4. Graphical help

Don't despair if you prefer a graphical user interface. Konqueror, the default KDE file manager, provides painless and colourful access to the man and Info pages. You may want to try "info:info" in the *Location* address bar, and you will get a browsable Info page about the **info** command. Similarly, "man:ls" will present you with the man page for the **ls** command. You even get command name completion: you will see the man pages for all the commands starting with "ls" in a scroll-down menu. Entering "info:/dir" in the address location toolbar displays all the Info pages, arranged in utility categories. Excellent Help content, including the Konqueror Handbook. Start up from the menu or by typing the command **konqueror** in a terminal window, followed by **Enter**; see the screenshot below.

Figure 2-2. Konqueror as help browser



The Gnome Help Browser is very user friendly as well. You can start it selecting Applications->Help from the Gnome menu, by clicking the lifeguard icon on your desktop or by entering the command **gnome-help** in a terminal window. The system documentation and man pages are easily browsable with a plain interface.

The **nautilus** file manager provides a searchable index of the man and Info pages, they are easily browsable and interlinked. Nautilus is started from the command line, or clicking your home directory icon, or from the Gnome menu.

The big advantage of GUIs for system documentation is that all information is completely interlinked, so you can click through in the "SEE ALSO" sections and wherever links to other man pages appear, and thus browse and acquire knowledge without interruption for hours at the time.

2.3.3.5. Exceptions

Some commands don't have separate documentation, because they are part of another command. **cd**, **exit**, **logout** and **pwd** are such exceptions. They are part of your shell program and are called *shell built-in* commands. For information about these, refer to the man or info page of your shell. Most beginning Linux users have a Bash shell. See Section 3.2.3.2 for more about shells.

If you have been changing your original system configuration, it might also be possible that man pages are still there, but not visible because your shell environment has changed. In that case, you will need to check the `MANPATH` variable. How to do this is explained in Section 7.2.1.2.

Some programs or packages only have a set of instructions or references in the directory `/usr/share/doc`. See Section 3.3.4 to display.

In the worst case, you may have removed the documentation from your system by accident (hopefully by accident, because it is a very bad idea to do this on purpose). In that case, first try to make sure that there is really nothing appropriate left using a search tool, read on in Section 3.3.3. If so, you may have to re-install the package that contains the command to which the documentation applied, see Section 7.5.

2.4. Summary

Linux traditionally operates in text mode or in graphical mode. Since CPU power and RAM are not the cost anymore these days, every Linux user can afford to work in graphical mode and will usually do so. This does not mean that you don't have to know about text mode: we will work in the text environment throughout this course, using a terminal window.

Linux encourages its users to acquire knowledge and to become independent. Inevitably, you will have to read a lot of documentation to achieve that goal; that is why, as you will notice, we refer to extra documentation for almost every command, tool and problem listed in this book. The more docs you read, the easier it will become and the faster you will leaf through manuals. Make reading documentation a habit as soon as possible. When you don't know the answer to a problem, refering to the documentation should become a second nature.

We already learned some commands:

Table 2-3. New commands in chapter 2: Basics

Command	Meaning

apropos	Search information about a command or subject.
cat	Show content of one or more files.
cd	Change into another directory.
exit	Leave a shell session.
file	Get information about the content of a file.
info	Read Info pages about a command.
logout	Leave a shell session.
ls	List directory content.
man	Read manual pages of a command.
passwd	Change your password.
pwd	Display the current working directory.

2.5. Exercises

Most of what we learn is by making mistakes and by seeing how things can go wrong. These exercises are made to get you to read some error messages. The order in which you do these exercises is important.

Don't forget to use the Bash features on the command line: try to do the exercises typing as few characters as possible!

2.5.1. Connecting and disconnecting

- Determine whether you are working in text or in graphical mode.

  I am working in text/graphical mode. (cross out what's not applicable)
- Log in with the user name and password you made for yourself during the installation.
- Log out.
- Log in again, using a non-existent user name

  -> What happens?

2.5.2. Passwords

Log in again with your user name and password.

- Change your password into *P6p3.aa!* and hit the **Enter** key.

  -> What happens?
- Try again, this time enter a password that is ridiculously easy, like *123* or *aaa*.

  -> What happens?
- Try again, this time don't enter a password but just hit the **Enter** key.

  -> What happens?
- Try the command **psswd** instead of **passwd**

  -> What happens?

New password

Unless you change your password back again to what it was before this exercise, it will be "P6p3.aa!". Change your password after this exercise!

Note that some systems might not allow to recycle passwords, i.e. restore the original one within a certain amount of time or a certain amount of password changes, or both.

2.5.3. Directories

These are some exercises to help you get the feel.

- Enter the command **cd `blah`**

  -> What happens?
- Enter the command **cd `..`**

  Mind the space between "cd" and ".."! Use the **pwd** command.

  -> What happens?
- List the directory contents with the **ls** command.

  -> What do you see?

  -> What do you think these are?

  -> Check using the **pwd** command.
- Enter the **cd** command.

  -> What happens?
- Repeat step 2 two times.

  -> What happens?
- Display the content of this directory.
- Try the command **cd `root`**

  -> What happens?

  -> To which directories do you have access?
- Repeat step 4.

  Do you know another possibility to get where you are now?

2.5.4. Files

- Change directory to `/` and then to `etc`. Type **ls**; if the output is longer than your screen, make the window longer, or try **Shift**+**PageUp** and **Shift**+**PageDown**.

  The file `inittab` contains the answer to the first question in this list. Try the **file** command on it.

-> The file type of my `inittab` is .....

- Use the command **cat `inittab`** and read the file.

-> What is the default mode of your computer?

- Return to your home directory using the **cd** command.
- Enter the command **file .**

-> Does this help to find the meaning of "."?

- Can you look at "." using the **cat** command?
- Display help for the **cat** program, using the `--help` option. Use the option for numbering of output lines to count how many users are listed in the file `/etc/passwd`.

2.5.5. Getting help

- Read **man *`intro`***
- Read **man *`ls`***
- Read **info *`passwd`***
- Enter the **apropos `pwd`** command.
- Try **man** or **info** on **cd**.

-> How would you find out more about **cd**?

- Read **ls `--help`** and try it out.

Chapter 3. About files and the file system

After the initial exploration in Chapter 2, we are ready to discuss the files and directories on a Linux system in more detail. Many users have difficulties with Linux because they lack an overview of what kind of data is kept in which locations. We will try to shine some light on the organization of files in the file system.

We will also list the most important files and directories and use different methods of viewing the content of those files, and learn how files and directories can be created, moved and deleted.

After completion of the exercises in this chapter, you will be able to:

- Describe the layout of a Linux file system
- Display and set paths
- Describe the most important files, including kernel and shell
- Find lost and hidden files
- Create, move and delete files and directories
- Display contents of files
- Understand and use different link types
- Find out about file properties and change file permissions

3.1. General overview of the Linux file system

3.1.1. Files

3.1.1.1. General

A simple description of the UNIX system, also applicable to Linux, is this:

"On a UNIX system, everything is a file; if something is not a file, it is a process."

This statement is true because there are special files that are more than just files (named pipes and sockets, for instance), but to keep things simple, saying that everything is a file is an acceptable generalization. A Linux system, just like UNIX, makes no difference between a file and a directory, since a directory is just a file containing names of other files. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files, according to the system.

In order to manage all those files in an orderly fashion, man likes to think of them in an ordered tree-like structure on the hard disk, as we know from MS-DOS (Disk Operating System) for instance. The large branches contain more branches, and the branches at the end contain the tree's leaves or normal files. For now we will use this image of the tree, but we will find out later why this is not a fully accurate image.

3.1.1.2. Sorts of files

Most files are just files, called *regular* files; they contain normal data, for example text files, executable files or programs, input for or output from a program and so on.

paths upon a call for **wc**, this "home-made" program is executed, with input it probably doesn't understand, so we have to stop it. To resolve this problem there are several ways (there are always several ways to solve a problem in UNIX/Linux): one answer could be to rename the user's **wc** program, or the user can give the full path to the exact command he wants, which can be found by using the `-a` option to the **which** command.

If the user uses programs in the other directories more frequently, he can change his path to look in his own directories last:

```
jumper:~> export PATH=/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin:\
/usr/bin:/usr/sbin:/bin:/sbin:/home/jumper/bin
```

Changes are not permanent!

Note that when using the **export** command in a shell, the changes are temporary and only valid for this session (until you log out). Opening new sessions, even while the current one is still running, will not result in a new path in the new session. We will see in Section 7.2 how we can make these kinds of changes to the environment permanent, adding these lines to the shell configuration files.

3.2.2. Absolute and relative paths

A path, which is the way you need to follow in the tree structure to reach a given file, can be described as starting from the trunk of the tree (the / or root directory). In that case, the path starts with a slash and is called an absolute path, since there can be no mistake: only one file on the system can comply.

In the other case, the path doesn't start with a slash and confusion is possible between `~/bin/wc` (in the user's home directory) and `bin/wc` in `/usr`, from the previous example. Paths that don't start with a slash are always relative.

In relative paths we also use the . and .. indications for the current and the parent directory. A couple of practical examples:

- When you want to compile source code, the installation documentation often instructs you to run the command **./configure**, which runs the *configure* program located in the current directory (that came with the new code), as opposed to running another configure program elsewhere on the system.
- In HTML files, relative paths are often used to make a set of pages easily movable to another place:
  ```
  <img alt="Garden with trees" src="../images/garden.jpg">
  ```
- Notice the difference one more time:
  ```
  theo:~> ls /mp3
  ls: /mp3: No such file or directory
  theo:~>ls mp3/
  oriental/  pop/  sixties/
  ```

3.2.3. The most important files and directories

3.2.3.1. The kernel

The kernel is the heart of the system. It manages the communication between the underlying hardware and the peripherals. The kernel also makes sure that processes and daemons (server processes) are started and stopped at the exact right times. The kernel has a lot of other important tasks, so many that there is a special kernel-development mailing list on this subject only, where huge amounts of information are shared. It would lead us too far to discuss the kernel in detail. For now it suffices to know that the kernel is the most important

file on the system.

3.2.3.2. The shell

3.2.3.2.1. What is a shell?

When I was looking for an appropriate explanation on the concept of a *shell*, it gave me more trouble than I expected. All kinds of definitions are available, ranging from the simple comparison that "the shell is the steering wheel of the car", to the vague definition in the Bash manual which says that "bash is an sh-compatible command language interpreter," or an even more obscure expression, "a shell manages the interaction between the system and its users". A shell is much more than that.

A shell can best be compared with a way of talking to the computer, a language. Most users do know that other language, the point-and-click language of the desktop. But in that language the computer is leading the conversation, while the user has the passive role of picking tasks from the ones presented. It is very difficult for a programmer to include all options and possible uses of a command in the GUI-format. Thus, GUIs are almost always less capable than the command or commands that form the backend.

The shell, on the other hand, is an advanced way of communicating with the system, because it allows for two-way conversation and taking initiative. Both partners in the communication are equal, so new ideas can be tested. The shell allows the user to handle a system in a very flexible way. An additional asset is that the shell allows for task automation.

3.2.3.2.2. Shell types

Just like people know different languages and dialects, the computer knows different shell types:

- **sh** or Bourne Shell: the original shell still used on UNIX systems and in UNIX related environments. This is the basic shell, a small program with few features. When in POSIX-compatible mode, **bash** will emulate this shell.
- **bash** or Bourne Again SHell: the standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user. On Linux, **bash** is the standard shell for common users. This shell is a so-called *superset* of the Bourne shell, a set of add-ons and plug-ins. This means that the Bourne Again SHell is compatible with the Bourne shell: commands that work in **sh**, also work in **bash**. However, the reverse is not always the case. All examples and exercises in this book use **bash**.
- **csh** or C Shell: the syntax of this shell resembles that of the C programming language. Sometimes asked for by programmers.
- **tcsh** or Turbo C Shell: a superset of the common C Shell, enhancing user-friendliness and speed.
- **ksh** or the Korn shell: sometimes appreciated by people with a UNIX background. A superset of the Bourne shell; with standard configuration a nightmare for beginning users.

The file `/etc/shells` gives an overview of known shells on a Linux system:

```
mia:~> cat /etc/shells
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
```

 **Fake Bourne shell**

Note that `/bin/sh` is usually a link to Bash, which will execute in Bourne shell compatible mode when called on this way.

Your default shell is set in the `/etc/passwd` file, like this line for user *mia*:

```
mia:L2NOfqdlPrHwE:504:504:Mia Maya:/home/mia:/bin/bash
```

To switch from one shell to another, just enter the name of the new shell in the active terminal. The system finds the directory where the name occurs using the PATH settings, and since a shell is an executable file (program), the current shell activates it and it gets executed. A new prompt is usually shown, because each shell has its typical appearance:

```
mia:~> tcsh
[mia@post21 ~]$
```

3.2.3.2.3. Which shell am I using?

If you don't know which shell you are using, either check the line for your account in `/etc/passwd` or type the command

echo $SHELL

3.2.3.3. Your home directory

Your home directory is your default destination when connecting to the system. In most cases it is a subdirectory of `/home`, though this may vary. Your home directory may be located on the hard disk of a remote file server; in that case your home directory may be found in `/nethome/your_user_name`. In another case the system administrator may have opted for a less comprehensible layout and your home directory may be on `/disk6/HU/07/jgillard`.

Whatever the path to your home directory, you don't have to worry too much about it. The correct path to your home directory is stored in the HOME environment variable, in case some program needs it. With the **echo** command you can display the content of this variable:

```
orlando:~> echo $HOME
/nethome/orlando
```

You can do whatever you like in your home directory. You can put as many files in as many directories as you want, although the total amount of data and files is naturally limited because of the hardware and size of the partitions, and sometimes because the system administrator has applied a quota system. Limiting disk usage was common practice when hard disk space was still expensive. Nowadays, limits are almost exclusively applied in large environments. You can see for yourself if a limit is set using the **quota** command:

```
pierre@lamaison:/> quota -v
Diskquotas for user pierre (uid 501): none
```

In case quotas have been set, you get a list of the limited partitions and their specific limitations. Exceeding the limits may be tolerated during a grace period with fewer or no restrictions at all. Detailed information can be found using the **info *quota*** or **man *quota*** commands.

No Quota?

If your system can not find the **quota**, then no limitation of file system usage is being applied.

Your home directory is indicated by a tilde (~), shorthand for `/path_to_home/user_name`. This same path is stored in the HOME variable, so you don't have to do anything to activate it. A simple application:

switch from `/var/music/albums/arno/2001` to `images` in your home directory using one elegant command:

```
rom:/var/music/albums/arno/2001> cd ~/images

rom:~/images> pwd
/home/rom/images
```

Later in this chapter we will talk about the commands for managing files and directories in order to keep your home directory tidy.

3.2.4. The most important configuration files

As we mentioned before, most configuration files are stored in the `/etc` directory. Content can be viewed using the **cat** command, which sends text files to the standard output (usually your monitor). The syntax is straight forward:

cat `file1` `file2` ... `fileN`

In this section we try to give an overview of the most common configuration files. This is certainly not a complete list. Adding extra packages may also add extra configuration files in `/etc`. When reading the configuration files, you will find that they are usually quite well commented and self-explanatory. Some files also have man pages which contain extra documentation, such as **man *`group`***.

Table 3-3. Most common configuration files

File	Information/service
`aliases`	Mail aliases file for use with the Sendmail and Postfix mail server. Running a mail server on each and every system has long been common use in the UNIX world, and almost every Linux distribution still comes with a Sendmail package. In this file local user names are matched with real names as they occur in E-mail addresses, or with other local addresses.
`apache`	Config files for the Apache web server.
`bashrc`	The system-wide configuration file for the Bourne Again SHell. Defines functions and aliases for all users. Other shells may have their own system-wide config files, like `cshrc`.
`crontab` and the `cron.*` directories	Configuration of tasks that need to be executed periodically - backups, updates of the system databases, cleaning of the system, rotating logs etc.
`default`	Default options for certain commands, such as **useradd**.
`filesystems`	Known file systems: ext3, vfat, iso9660 etc.
`fstab`	Lists partitions and their *mount points*.
`ftp*`	Configuration of the ftp-server: who can connect, what parts of the system are accessible etc.
`group`	Configuration file for user groups. Use the shadow utilities **groupadd**, **groupmod** and **groupdel** to edit this file. Edit manually only if you really know what you are doing.

red	compressed archives
white	text files
pink	images
cyan	links
yellow	devices
green	executables
flashing red	broken links

More information is in the man page. The same information was in earlier days displayed using suffixes to every non-standard file name. For mono-color use (like printing a directory listing) and for general readability, this scheme is still in use:

Table 3-6. Default suffix scheme for ls

Character	File type
nothing	regular file
/	directory
*	executable file
@	link
=	socket
|	named pipe

A description of the full functionality and features of the **ls** command can be read with **info *coreutils ls***.

3.3.1.2. More tools

To find out more about the kind of data we are dealing with, we use the **file** command. By applying certain tests that check properties of a file in the file system, magic numbers and language tests, **file** tries to make an educated guess about the format of a file. Some examples:

```
mike:~> file Documents/
Documents/: directory

mike:~> file high-tech-stats.pdf
high-tech-stats.pdf: PDF document, version 1.2

mike:~> file Nari-288.rm
Nari-288.rm: RealMedia file

mike:~> file bijlage10.sdw
bijlage10.sdw: Microsoft Office Document

mike:~> file logo.xcf
logo.xcf: GIMP XCF image data, version 0, 150 x 38, RGB Color

mike:~> file cv.txt
cv.txt: ISO-8859 text

mike:~> file image.png
image.png: PNG image data, 616 x 862, 8-bit grayscale, non-interlaced

mike:~> file figure
figure: ASCII text
```

```
mike:~> file me+tux.jpg
me+tux.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI),
            "28 Jun 1999", 144 x 144

mike:~> file 42.zip.gz
42.zip.gz: gzip compressed data, deflated, original filename,
         `42.zip', last modified: Thu Nov  1 23:45:39 2001, os: Unix

mike:~> file vi.gif
vi.gif: GIF image data, version 89a, 88 x 31

mike:~> file slide1
slide1: HTML document text

mike:~> file template.xls
template.xls: Microsoft Office Document

mike:~> file abook.ps
abook.ps: PostScript document text conforming at level 2.0

mike:~> file /dev/log
/dev/log: socket

mike:~> file /dev/hda
/dev/hda: block special (3/0)
```

The **file** command has a series of options, among others the `-z` option to look into compressed files. See **info *file*** for a detailed description. Keep in mind that the results of **file** are not absolute, it is only a guess. In other words, **file** can be tricked.

Why all the fuss about file types and formats?

Shortly, we will discuss a couple of command-line tools for looking at *plain text files*. These tools will not work when used on the wrong type of files. In the worst case, they will crash your terminal and/or make a lot of beeping noises. If this happens to you, just close the terminal session and start a new one. But try to avoid it, because it is usually very disturbing for other people.

3.3.2. Creating and deleting files and directories

3.3.2.1. Making a mess...

... Is not a difficult thing to do. Today almost every system is networked, so naturally files get copied from one machine to another. And especially when working in a graphical environment, creating new files is a piece of cake and is often done without the approval of the user. To illustrate the problem, here's the full content of a new user's directory, created on a standard RedHat system:

```
[newuser@blob user]$ ls -al
total 32
drwx------   3 user     user        4096 Jan 16 13:32 .
drwxr-xr-x   6 root     root        4096 Jan 16 13:32 ..
-rw-r--r--   1 user     user          24 Jan 16 13:32 .bash_logout
-rw-r--r--   1 user     user         191 Jan 16 13:32 .bash_profile
-rw-r--r--   1 user     user         124 Jan 16 13:32 .bashrc
drwxr-xr-x   3 user     user        4096 Jan 16 13:32 .kde
-rw-r--r--   1 user     user        3511 Jan 16 13:32 .screenrc
-rw-------   1 user     user          61 Jan 16 13:32 .xauthDqztLr
```

On first sight, the content of a "used" home directory doesn't look that bad either:

```
olduser:~> ls
app-defaults/ crossover/   Fvwm@     mp3/      OpenOffice.org638/
articles/     Desktop/     GNUstep/  Nautilus/ staroffice6.0/
bin/          Desktop1/    images/   nqc/      training/
brol/         desktoptest/ Machines@ ns_imap/  webstart/
C/            Documents/   mail/     nsmail/   xml/
closed/       Emacs@       Mail/     office52/ Xrootenv.0
```

But when all the directories and files starting with a dot are included, there are 185 items in this directory. This is because most applications have their own directories and/or files, containing user-specific settings, in the home directory of that user. Usually these files are created the first time you start an application. In some cases you will be notified when a non-existent directory needs to be created, but most of the time everything is done automatically.

Furthermore, new files are created seemingly continuously because users want to save files, keep different versions of their work, use Internet applications, and download files and attachments to their local machine. It doesn't stop. It is clear that one definitely needs a scheme to keep an overview on things.

In the next section, we will discuss our means of keeping order. We only discuss text tools available to the shell, since the graphical tools are very intuitive and have the same look and feel as the well known point-and-click MS Windows-style file managers, including graphical help functions and other features you expect from this kind of applications. The following list is an overview of the most popular file managers for GNU/Linux. Most file managers can be started from the menu of your desktop manager, or by clicking your home directory icon, or from the command line, issuing these commands:

- **nautilus**: The default file manager in Gnome, the GNU desktop. Excellent documentation about working with this tool can be found at http://www.gnome.org.
- **konqueror**: The file manager typically used on a KDE desktop. The handbook is at http://docs.kde.org.
- **mc**: Midnight Commander, the Unix file manager after the fashion of Norton Commander. All documentation available from http://gnu.org/directory/ or a mirror, such as http://www.ibiblio.org.

These applications are certainly worth giving a try and usually impress newcomers to Linux, if only because there is such a wide variety: these are only the most popular tools for managing directories and files, and many other projects are being developed. Now let's find out about the internals and see how these graphical tools use common UNIX commands.

3.3.2.2. The tools

3.3.2.2.1. Creating directories

A way of keeping things in place is to give certain files specific default locations by creating directories and subdirectories (or folders and sub-folders if you wish). This is done with the **mkdir** command:

```
richard:~> mkdir archive

richard:~> ls -ld archive
drwxrwxrwx  2 richard richard           4096 Jan 13 14:09 archive/
```

Creating directories and subdirectories in one step is done using the `-p` option:

```
richard:~> cd archive

richard:~/archive> mkdir 1999 2000 2001
```

```
richard:~/archive> ls
1999/  2000/  2001/

richard:~/archive> mkdir 2001/reports/Restaurants-Michelin/
mkdir: cannot create directory `2001/reports/Restaurants-Michelin/':
No such file or directory

richard:~/archive> mkdir -p 2001/reports/Restaurants-Michelin/

richard:~/archive> ls 2001/reports/
Restaurants-Michelin/
```

If the new file needs other permissions than the default file creation permissions, the new access rights can be set in one move, still using the **mkdir** command, see the Info pages for more. We are going to discuss access modes in the next section on file security.

The name of a directory has to comply with the same rules as those applied on regular file names. One of the most important restrictions is that you can't have two files with the same name in one directory (but keep in mind that Linux is, like UNIX, a case sensitive operating system). There are virtually no limits on the length of a file name, but it is usually kept shorter than 80 characters, so it can fit on one line of a terminal. You can use any character you want in a file name, although it is advised to exclude characters that have a special meaning to the shell. When in doubt, check with Appendix C.

3.3.2.2.2. Moving files

Now that we have properly structured our home directory, it is time to clean up unclassified files using the **mv** command:

```
richard:~/archive> mv ../report[1-4].doc reports/Restaurants-Michelin/
```

This command is also applicable when renaming files:

```
richard:~> ls To_Do
-rw-rw-r--    1 richard richard      2534 Jan 15 12:39 To_Do

richard:~> mv To_Do done

richard:~> ls -l done
-rw-rw-r--    1 richard richard      2534 Jan 15 12:39 done
```

It is clear that only the name of the file changes. All other properties remain the same.

Detailed information about the syntax and features of the **mv** command can be found in the man or Info pages. The use of this documentation should always be your first reflex when confronted with a problem. The answer to your problem is likely to be in the system documentation. Even experienced users read man pages every day, so beginning users should read them all the time. After a while, you will get to know the most common options to the common commands, but you will still need the documentation as a primary source of information. Note that the information contained in the HOWTOs, FAQs, man pages and other sources is slowly being merged into the Info pages, which are today the most up-to-date source of online (as in readily available on the system) documentation.

3.3.2.2.3. Copying files

Copying files and directories is done with the **cp** command. A useful option is recursive copy (copy all underlying files and subdirectories), using the `-R` option to **cp**. The general syntax is

cp [-R] `fromfile` `tofile`

3.6.3. Tour of the system

- Change to the `/proc` directory.
- What CPU(s) is the system running on?
- How much RAM does it currently use?
- How much swap space do you have?
- What drivers are loaded?
- How many hours has the system been running?
- Which filesystems are known by your system?
- Change to `/etc/rc.d | /etc/init.d | /etc/runlevels` and choose the directory appropriate for your run level.
- What services should be running in this level?
- Which services run in graphical mode that don't run in text mode?
- Change to `/etc`
- How long does the system keep the log file in which user logins are monitored?
- Which release are you running?
- Are there any issues or messages of the day?
- How many users are defined on your system? Don't count them, let the computer do it for you!
- How many groups?
- Where is the time zone information kept?
- Are the HOWTOs installed on your system?
- Change to `/usr/share/doc`.
- Name three programs that come with the GNU *coreutils* package.
- Which version of **bash** is installed on this system?

3.6.4. Manipulating files

- Create a new directory in your home directory.
- Can you move this directory to the same level as your home directory?
- Copy all XPM files from `/usr/share/pixmaps` to the new directory. What does XPM mean?
- List the files in reverse alphabetical order.
- Change to your home directory. Create a new directory and copy all the files of the `/etc` directory into it. Make sure that you also copy the files and directories which are in the subdirectories of `/etc`! (recursive copy)
- Change into the new directory and make a directory for files starting with an upper case character and one for files starting with a lower case character. Move all the files to the appropriate directories. Use as few commands as possible.
- Remove the remaining files.
- Delete the directory and its entire content using a single command.
- Use **grep** to find out which script starts the Font Server in the graphical run level.
- Where is the *sendmail* server program?
- Make a symbolic link in your home directory to `/var/tmp`. Check that it really works.
- Make another symbolic link in your home directory to this link. Check that it works. Remove the first link and list directory content. What happened to the second link?

3.6.5. File permissions

- Can you change file permissions on `/home`?
- What is your standard file creation mode?
- Change ownership of `/etc` to your own user and group.

- Change file permissions of `~/.bashrc` so that only you and your primary group can read it.
- Issue the command **locate `root`**. Do you notice anything special?
- Make a symbolic link to `/root`. Can it be used?

Chapter 4. Processes

Next to files, processes are the most important things on a UNIX/Linux system. In this chapter, we will take a closer look at those processes. We will learn more about:

- Multi-user processing and multi-tasking
- Process types
- Controlling processes with different signals
- Process attributes
- The life cycle of a process
- System startup and shutdown
- SUID and SGID
- System speed and response
- Scheduling processes
- The Vixie cron system
- How to get the most out of your system

4.1. Processes inside out

4.1.1. Multi-user and multi-tasking

Now that we are more used to our environment and we are able to communicate a little bit with our system, it is time to study the processes we can start in more detail. Not every command starts a single process. Some commands initiate a series of processes, such as **mozilla**; others, like **ls**, are executed as a single command.

Furthermore, Linux is based on UNIX, where it has been common policy to have multiple users running multiple commands, at the same time and on the same system. It is obvious that measures have to be taken to have the CPU manage all these processes, and that functionality has to be provided so users can switch between processes. In some cases, processes will have to continue to run even when the user who started them logs out. And users need a means to reactivate interrupted processes.

We will explain the structure of Linux processes in the next sections.

4.1.2. Process types

4.1.2.1. Interactive processes

Interactive processes are initialized and controlled through a terminal session. In other words, there has to be someone connected to the system to start these processes; they are not started automatically as part of the system functions. These processes can run in the foreground, occupying the terminal that started the program, and you can't start other applications as long as this process is running in the foreground. Alternatively, they can run in the background, so that the terminal in which you started the program can accept new commands while the program is running. Until now, we mainly focussed on programs running in the foreground - the length of time taken to run them was too short to notice - but viewing a file with the **less** command is a good example of a command occupying the terminal session. In this case, the activated program is waiting for you to do something. The program is still connected to the terminal from where it was started, and the terminal is only useful for entering commands this program can understand. Other commands will just result in errors or

unresponsiveness of the system.

While a process runs in the background, however, the user is not prevented from doing other things in the terminal in which he started the program, while it is running.

The shell offers a feature called *job control* which allows easy handling of multiple processes. This mechanism switches processes between the foreground and the background. Using this system, programs can also be started in the background immediately.

Running a process in the background is only useful for programs that don't need user input (via the shell). Putting a job in the background is typically done when execution of a job is expected to take a long time. In order to free the issuing terminal after entering the command, a trailing ampersand is added. In the example, using graphical mode, we open an extra terminal window from the existing one:

```
billy:~> xterm &
[1] 26558

billy:~> jobs
[1]+  Running                 xterm &
```

The full job control features are explained in detail in the **bash** Info pages, so only the frequently used job control applications are listed here:

Table 4-1. Controlling processes

(part of) command	Meaning
regular_command	Runs this command in the foreground.
command &	Run this command in the background (release the terminal)
jobs	Show commands running in the background.
Ctrl+**Z**	Suspend (stop, but not quit) a process running in the foreground (suspend).
Ctrl+**C**	Interrupt (terminate and quit) a process running in the foreground.
%n	Every process running in the background gets a number assigned to it. By using the % expression a job can be referred to using its number, for instance **fg *%2***.
bg	Reactivate a suspended program in the background.
fg	Puts the job back in the foreground.
kill	End a process (also see Shell Builtin Commands in the Info pages of **bash**)

More practical examples can be found in the exercises.

Most UNIX systems are likely to be able to run **screen**, which is useful when you actually want another shell to execute commands. Upon calling **screen**, a new session is created with an accompanying shell and/or commands as specified, which you can then put out of the way. In this new session you may do whatever it is you want to do. All programs and operations will run independent of the issuing shell. You can then detach this session, while the programs you started in it continue to run, even when you log out of the originating shell, and pick your *screen* up again any time you like.

This program originates from a time when virtual consoles were not invented yet, and everything needed to be done using one text terminal. To addicts, it still has meaning in Linux, even though we've had virtual consoles for almost ten years.

```
     |-3*[kdeinit]
     |-keventd
     |-khubd
     |-5*[kjournald]
     |-klogd
     |-lockd---rpciod
     |-lpd
     |-mdrecoveryd
     |-6*[mingetty]
     |-8*[nfsd]
     |-nscd---nscd---5*[nscd]
     |-ntpd
     |-3*[oafd]
     |-panel
     |-portmap
     |-rhnsd
     |-rpc.mountd
     |-rpc.rquotad
     |-rpc.statd
     |-sawfish
     |-screenshooter_a
     |-sendmail
     |-sshd---sshd---bash---su---bash
     |-syslogd
     |-tasklist_applet
     |-vmnet-bridge
     |-xfs
     `-xinetd-ipv6
```

The `-u` and `-a` options give additional information. For more options and what they do, refer to the Info pages.

In the next section, we will see how one process can create another.

4.1.5. Life and death of a process

4.1.5.1. Process creation

A new process is created because an existing process makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different. This procedure is called *forking*.

After the forking process, the address space of the child process is overwritten with the new process data. This is done through an *exec* call to the system.

The *fork-and-exec* mechanism thus switches an old command with a new, while the environment in which the new program is executed remains the same, including configuration of input and output devices, environment variables and priority. This mechanism is used to create all UNIX processes, so it also applies to the Linux operating system. Even the first process, **init**, with process ID 1, is forked during the boot procedure in the so-called *bootstrapping* procedure.

This scheme illustrates the fork-and-exec mechanism. The process ID changes after the fork procedure:

Figure 4-1. Fork-and-exec mechanism

```
mia:~> ls -l /usr/bin/passwd
-r-s--x--x    1 root     root    13476 Aug  7 06:03 /usr/bin/passwd*
```

When called, the **passwd** command will run using the access permissions of *root*, thus enabling a common user to edit the password file which is owned by the system admin.

SGID modes on a file don't occur nearly as frequently as SUID, because SGID often involves the creation of extra groups. In some cases, however, we have to go through this trouble in order to build an elegant solution (don't worry about this too much - the necessary groups are usually created upon installation). This is the case for the **write** and **wall** programs, which are used to send messages to other users' terminals (ttys). The **write** command writes a message to a single user, while **wall** writes to all connected users.

Sending text to another user's terminal or graphical display is normally not allowed. In order to bypass this problem, a group has been created, which owns all terminal devices. When the **write** and **wall** commands are granted SGID permissions, the commands will run using the access rights as applicable to this group, *tty* in the example. Since this group has write access to the destination terminal, also a user having no permissions to use that terminal in any way can send messages to it.

In the example below, user *joe* first finds out on which terminal his correspondent is connected, using the **who** command. Then he sends her a message using the **write** command. Also illustrated are the access rights on the **write** program and on the terminals occupied by the receiving user: it is clear that others than the user owner have no permissions on the device, except for the group owner, which can write to it.

```
joe:~> which write
write is /usr/bin/write

joe:~> ls -l /usr/bin/write
-rwxr-sr-x    1 root     tty      8744 Dec  5 00:55 /usr/bin/write*

joe:~> who
jenny     tty1     Jan 23 11:41
jenny     pts/1    Jan 23 12:21 (:0)
jenny     pts/2    Jan 23 12:22 (:0)
jenny     pts/3    Jan 23 12:22 (:0)
joe       pts/0    Jan 20 10:13 (lo.callhost.org)

joe:~> ls -l /dev/tty1
crw--w----    1 jenny   tty  4,     1 Jan 23 11:41 /dev/tty1

joe:~> write jenny tty1
hey Jenny, shall we have lunch together?
^C
```

User *jenny* gets this on her screen:

```
Message from joe@lo.callhost.org on ptys/1 at 12:36 ...
hey Jenny, shall we have lunch together?
EOF
```

After receiving a message, the terminal can be cleared using the **Ctrl**+**L** key combination. In order to receive no messages at all (except from the system administrator), use the **mesg** command. To see which connected users accept messages from others use **who -w**. All features are fully explained in the Info pages of each command.

Group names may vary

The group scheme is specific to the distribution. Other distributions may use other names or other solutions.

necessary to actually boot that operating system.

GRUB supports both boot methods, allowing you to use it with almost any operating system, most popular file systems, and almost any hard disk your BIOS can recognize.

GRUB contains a number of other features; the most important include:

- GRUB provides a true command-based, pre-OS environment on x86 machines to allow maximum flexibility in loading operating systems with certain options or gathering information about the system.
- GRUB supports Logical Block Addressing (LBA) mode, needed to access many IDE and all SCSI hard disks. Before LBA, hard drives could encounter a 1024-cylinder limit, where the BIOS could not find a file after that point.
- GRUB's configuration file is read from the disk every time the system boots, preventing you from having to write over the MBR every time you change the boot options.

A full description of GRUB may be found by issuing the **info *grub*** command or at the GRUB site. The Linux Documentation Project has a Multiboot with GRUB Mini-HOWTO.

4.2.4. Init

The kernel, once it is loaded, finds **init** in `sbin` and executes it.

When **init** starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing **init** does, is reading its initialization file, `/etc/inittab`. This instructs **init** to read an initial configuration script for the environment, which sets the path, starts swapping, checks the file systems, and so on. Basically, this step takes care of everything that your system needs to have done at system initialization: setting the clock, initializing serial ports and so forth.

Then **init** continues to read the `/etc/inittab` file, which describes how the system should be set up in each run level and sets the default *run level*. A run level is a configuration of processes. All UNIX-like systems can be run in different process configurations, such as the single user mode, which is referred to as run level 1 or run level S (or s). In this mode, only the system administrator can connect to the system. It is used to perform maintenance tasks without risks of damaging the system or user data. Naturally, in this configuration we don't need to offer user services, so they will all be disabled. Another run level is the reboot run level, or run level 6, which shuts down all running services according to the appropriate procedures and then restarts the system.

Use the **who** to check what your current run level is:

```
willy@ubuntu:~$ who -r
        run-level 2 2006-10-17 23:22            last=S
```

More about run levels in the next section, see Section 4.2.5.

After having determined the default run level for your system, **init** starts all of the background processes necessary for the system to run by looking in the appropriate `rc` directory for that run level. **init** runs each of the kill scripts (their file names start with a K) with a stop parameter. It then runs all of the start scripts (their file names start with an S) in the appropriate run level directory so that all services and applications are started correctly. In fact, you can execute these same scripts manually after the system is finished booting with a command like **/etc/init.d/httpd *stop*** or **service *httpd stop*** logged in as *root*, in this case stopping the

The GNU **time** command in `/usr/bin` (as opposed to the shell built-in version) displays more information that can be formatted in different ways. It also shows the exit status of the command, and the total elapsed time. The same command as the above using the independent **time** gives this output:

```
tilly:~/xml/src> /usr/bin/time make
Output written on abook.pdf (222 pages, 1595027 bytes).
Transcript written on abook.log.

Command exited with non-zero status 2
88.87user 1.74system 1:36.21elapsed 94%CPU
                                (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (2192major+30002minor)pagefaults 0swaps
```

Refer again to the Info pages for all the information.

4.3.3. Performance

To a user, performance means quick execution of commands. To a system manager, on the other hand, it means much more: the system admin has to optimize system performance for the whole system, including users, all programs and daemons. System performance can depend on a thousand tiny things which are not accounted for with the **time** command:

- the program executing is badly written or doesn't use the computer appropriately
- access to disks, controllers, display, all kinds of interfaces, etc.
- reachability of remote systems (network performance)
- amount of users on the system, amount of users actually working simultaneously
- time of day
- ...

4.3.4. Load

In short: the load depends on what is normal for your system. My old P133 running a firewall, SSH server, file server, a route daemon, a sendmail server, a proxy server and some other services doesn't complain with 7 users connected; the load is still 0 on average. Some (multi-CPU) systems I've seen were quite happy with a load of 67. There is only one way to find out - check the load regularly if you want to know what's normal. If you don't, you will only be able to measure system load from the response time of the command line, which is a very rough measurement since this speed is influenced by a hundred other factors.

Keep in mind that different systems will behave different with the same load average. For example, a system with a graphics card supporting hardware acceleration will have no problem rendering 3D images, while the same system with a cheap VGA card will slow down tremendously while rendering. My old P133 will become quite uncomfortable when I start the X server, but on a modern system you hardly notice the difference in the system load.

4.3.5. Can I do anything as a user?

A big environment can slow you down. If you have lots of environment variables set (instead of shell variables), long search paths that are not optimized (errors in setting the path environment variable) and more of those settings that are usually made "on the fly", the system will need more time to search and read data.

In X, window managers and desktop environments can be real CPU-eaters. A really fancy desktop comes with a price, even when you can download it for free, since most desktops provide add-ons ad infinitum. Modesty

is a virtue if you don't buy a new computer every year.

4.3.5.1. Priority

The priority or importance of a job is defined by it's *nice* number. A program with a high nice number is friendly to other programs, other users and the system; it is not an important job. The lower the nice number, the more important a job is and the more resources it will take without sharing them.

Making a job nicer by increasing its nice number is only useful for processes that use a lot of CPU time (compilers, math applications and the like). Processes that always use a lot of I/O time are automatically rewarded by the system and given a higher priority (a lower nice number), for example keyboard input always gets highest priority on a system.

Defining the priority of a program is done with the **nice** command.

Most systems also provide the BSD **renice** command, which allows you to change the *niceness* of a running command. Again, read the man page for your system-specific information.

Interactive programs

It is NOT a good idea to **nice** or **renice** an interactive program or a job running in the foreground. Use of these commands is usually a task for the system administrator. Read the man page for more info on extra functionality available to the system administrator.

4.3.5.2. CPU resources

On every Linux system, many programs want to use the CPU(s) at the same time, even if you are the only user on the system. Every program needs a certain amount of cycles on the CPU to run. There may be times when there are not enough cycles because the CPU is too busy. The **uptime** command is wildly inaccurate (it only displays averages, you have to know what is normal), but far from being useless. There are some actions you can undertake if you think your CPU is to blame for the unresponsiveness of your system:

- Run heavy programs when the load is low. This may be the case on your system during the night. See next section for scheduling.
- Prevent the system from doing unnecessary work: stop daemons and programs that you don't use, use **locate** instead of a heavy **find**, ...
- Run big jobs with a low priority

If none of these solutions are an option in your particular situation, you may want to upgrade your CPU. On a UNIX machine this is a job for the system admin.

4.3.5.3. Memory resources

When the currently running processes expect more memory than the system has physically available, a Linux system will not crash; it will start paging, or *swapping*, meaning the process uses the memory on disk or in swap space, moving contents of the physical memory (pieces of running programs or entire programs in the case of swapping) to disk, thus reclaiming the physical memory to handle more processes. This slows the system down enormously since access to disk is much slower than access to memory. The **top** command can be used to display memory and swap use. Systems using glibc offer the **memusage** and **memusagestat** commands to visualize memory usage.

4.3.5.5. Users

Users can be divided in several classes, depending on their behavior with resource usage:

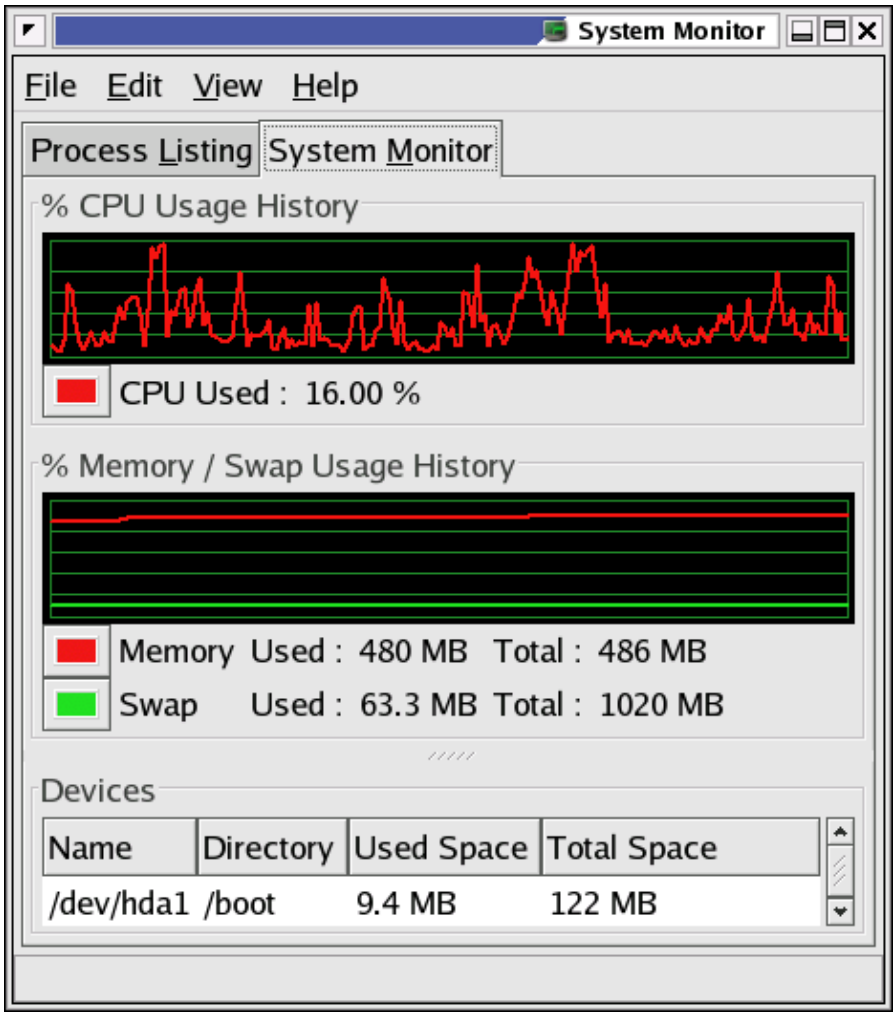
- Users who run a (large) number of small jobs: you, the beginning Linux user, for instance.
- Users who run relatively few but large jobs: users running simulations, calculations, emulators or other programs that eat a lot of memory, and usually these users have accompanying large data files.
- Users who run few jobs but use a lot of CPU time (developers and the like).

You can see that system requirements may vary for each class of users, and that it can be hard to satisfy everyone. If you are on a multi-user system, it is useful (and fun) to find out habits of other users and the system, in order to get the most out of it for your specific purposes.

4.3.5.6. Graphical tools

For the graphical environment, there are a whole bunch of monitoring tools available. Below is a screen shot of the Gnome System Monitor, which has features for displaying and searching process information, and monitoring system resources:

Figure 4-3. Gnome System Monitor



There are also a couple of handy icons you can install in the task bar, such as a disk, memory and load monitor. **xload** is another small X application for monitoring system load. Find your favorite!

4.3.5.7. Interrupting your processes

As a non-privileged user, you can only influence your own processes. We already saw how you can display processes and filter out processes that belong to a particular user, and what possible restrictions can occur. When you see that one of your processes is eating too much of the system's resources, there are two things that you can do:

1. Make the process use less resources without interrupting it;
2. Stop the process altogether.

In the case that you want the process to continue to run, but you also want to give the other processes on the system a chance, you can **renice** the process. Appart from using the **nice** or **renice** commands, **top** is an easy way of spotting the troublesome process(es) and reducing priority.

Identify the process in the "NI" column, it will most likely have a negative priority. Type **r** and enter the process ID of the process that you want to renice. Then enter the nice value, for instance "20". That means that from now on, this process will take 1/5 of the CPU cycles at the most.

Examples of processes that you want to keep on running are emulators, virtual machines, compilers and so on.

If you want to stop a process because it hangs or is going totally berserk in the way of I/O consumption, file creation or use of other system resources, use the **kill** command. If you have the opportunity, first try to kill the process softly, sending it the *SIGTERM* signal. This is an instruction to terminate whatever it is doing, according to procedures as described in the code of the program:

```
joe:~> ps -ef | grep mozilla
joe    25822    1  0 Mar11 ?    00:34:04 /usr/lib/mozilla-1.4.1/mozilla-

joe:~> kill -15 25822
```

In the example above, user *joe* stopped his Mozilla browser because it hung.

Some processes are a little bit harder to get rid of. If you have the time, you might want to send them the SIGINT signal to interrupt them. If that does not do the trick either, use the strongest signal, SIGKILL. In the example below, *joe* stops a Mozilla that is frozen:

```
joe:~> ps -ef | grep mozilla
joe    25915    1  0 Mar11 ?    00:15:06 /usr/lib/mozilla-1.4.1/mozilla-

joe:~> kill -9 25915

joe:~> ps -ef | grep 25915
joe     2634 32273 0 18:09 pts/4   00:00:00 grep 25915
```

In such cases, you might want to check that the process is really dead, using the **grep** filter again on the PID. If this only returns the **grep** process, you can be sure that you succeeded in stopping the process.

Among processes that are hard to kill is your shell. And that is a good thing: if they would be easy to kill, you woud loose your shell every time you type **Ctrl-C** on the command line accidentally, since this is equivalent to sending a SIGINT.

UNIX without pipes is almost unthinkable

The usage of pipes (|) for using output of one command as input of another is explained in the next chapter, Chapter 5.

In a graphical environment, the **xkill** program is very easy to use. Just type the name of the command, followed by an **Enter** and select the window of the application that you want to stop. It is rather dangerous because it sends a SIGKILL by default, so only use it when an application hangs.

4.4. Scheduling processes

4.4.1. Use that idle time!

A Linux system can have a lot to suffer from, but it usually suffers only during office hours. Whether in an office environment, a server room or at home, most Linux systems are just idling away during the morning, the evening, the nights and weekends. Using this idle time can be a lot cheaper than buying those machines you'd absolutely need if you want everything done at the same time.

There are three types of delayed execution:

- Waiting a little while and then resuming job execution, using the **sleep** command. Execution time depends on the system time at the moment of submission.
- Running a command at a specified time, using the **at** command. Execution of the job(s) depends on system time, not the time of submission.
- Regularly running a command on a monthly, weekly, daily or hourly basis, using the **cron** facilities.

The following sections discuss each possibility.

4.4.2. The sleep command

The Info page on sleep is probably one of the shortest there is. All **sleep** does is wait. By default the time to wait is expressed in seconds.

So why does it exist? Some practical examples:

Somebody calls you on the phone, you say "Yes I'll be with you in half an hour" but you're about drowned in work as it is and bound to forget your lunch:

(sleep *1800*; echo *"Lunch time.."*) &

When you can't use the **at** command for some reason, it's five o'clock, you want to go home but there's still work to do and right now somebody is eating system resources:

(sleep *10000*; myprogram) &

Make sure there's an auto-logout on your system, and that you log out or lock your desktop/office when submitting this kind of job, or run it in a **screen** session.

When you run a series of printouts of large files, but you want other users to be able to print in between:

lp `lotoftext`; sleep *900*; lp `hugefile`; sleep *900*; lp `anotherlargefile`

Printing files is discussed in Chapter 8.

Programmers often use the sleep command to halt script or program execution for a certain time.

4.4.3. The at command

The **at** command executes commands at a given time, using your default shell unless you tell the command otherwise (see the man page).

The options to **at** are rather user-friendly, which is demonstrated in the examples below:

```
steven@home:~> at tomorrow + 2 days
warning: commands will be executed using (in order) a) $SHELL
        b) login shell c) /bin/sh
at>  cat reports | mail myboss@mycompany
at> <EOT>
job 1 at 2001-06-16 12:36
```

Typing **Ctrl**+**D** quits the **at** utility and generates the "EOT" message.

User *steven* does a strange thing here combining two commands; we will study this sort of practice in Chapter 5, Redirecting Input and Output.

```
steven@home:~> at 0237
warning: commands will be executed using (in order) a) $SHELL
        b) login shell c) /bin/sh
at>  cd new-programs
at>  ./configure; make
at> <EOT>
job 2 at 2001-06-14 02:00
```

The `-m` option sends mail to the user when the job is done, or explains when a job can't be done. The command **atq** lists jobs; perform this command before submitting jobs in order prevent them from starting at the same time as others. With the **atrm** command you can remove scheduled jobs if you change your mind.

It is a good idea to pick strange execution times, because system jobs are often run at "round" hours, as you can see in Section 4.4.4 the next section. For example, jobs are often run at exactly 1 o'clock in the morning (e.g. system indexing to update a standard locate database), so entering a time of 0100 may easily slow your system down rather than fire it up. To prevent jobs from running all at the same time, you may also use the **batch** command, which queues processes and feeds the work in the queue to the system in an evenly balanced way, preventing excessive bursts of system resource usage. See the Info pages for more information.

4.4.4. Cron and crontab

The cron system is managed by the **cron** daemon. It gets information about which programs and when they should run from the system's and users' crontab entries. Only the root user has access to the system crontabs, while each user should only have access to his own crontabs. On some systems (some) users may not have access to the cron facility.

At system startup the cron daemon searches `/var/spool/cron/` for crontab entries which are named after accounts in `/etc/passwd`, it searches `/etc/cron.d/` and it searches `/etc/crontab`, then uses this information every minute to check if there is something to be done. It executes commands as the user who owns the crontab file and mails any output of commands to the owner.

5.2.2. Examples

5.2.2.1. Analyzing errors

If your process generates a lot of errors, this is a way to thoroughly examine them:

command 2>&1 | less

This is often used when creating new software using the **make** command, such as in:

```
andy:~/newsoft> make all 2>&1 | less
--output ommitted--
```

5.2.2.2. Separating standard output from standard error

Constructs like these are often used by programmers, so that output is displayed in one terminal window, and errors in another. Find out which pseudo terminal you are using issuing the **tty** command first:

```
andy:~/newsoft> make all 2> /dev/pts/7
```

5.2.2.3. Writing to output and files simultaneously

You can use the **tee** command to copy input to standard output and one or more output files in one move. Using the `-a` option to **tee** results in appending input to the file(s). This command is useful if you want to both see and save output. The > and >> operators do not allow to perform both actions simultaneously.

This tool is usually called on through a pipe (|), as demonstrated in the example below:

```
mireille ~/test> date | tee file1 file2
Thu Jun 10 11:10:34 CEST 2004

mireille ~/test> cat file1
Thu Jun 10 11:10:34 CEST 2004

mireille ~/test> cat file2
Thu Jun 10 11:10:34 CEST 2004

mireille ~/test> uptime | tee -a file2
 11:10:51 up 21 days, 21:21, 57 users,  load average: 0.04, 0.16, 0.26

mireille ~/test> cat file2
Thu Jun 10 11:10:34 CEST 2004
 11:10:51 up 21 days, 21:21, 57 users,  load average: 0.04, 0.16, 0.26
```

5.3. Filters

When a program performs operations on input and writes the result to the standard output, it is called a filter. One of the most common uses of filters is to restructure output. We'll discuss a couple of the most important filters below.

- List the devices in `/dev` which are currently used by your UID. Pipe through **less** to view them properly.
- Issue the following commands as a non-privileged user. Determine standard input, output and error for each command.

    - **cat `nonexistentfile`**
    - **file `/sbin/ifconfig`**
    - **grep *`root`* `/etc/passwd /etc/nofiles` > `grepresults`**
    - **/etc/init.d/sshd *`start`* > `/var/tmp/output`**
    - **/etc/init.d/crond *`start`* > `/var/tmp/output` 2>&1**
    - Now check your results by issuing the commands again, now redirecting standardoutput to the file `/var/tmp/output` and standard error to the file `/var/tmp/error`.
- How many processes are you currently running?
- How many invisible files are in your home directory?
- Use **locate** to find documentation about the kernel.
- Find out which file contains the following entry:

```
root:x:0:0:root:/root:/bin/bash
```

  And this one:

```
system:         root
```

- See what happens upon issuing this command:

  > time; date >> time; cat < time
- What command would you use to check which script in `/etc/init.d` starts a given process?

7.1.2.3. Mail

Regularly clean out your mailbox, make sub-folders and automatic redirects using **procmail** (see the Info pages) or the filters of your favorite mail reading application. If you have a trash folder, clean it out on a regular basis.

To redirect mail, use the `.forward` file in your home directory. The Linux mail service looks for this file whenever it has to deliver local mail. The content of the file defines what the mail system should do with your mail. It can contain a single line holding a fully qualified E-mail address. In that case the system will send all your mail to this address. For instance, when renting space for a website, you might want to forward the mail destined for the webmaster to your own account in order not to waste disk space. The webmaster's `.forward` may look like this:

```
webmaster@www ~/> cat .forward
mike@pandora.be
```

Using mail forwarding is also useful to prevent yourself from having to check several different mailboxes. You can make every address point to a central and easily accessible account.

You can ask your system administrator to define a forward for you in the local mail aliases file, like when an account is being closed but E-mail remains active for a while.

7.1.2.4. Save space with a link

When several users need access to the same file or program, when the original file name is too long or too difficult to remember, use a symbolic link instead of a separate copy for each user or purpose.

Multiple symbolic links may have different names, e.g. a link may be called `monfichier` in one user's directory, and `mylink` in another's. Multiple links (different names) to the same file may also occur in the same directory. This is often done in the `/lib` directory: when issuing the command

ls -l /lib

you will see that this directory is plenty of links pointing to the same files. These are created so that programs searching for one name would not get stuck, so they are pointed to the correct/current name of the libraries they need.

7.1.2.5. Limit file sizes

The shell contains a built-in command to limit file sizes, **ulimit**, which can also be used to display limitations on system resources:

```
cindy:~> ulimit -a
core file size (blocks)     0
data seg size (kbytes)      unlimited
file size (blocks)          unlimited
max locked memory (kbytes)  unlimited
max memory size (kbytes)    unlimited
open files                  1024
pipe size (512 bytes)       8
stack size (kbytes)         8192
cpu time (seconds)          unlimited
max user processes          512
virtual memory (kbytes)     unlimited
```

Cindy is not a developer and doesn't care about core dumps, which contain debugging information on a program. If you do want core dumps, you can set their size using the **ulimit** command. Read the Info pages on **bash** for a detailed explanation.

Core file?

A core file or *core dump* is sometimes generated when things go wrong with a program during its execution. The core file contains a copy of the system's memory, as it was at the time that the error occured.

7.1.2.6. Compressed files

Compressed files are useful because they take less space on your hard disk. Another advantage is that it takes less bandwidth to send a compressed file over your network. A lot of files, such as the man pages, are stored in a compressed format on your system. Yet unpacking these to get a little bit of information and then having to compress them again is rather time-consuming. You don't want to unpack a man page, for instance, read about an option to a command and then compress the man page again. Most people will probably forget to clean up after they found the information they needed.

So we have tools that work on compressed files, by uncompressing them only in memory. The actual compressed file stays on your disk as it is. Most systems support **zgrep**, **zcat**, **bzless** and other members of the z-family to prevent unnecessary decompressing/compressing actions. See your system's binary directory and the Info pages.

See Chapter 9 for more on the actual compressing of files and examples on making archives.

7.2. Your text environment

7.2.1. Environment variables

7.2.1.1. General

We already mentioned a couple of environment variables, such as `PATH` and `HOME`. Until now, we only saw examples in which they serve a certain purpose to the shell. But there are many other Linux utilities that need information about you in order to do a good job.

What other information do programs need apart from paths and home directories?

A lot of programs want to know about the kind of terminal you are using; this information is stored in the `TERM` variable. In text mode, this will be the *linux* terminal emulation, in graphical mode you are likely to use *xterm*. Lots of programs want to know what your favorite editor is, in case they have to start an editor in a subprocess. The shell you are using is stored in the `SHELL` variable, the operating system type in `OS` and so on. A list of all variables currently defined for your session can be viewed entering the **printenv** command.

The environment variables are managed by the shell. As opposed to regular shell variables, environment variables are inherited by any program you start, including another shell. New processes are assigned a copy of these variables, which they can read, modify and pass on in turn to their own child processes.

There is nothing special about variable names, except that the common ones are in upper case characters by convention. You may come up with any name you want, although there are standard variables that are

important enough to be the same on every Linux system, such as PATH and HOME.

7.2.1.2. Exporting variables

An individual variable's content is usually displayed using the **echo** command, as in these examples:

```
debby:~> echo $PATH
/usr/bin:/usr/sbin:/bin:/sbin:/usr/X11R6/bin:/usr/local/bin

debby:~> echo $MANPATH
/usr/man:/usr/share/man/:/usr/local/man:/usr/X11R6/man
```

If you want to change the content of a variable in a way that is useful to other programs, you have to export the new value from your environment into the environment that runs these programs. A common example is exporting the PATH variable. You may declare it as follows, in order to be able to play with the flight simulator software that is in /opt/FlightGear/bin:

```
debby:~> PATH=$PATH:/opt/FlightGear/bin
```

This instructs the shell to not only search programs in the current path, $PATH, but also in the additional directory /opt/FlightGear/bin.

However, as long as the new value of the PATH variable is not known to the environment, things will still not work:

```
debby:~> runfgfs
bash: runfgfs: command not found
```

Exporting variables is done using the shell built-in command **export**:

```
debby:~> export PATH

debby:~> runfgfs
--flight simulator starts--
```

In Bash, we normally do this in one elegant step:

export VARIABLE=*value*

The same technique is used for the MANPATH variable, that tells the **man** command where to look for compressed man pages. If new software is added to the system in new or unusual directories, the documentation for it will probably also be in an unusual directory. If you want to read the man pages for the new software, extend the MANPATH variable:

```
debby:~> export MANPATH=$MANPATH:/opt/FlightGear/man

debby:~> echo $MANPATH
/usr/man:/usr/share/man:/usr/local/man:/usr/X11R6/man:/opt/FlightGear/man
```

You can avoid retyping this command in every window you open by adding it to one of your shell setup files, see Section 7.2.2.

7.2.1.3. Reserved variables

The following table gives an overview of the most common predefined variables:

Table 7-1. Common environment variables

Variable name	Stored information
DISPLAY	used by the X Window system to identify the display server
DOMAIN	domain name
EDITOR	stores your favorite line editor
HISTSIZE	size of the shell history file in number of lines
HOME	path to your home directory
HOSTNAME	local host name
INPUTRC	location of definition file for input devices such as keyboard
LANG	preferred language
LD_LIBRARY_PATH	paths to search for libraries
LOGNAME	login name
MAIL	location of your incoming mail folder
MANPATH	paths to search for man pages
OS	string describing the operating system
OSTYPE	more information about version etc.
PAGER	used by programs like **man** which need to know what to do in case output is more than one terminal window.
PATH	search paths for commands
PS1	primary prompt
PS2	secondary prompt
PWD	present working directory
SHELL	current shell
TERM	terminal type
UID	user ID
USER(NAME)	user name
VISUAL	your favorite full-screen editor
XENVIRONMENT	location of your personal settings for X behavior
XFILESEARCHPATH	paths to search for graphical libraries

A lot of variables are not only predefined but also preset, using configuration files. We discuss these in the next section.

7.2.2. Shell setup files

When entering the **ls `-al`** command to get a long listing of all files, including the ones starting with a dot, in your home directory, you will see one or more files starting with a . and ending in *rc*. For the case of **bash**, this is `.bashrc`. This is the counterpart of the system-wide configuration file `/etc/bashrc`.

When logging into an interactive login shell, **login** will do the authentication, set the environment and start your shell. In the case of **bash**, the next step is reading the general `profile` from `/etc`, if that file exists. **bash** then looks for `~/.bash_profile`, `~/.bash_login` and `~/.profile`, in that order, and reads and executes commands from the first one that exists and is readable. If none exists, `/etc/bashrc` is applied.

Many other distributions support RPM packages, among the popular ones RedHat Enterprise Linux, Mandriva (former Mandrake), Fedora Core and SuSE Linux. Apart from the advice for your distribution, you will want to read **man *rpm***.

7.5.2.1.2. RPM examples

Most packages are simply installed with the upgrade option, `-U`, whether the package is already installed or not. The RPM package contains a complete version of the program, which overwrites existing versions or installs as a new package. The typical usage is as follows:

rpm -Uvh /path/to/rpm-package(s)

The `-v` option generates more verbose output, and `-h` makes **rpm** print a progress bar:

```
[root@jupiter tmp]# rpm -Uvh totem-0.99.5-1.fr.i386.rpm
Preparing...                ########################################### [100%]
   1:totem                  ########################################### [100%]
[root@jupiter tmp]#
```

New kernel packages, however, are installed with the install option `-i`, which does not overwrite existing version(s) of the package. That way, you will still be able to boot your system with the old kernel if the new one does not work.

You can also use **rpm** to check whether a package is installed on your system:

```
[david@jupiter ~] rpm -qa | grep vim
vim-minimal-6.1-29
vim-X11-6.1-29
vim-enhanced-6.1-29
vim-common-6.1-29
```

Or you can find out which package contains a certain file or executable:

```
[david@jupiter ~] rpm -qf /etc/profile
setup-2.5.25-1

[david@jupiter ~] which cat
cat is /bin/cat

[david@jupiter ~] rpm -qf /bin/cat
coreutils-4.5.3-19
```

Note that you need not have access to administrative privileges in order to use **rpm** to query the RPM database. You only need to be *root* when adding, modifying or deleting packages.

Below is one last example, demonstrating how to uninstall a package using **rpm**:

```
[root@jupiter root]# rpm -e totem
[root@jupiter root]#
```

Note that uninstalling is not that verbose by default, it is normal that you don't see much happening. When in doubt, use **rpm -qa** again to verify that the package has been removed.

RPM can do much more than the couple of basic functions we discussed in this introduction; the RPM HOWTO contains further references.

7.5.2.2. DEB (.deb) packages

7.5.2.2.1. What are Debian packages?

This package format is the default on Debian GNU/Linux, where **dselect**, and, nowadays more common, **aptitude**, is the standard tool for managing the packages. It is used to select packages that you want to install or upgrade, but it will also run during the installation of a Debian system and help you to define the access method to use, to list available packages and to configure packages.

The Debian web site contains all information you need, including a "dselect Documentation for Beginners".

According to the latest news, the Debian package format is becoming more and more popular. At the time of this writing, 5 of the top-10 distributions use it. Also **apt-get** (see Section 7.5.3.2 is becoming extremely popular, also on non-DEB systems.

7.5.2.2.2. Examples with DEB tools

Checking whether a package is installed is done using the **dpkg** command. For instance, if you want to know which version of the Gallery software is installed on your machine:

```
nghtwsh@gorefest:~$ dpkg -l *gallery*
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============-============================================
ii  gallery        1.5-1sarge2    a web-based photo album written in php
```

The "ii" prefix means the package is installed. Should you see "un" as a prefix, that means that the package is known in the list that your computer keeps, but that it is not installed.

Searching which package a file belongs to is done using the `-S` to **dpkg**:

```
nghtwsh@gorefest:~$ dpkg -S /bin/cat
coreutils: /bin/cat
```

More information can be found in the Info pages for **dpkg**.

7.5.2.3. Source packages

The largest part of Linux programs is Free/Open Source, so source packages are available for these programs. Source files are needed for compiling your own program version. Sources for a program can be downloaded from its web site, often as a compressed tarball (`program-version.tar.gz` or similar). For RPM-based distributions, the source is often provided in the `program-version.src.rpm`. Debian, and most distributions based on it, provide themselves the adapted source which can be obtained using **apt-get *source***.

Specific requirements, dependencies and installation instructions are provided in the `README` file. You will probably need a C compiler, **gcc**. This GNU C compiler is included in most Linux systems and is ported to many other platforms.

7.5.3. Automating package management and updates

7.5.3.1. General remarks

The first thing you do after installing a new system is applying updates; this applies to all operating systems and Linux is not different.

The updates for most Linux systems can usually be found on a nearby site mirroring your distribution. Lists of sites offering this service can be found at your distribution's web site, see Appendix A.

Updates should be applied regularly, daily if possible - but every couple of weeks would be a reasonable start. You really should try to have the most recent version of your distribution, since Linux changes constantly. As we said before, new features, improvements and bug fixes are supplied at a steady rhythm, and sometimes important security problems are addressed.

The good news is that most Linux distributions provide tools so that you don't have to upgrade tens of packages daily by hand. The following sections give an overview of *package manager managers.* There is much more to this subject, even regular updates of source packages is manageable automatically; we only list the most commonly known systems. Always refer to the documentation for your specific distribution for advised procedures.

7.5.3.2. APT

The Advanced Package Tool is a management system for software packages. The command line tool for handling packages is **apt-get**, which comes with an excellent man page describing how to install and update packages and how to upgrade singular packages or your entire distribution. APT has its roots in the Debian GNU/Linux distribution, where it is the default manager for the Debian packages. APT has been ported to work with RPM packages as well. The main advantage of APT is that it is free and flexible to use. It will allow you to set up systems similar to the distribution specific (and in some cases commercial) ones listed in the next sections.

Generally, when first using **apt-get**, you will need to get an index of the available packages. This is done using the command

apt-get *update*

After that, you can use **apt-get** to upgrade your system:

apt-get *upgrade*

Do this often, it's an easy way to keep your system up-to-date and thus safe.

Apart from this general usage, **apt-get** is also very fast for installing individual packages. This is how it works:

```
[david@jupiter ~] su - -c "apt-get install xsnow"
Password:
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  xsnow
```

```
0 packages upgraded, 1 newly installed, 0 removed and 3 not upgraded.
Need to get 33.6kB of archives.
After unpacking 104kB of additional disk space will be used.
Get:1 http://ayo.freshrpms.net redhat/9/i386/os xsnow 1.42-10 [33.6kB]
Fetched 33.6kB in 0s (106kB/s)
Executing RPM (-Uvh)...
Preparing...                ########################################### [100%]
   1:xsnow                  ########################################### [100%]
```

Note the `-c` option to the **su** command, which indicates to the root shell to only execute this command, and then return to the user's environment. This way, you cannot forget to quit the root account.

If there are any dependencies on other packages, **apt-get** will download and install these supporting packages.

More information can be found in the APT HOWTO.

7.5.3.3. Systems using RPM packages

Update Agent, which originally only supported RedHat RPM packages, is now ported to a wider set of software, including non-RedHat repositories. This tool provides a complete system for updating the RPM packages on a RedHat or Fedora Core system. On the command line, type **up2date** to update your system. On the desktop, by default a small icon is activated, telleng you whether or not there are updates available for your system.

Yellowdog's Updater Modified (**yum**) is another tool that recently became more popular. It is an interactive but automated update program for installing, updating or removing RPM packages on a system. It is the tool of choice on Fedora systems.

On SuSE Linux, everything is done with YaST, Yet another Setup Tool, which supports a wide variety of system administration tasks, among which updating RPM packages. Starting from SuSE Linux 7.1 you can also upgrade using a web interface and YOU, Yast Online Update.

Mandrake Linux and Mandriva provide so-called URPMI tools, a set of wrapper programs that make installing new software easier for the user. These tools combine with RPMDrake and MandrakeUpdate to provide everything needed for smooth install and uninstall of software packages. MandrakeOnline offers an extended range of services and can automatically notify administrators when updates are available for your particular Mandrake system. See **man *urpmi***, among others, for more info.

Also the KDE and Gnome desktop suites have their own (graphical) versions of package managers.

7.5.4. Upgrading your kernel

Most Linux installations are fine if you periodically upgrade your distribution. The upgrade procedure will install a new kernel when needed and make all necessary changes to your system. You should only compile or install a new kernel manually if you need kernel features that are not supported by the default kernel included in your Linux distribution.

Whether compiling your own optimized kernel or using a pre-compiled kernel package, install it in co-existence with the old kernel until you are sure that everything works according to plan.

Then create a dual boot system that will allow you to choose which kernel to boot by updating your boot loader configuration file `grub.conf`. This is a simple example:

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making config changes.
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, e.g.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/hde8
#          initrd /initrd-version.img
#boot=/dev/hde
default=0
timeout=10
splashimage=(hd0,0)/grub/splash.xpm.gz
title Red Hat Linux new (2.4.9-31)
        root (hd0,0)
        kernel /vmlinuz-2.4.9-31 ro root=/dev/hde8
        initrd /initrd-2.4.9-31.img
title old-kernel
        root (hd0,0)
        kernel /vmlinuz-2.4.9-21 ro root=/dev/hde8
        initrd /initrd-2.4.9-21.img
```

After the new kernel has proven to work, you may remove the lines for the old one from the GRUB config file, although it is best to wait a couple of days just to be sure.

7.5.5. Installing extra packages from the installation CDs

7.5.5.1. Mounting a CD

This is basically done in the same way as installing packages manually, except that you have to append the file system of the CD to your machine's file system to make it accessible. On most systems, this will be done automatically upon insertion of a CD in the drive because the **automount** daemon is started up at boot time. If your CD is not made available automatically, issue the **mount** command in a terminal window. Depending on your actual system configuration, a line similar to this one will usually do the trick:

mount `/dev/cdrom /mnt/cdrom`

On some systems, only *root* can mount removable media; this depends on the configuration.

For automation purposes, the CD drive usually has an entry in `/etc/fstab`, which lists the file systems and their mount points, that make up your file system tree. This is such a line:

```
[david@jupiter ~] grep cdrom /etc/fstab
/dev/cdrom      /mnt/cdrom      iso9660         noauto,owner,ro 0 0
```

This indicates that the system will understand the command **mount `/mnt/cdrom`**. The `noauto` option means that on this system, CDs are not mounted at boot time.

You may even try to right click on the CD icon on your desktop to mount the CD if your file manager doesn't do it for you. You can check whether it worked issuing the **mount** command with no arguments:

```
[david@jupiter ~] mount | grep cdrom
/dev/cdrom on /mnt/cdrom type iso9660 (ro,nosuid,nodev)
```

7.7.2. Graphical environment

- Try all the mouse buttons in different regions (terminal, background, task bar).
- Explore the menus.
- Customize your terminal window.
- Use the mouse buttons to copy and paste text from one terminal to another.
- Find out how to configure your window manager; try different workspaces (virtual screens).
- Add an applet, such as a load monitor, to the task bar.
- Apply a different theme.
- Enable the so-called *sloppy* focus - this is when a window is activated by just moving the mouse over it, so that you do not need to click the window in order to be able to use it.
- Switch to a different window manager.
- Log out and select a different session type, like KDE if you were using Gnome before. Repeat the previous steps.

Apart from these command line tools there are a lot of graphical word processing programs. Several complete office suites are available, many are free. These do the formatting automatically upon submission of a print job. Just to name a few: OpenOffice.org, KOffice, AbiWord, WordPerfect, etc.

The following are common languages in a printing context:

- **groff**: GNU version of the UNIX **roff** command. It is a front-end to the groff document formatting system. Normally it runs the **troff** command and a post-processor appropriate for the selected device. It allows generation of PostScript files.
- TeX and the macro package LaTeX: one of the most widely used markup languages on UNIX systems. Usually invoked as **tex**, it formats files and outputs a corresponding device-independent representation of the typeset document.

  Technical works are *still* frequently written in LaTeX because of its support for mathematic formulas, although efforts are being made at W3C (the World Wide Web Consortium) to include this feature in other applications.
- SGML and XML: Free parsers are available for UNIX and Linux. XML is the next generation SGML, it forms the basis for DocBook XML, a document system (this book is written in XML, for instance).

Printing documentation

The man pages contain pre-formatted **troff** data which has to be formatted before it can roll out of your printer. Printing is done using the `-t` option to the **man** command:

man `-t` *`command`* > `man-command.ps`

Then print the PostScript file. If a default print destination is configured for your system/account, you can just issue the command **man `-t` *`command`*** to send the formatted page to the printer directly.

8.1.2.2. Previewing formatted files

Anything that you can send to the printer can normally be sent to the screen as well. Depending on the file format, you can use one of these commands:

- PostScript files: with the **gv** (GhostView) command.
- TeX dvi files: with **xdvi**, or with KDE's **kdvi**.
- PDF files: **xpdf**, **kpdf**, **gpdf** or Adobe's viewer, **acroread**, which is also available for free but is not free software. Adobe's reader supports PDF 1.6, the others only support PDF versions up to 1.5. The version of a PDF file can be determined using the **file** command.
- From within applications, such as Firefox or OpenOffice, you can usually select Print Preview from one of the menus.

8.2. The server side

8.2.1. General

Until a couple of years ago, the choice for Linux users was simple: everyone ran the same old LPD from BSD's Net-2 code. Then LPRng became more popular, but nowadays most modern Linux distributions use CUPS, the Common UNIX Printing System. CUPS is an implementation of the Internet Printing Protocol (IPP), an HTTP-like RFC standard replacement protocol for the venerable (and clunky) LPD protocol. CUPS

is distributed under the GNU Public License. CUPS is also the default print system on MacOS X.

8.2.2. Graphical printer configuration

Most distributions come with a GUI for configuring networked and local (parallel port or USB) printers. They let you choose the printer type from a list and allow easy testing. You don't have to bother about syntax and location of configuration files. Check your system documentation before you attempt installing your printer.

CUPS can also be configured using a web interface that runs on port 631 on your computer. To check if this feature is enabled, try browsing to localhost:631/help or localhost:631/.

8.2.3. Buying a printer for Linux

As more and more printer vendors make drivers for CUPS available, CUPS will allow easy connection with almost any printer that you can plug into a serial, parallel, or USB port, plus any printer on the network. CUPS will ensure a uniform presentation to you and your applications of all different types of printers.

Printers that only come with a Win9x driver could be problematic if they have no other support. Check with http://linuxprinting.org/ when in doubt.

In the past, your best choice would have been a printer with native PostScript support in the firmware, since nearly all UNIX or Linux software producing printable output, produces it in PostScript, the publishing industry's printer control language of choice. PostScript printers are usually a bit more expensive, but it is a device-independent, open programming language and you're always 100% sure that they will work. These days, however, the importance of this rule of thumb is dwindling.

8.3. Print problems

In this section, we will discuss what you can do as a user when something goes wrong. We won't discuss any problems that have to do with the daemon-part of the printing service, as that is a task for system administrators.

8.3.1. Wrong file

If you print the wrong file, the job may be canceled using the command **lprm *jobID***, where jobID is in the form *printername-printjobnumber* (get it from information displayed by **lpq** or **lpstat**). This will work when other jobs are waiting to be printed in this printer's queue. However, you have to be really quick if you are the only one using this printer, since jobs are usually spooled and send to the printer in only seconds. Once they arrive on the printer, it is too late to remove jobs using Linux tools.

What you can try in those cases, or in cases where the wrong print driver is configured and only rubbish comes out of the printer, is power off the printer. However, that might not be the best course of action, as you might cause paper jams and other irregularities.

8.3.2. My print hasn't come out

Use the **lpq** command and see if you can spot your job:

Note that this is not the CUPS web interface and only works for printers supporting this feature. Check the documentation of your printer.

If your job ID is not there and not on the printer, contact your system administrator. If your job ID is listed in the output, check that the printer is currently printing. If so, just wait, your job will get done in due time.

If the printer is not printing, check that it has paper, check the physical connections to both electricity and data network. If that's okay, the printer may need restarting. Ask your system admin for advice.

In the case of a network printer, try printing from another host. If the printer is reachable from your own host (see Chapter 10 for the **ping** utility), you may try to put the formatted file on it, like `file.ps` in case of a PostScript printer, using an FTP client. If that works, your print system is misconfigured. If it doesn't work, maybe the printer doesn't understand the format you are feeding it.

The GNU/Linux Printing site contains more tips and tricks.

8.4. Summary

The Linux print service comes with a set of printing tools based on the standard UNIX LPD tools, whether it be the SystemV or BSD implementation. Below is a list of print-related commands.

Table 8-1. New commands in chapter 8: Printing

Command	Meaning
lpr or **lp**	Print file
lpq or **lpstat**	Query print queue
lprm or **cancel**	Remove print job
acroread	PDF viewer
groff	Formatting tool
gv	PostScript viewer
printconf	Configure printers
xdvi	DVI viewer
xpdf	PDF viewer
***2ps**	Convert file to PostScript

8.5. Exercises

Configuring and testing printers involves being in the possession of one, and having access to the *root* account. If so, you may try:

- Installing the printer using the GUI on your system.
- Printing a test page using the GUI.
- Printing a test page using the **lp** command.
- Print from within an application, for example Mozilla or OpenOffice, by choosing File->Print from the menu.
- Disconnect the printer from the network or the local machine/print-server. What happens when you try to print something?

Chapter 9. Fundamental Backup Techniques

Accidents will happen sooner or later. In this chapter, we'll discuss how to get data to a safe place using other hosts, floppy disks, CD-ROMs and tapes. We will also discuss the most popular compressing and archiving commands.

Upon completion of this chapter, you will know how to:

- Make, query and unpack file archives
- Handle floppy disks and make a boot disk for your system
- Write CD-ROMs
- Make incremental backups
- Create Java archives
- Find documentation to use other backup devices and programs
- Encrypt your data

9.1. Introduction

Although Linux is one of the safest operating systems in existence, and even if it is designed to keep on going, data can get lost. Data loss is most often the consequence of user errors, but occasionally a system fault, such as a power or disk failure, is the cause, so it's always a good idea to keep an extra copy of sensitive and/or important data.

9.1.1. Preparing your data

9.1.1.1. Archiving with tar

In most cases, we will first collect all the data to back up in a single archive file, which we will compress later on. The process of archiving involves concatenating all listed files and taking out unnecessary blanks. In Linux, this is commonly done with the **tar** command. **tar** was originally designed to archive data on tapes, but it can also make archives, known as *tarballs*.

tar has many options, the most important ones are cited below:

- `-v`: verbose
- `-t`: test, shows content of a tarball
- `-x`: extract archive
- `-c`: create archive
- `-f archivedevice`: use `archivedevice` as source/destination for the tarball, the device defaults to the first tape device (usually `/dev/st0` or something similar)
- `-j`: filter through **bzip2**, see Section 9.1.1.2

It is common to leave out the dash-prefix with **tar** options, as you can see from the examples below.

Use GNU tar for compatibility

The archives made with a proprietary **tar** version on one system, may be incompatible with **tar** on another proprietary system. This may cause much headaches, such as if the archive needs to be recovered

on a system that doesn't exist anymore. Use the GNU **tar** version on all systems to prevent your system admin from bursting into tears. Linux always uses GNU tar. When working on other UNIX machines, enter **tar `--help`** to find out which version you are using. Contact your system admin if you don't see the word GNU somewhere.

In the example below, an archive is created and unpacked.

```
gaby:~> ls images/
me+tux.jpg  nimf.jpg

gaby:~> tar cvf images-in-a-dir.tar images/
images/
images/nimf.jpg
images/me+tux.jpg

gaby:~> cd images

gaby:~/images> tar cvf images-without-a-dir.tar *.jpg
me+tux.jpg
nimf.jpg

gaby:~/images> cd

gaby:~> ls */*.tar
images/images-without-a-dir.tar

gaby:~> ls *.tar
images-in-a-dir.tar

gaby:~> tar xvf images-in-a-dir.tar
images/
images/nimf.jpg
images/me+tux.jpg

gaby:~> tar tvf images/images-without-dir.tar
-rw-r--r-- gaby/gaby  42888 1999-06-30 20:52:25 me+tux.jpg
-rw-r--r-- gaby/gaby   7578 2000-01-26 12:58:46 nimf.jpg

gaby:~> tar xvf images/images-without-a-dir.tar
me+tux.jpg
nimf.jpg

gaby:~> ls *.jpg
me+tux.jpg  nimf.jpg
```

This example also illustrates the difference between a tarred directory and a bunch of tarred files. It is advisable to only compress directories, so files don't get spread all over when unpacking the tarball (which may be on another system, where you may not know which files were already there and which are the ones from the archive).

When a tape drive is connected to your machine and configured by your system administrator, the file names ending in `.tar` are replaced with the tape device name, for example:

tar `cvf /dev/tape mail/`

The directory `mail` and all the files it contains are compressed into a file that is written on the tape immediately. A content listing is displayed because we used the verbose option.

9.1.1.2. Incremental backups with tar

The **tar** tool supports the creation of incremental backups, using the `-N` option. With this option, you can specify a date, and **tar** will check modification time of all specified files against this date. If files are changed more recent than date, they will be included in the backup. The example below uses the timestamp on a previous archive as the date value. First, the initial archive is created and the timestamp on the initial backup file is shown. Then a new file is created, upon which we take a new backup, containing only this new file:

```
jimmy:~> tar cvpf /var/tmp/javaproggies.tar java/*.java
java/btw.java
java/error.java
java/hello.java
java/income2.java
java/income.java
java/inputdevice.java
java/input.java
java/master.java
java/method1.java
java/mood.java
java/moodywaitress.java
java/test3.java
java/TestOne.java
java/TestTwo.java
java/Vehicle.java

jimmy:~> ls -l /var/tmp/javaproggies.tar
-rw-rw-r-- 1 jimmy   jimmy   10240 Jan 21 11:58 /var/tmp/javaproggies.tar

jimmy:~> touch java/newprog.java

jimmy:~> tar -N /var/tmp/javaproggies.tar \
-cvp /var/tmp/incremental1-javaproggies.tar java/*.java 2> /dev/null
java/newprog.java

jimmy:~> cd /var/tmp/

jimmy:~> tar xvf incremental1-javaproggies.tar
java/newprog.java
```

Standard errors are redirected to `/dev/null`. If you don't do this, **tar** will print a message for each unchanged file, telling you it won't be dumped.

This way of working has the disadvantage that it looks at timestamps on files. Say that you download an archive into the directory containing your backups, and the archive contains files that have been created two years ago. When checking the timestamps of those files against the timestamp on the initial archive, the new files will actually seem old to **tar**, and will not be included in an incremental backup made using the `-N` option.

A better choice would be the `-g` option, which will create a list of files to backup. When making incremental backups, files are checked against this list. This is how it works:

```
jimmy:~> tar cvpf work-20030121.tar -g snapshot-20030121 work/
work/
work/file1
work/file2
work/file3

jimmy:~> file snapshot-20030121
```

```
snapshot-20030121: ASCII text
```

The next day, user *jimmy* works on `file3` a bit more, and creates `file4`. At the end of the day, he makes a new backup:

```
jimmy:~> tar cvpf work-20030122.tar -g snapshot-20030121 work/
work/
work/file3
work/file4
```

These are some very simple examples, but you could also use this kind of command in a cronjob (see Section 4.4.4), which specifies for instance a snapshot file for the weekly backup and one for the daily backup. Snapshot files should be replaced when taking full backups, in that case.

More information can be found in the **tar** documentation.

The real stuff

As you could probably notice, **tar** is OK when we are talking about a simple directory, a set of files that belongs together. There are tools that are easier to manage, however, when you want to archive entire partitions or disks or larger projects. We just explain about **tar** here because it is a very popular tool for distributing archives. It will happen quite often that you need to install a software that comes in a so-called "compressed tarball". See Section 9.3 for an easier way to perform regular backups.

9.1.1.3. Compressing and unpacking with gzip or bzip2

Data, including tarballs, can be compressed using zip tools. The **gzip** command will add the suffix .gz to the file name and remove the original file.

```
jimmy:~> ls -la | grep tar
-rw-rw-r-- 1 jimmy  jimmy    61440 Jun  6 14:08 images-without-dir.tar

jimmy:~> gzip images-without-dir.tar

jimmy:~> ls -la images-without-dir.tar.gz
-rw-rw-r-- 1 jimmy  jimmy    50562 Jun  6 14:08 images-without-dir.tar.gz
```

Uncompress gzipped files with the `-d` option.

bzip2 works in a similar way, but uses an improved compression algorithm, thus creating smaller files. See the **bzip2** info pages for more.

Linux software packages are often distributed in a gzipped tarball. The sensible thing to do after unpacking that kind of archives is find the `README` and read it. It will generally contain guidelines to installing the package.

The GNU **tar** command is aware of gzipped files. Use the command

tar `zxvf` `file.tar.gz`

for unzipping and untarring `.tar.gz` or `.tgz` files. Use

tar `jxvf` `file.tar.bz2`

for unpacking **tar** archives that were compressed with **bzip2**.

9.1.1.4. Java archives

The GNU project provides us with the **jar** tool for creating Java archives. It is a Java application that combines multiple files into a single JAR archive file. While also being a general purpose archiving and compression tool, based on ZIP and the ZLIB compression format, **jar** was mainly designed to facilitate the packing of Java code, applets and/or applications in a single file. When combined in a single archive, the components of a Java application, can be downloaded much faster.

Unlike **tar**, **jar** compresses by default, independent from other tools - because it is basically the Java version of **zip**. In addition, it allows individual entries in an archive to be signed by the author, so that origins can be authenticated.

The syntax is almost identical as for the **tar** command, we refer to **info** ***jar*** for specific differences.

tar, jar and symbolic links

One noteworthy feature not really mentioned in the standard documentation is that **jar** will follow symbolic links. Data to which these links are pointing will be included in the archive. The default in **tar** is to only backup the symbolic link, but this behavior can be changed using the `-h` to **tar**.

9.1.1.5. Transporting your data

Saving copies of your data on another host is a simple but accurate way of making backups. See Chapter 10 for more information on **scp**, **ftp** and more.

In the next section we'll discuss local backup devices.

9.2. Moving your data to a backup device

9.2.1. Making a copy on a floppy disk

9.2.1.1. Formatting the floppy

On most Linux systems, users have access to the floppy disk device. The name of the device may vary depending on the size and number of floppy drives, contact your system admin if you are unsure. On some systems, there will likely be a link `/dev/floppy` pointing to the right device, probably `/dev/fd0` (the auto-detecting floppy device) or `/dev/fd0H1440` (set for 1,44MB floppies).

fdformat is the low-level floppy disk formatting tool. It has the device name of the floppy disk as an option. **fdformat** will display an error when the floppy is write-protected.

```
emma:~> fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
emma:~>
```

The **mformat** command (from the mtools package) is used to create DOS-compatible floppies which can then be accessed using the **mcopy**, **mdir** and other m-commands.

10.3.7. The Domain Name System

All these applications need DNS services to match IP addresses to host names and vice versa. A DNS server does not know all the IP addresses in the world, but networks with other DNS servers which it can query to find an unknown address. Most UNIX systems can run **named**, which is part of the BIND (Berkeley Internet Name Domain) package distributed by the Internet Software Consortium. It can run as a stand-alone caching *nameserver*, which is often done on Linux systems in order to speed up network access.

Your main client configuration file is `/etc/resolv.conf`, which determines the order in which Domain Name Servers are contacted:

```
search somewhere.org
nameserver 192.168.42.1
nameserver 193.74.208.137
```

More information can be found in the Info pages on **named**, in the `/usr/share/doc/bind[-<version>]` files and on the Bind project homepage. The DNS HOWTO covers the use of BIND as a DNS server.

10.3.8. DHCP

DHCP is the Dynamic Host Configuration Protocol, which is gradually replacing good old **bootp** in larger environments. It is used to control vital networking parameters such as IP addresses and name servers of hosts. DHCP is backward compatible with **bootp**. For configuring the server, you will need to read the HOWTO.

DHCP client machines will usually be configured using a GUI that configures the **dhcpcd**, the DHCP client daemon. Check your system documentation if you need to configure your machine as a DHCP client.

10.3.9. Authentication services

10.3.9.1. Traditional

Traditionally, users are authenticated locally, using the information stored in `/etc/passwd` and `/etc/shadow` on each system. But even when using a network service for authenticating, the local files will always be present to configure system accounts for administrative use, such as the root account, the daemon accounts and often accounts for additional programs and purposes.

These files are often the first candidates for being examined by hackers, so make sure the permissions and ownerships are strictly set as should be:

```
bob:~> ls -l /etc/passwd /etc/shadow
-rw-r--r--    1 root     root         1803 Mar 10 13:08 /etc/passwd
-r--------    1 root     root         1116 Mar 10 13:08 /etc/shadow
```

10.3.9.2. PAM

Linux can use PAM, the Pluggable Authentication Module, a flexible method of UNIX authentication. Advantages of PAM:

- A common authentication scheme that can be used with a wide variety of applications.

- PAM can be implemented with various applications without having to recompile the applications to specifically support PAM.
- Great flexibility and control over authentication for the administrator and application developer.
- Application developers do not need to develop their program to use a particular authentication scheme. Instead, they can focus purely on the details of their program.

The directory `/etc/pam.d` contains the PAM configuration files (used to be `/etc/pam.conf`). Each application or service has its own file. Each line in the file has four elements:

- *Module*:
  - `auth`: provides the actual authentication (perhaps asking for and checking a password) and sets credentials, such as group membership or Kerberos tickets.
  - `account`: checks to make sure that access is allowed for the user (the account has not expired, the user is allowed to log in at this time of day, and so on).
  - `password`: used to set passwords.
  - `session`: used after a user has been authenticated. This module performs additional tasks which are needed to allow access (for example, mounting the user's home directory or making their mailbox available).

  The order in which modules are stacked, so that multiple modules can be used, is very important.
- *Control Flags*: tell PAM which actions to take upon failure or success. Values can be `required`, `requisite`, `sufficient` or `optional`.
- *Module Path*: path to the pluggable module to be used, usually in `/lib/security`.
- *Arguments*: information for the modules

Shadow password files are automatically detected by PAM.

More information can be found in the **pam** man pages or at the Linux-PAM project homepage.

10.3.9.3. LDAP

The Lightweight Directory Access Protocol is a client-server system for accessing global or local directory services over a network. On Linux, the OpenLDAP implementation is used. It includes **slapd**, a stand-alone server; **slurpd**, a stand-alone LDAP replication server; libraries implementing the LDAP protocol and a series of utilities, tools and sample clients.

The main benefit of using LDAP is the consolidation of certain types of information within your organization. For example, all of the different lists of users within your organization can be merged into one LDAP directory. This directory can be queried by any LDAP-enabled applications that need this information. It can also be accessed by users who need directory information.

Other LDAP or X.500 Lite benefits include its ease of implementation (compared to X.500) and its well-defined Application Programming Interface (API), which means that the number of LDAP-enabled applications and LDAP gateways should increase in the future.

On the negative side, if you want to use LDAP, you will need LDAP-enabled applications or the ability to use LDAP gateways. While LDAP usage should only increase, currently there are not very many LDAP-enabled applications available for Linux. Also, while LDAP does support some access control, it does not possess as many security features as X.500.

Since LDAP is an open and configurable protocol, it can be used to store almost any type of information relating to a particular organizational structure. Common examples are mail address lookups, central authentication in combination with PAM, telephone directories and machine configuration databases.

See your system specific information and the man pages for related commands such as **ldapmodify** and **ldapsearch** for details. More information can be found in the LDAP Linux HOWTO, which discusses installation, configuration, running and maintenance of an LDAP server on Linux. The author of this Introduction to Linux document also wrote an LDAP Operations HOWTO, describing the basics everyone should know about when dealing with LDAP management, operations and integration of services.

10.4. Remote execution of applications

10.4.1. Introduction

There are a couple of different ways to execute commands or run programs on a remote machine and have the output, be it text or graphics, sent to your workstation. The connections can be secure or insecure. While it is of course advised to use secure connections instead of transporting your password over the network unencrypted, we will discuss some practical applications of the older (unsafe) mechanisms, as they are still useful in a modern networked environment, such as for troubleshooting or running exotic programs.

10.4.2. Rsh, rlogin and telnet

The **rlogin** and **rsh** commands for remote login and remote execution of commands are inherited from UNIX. While seldom used because they are blatantly insecure, they still come with almost every Linux distribution for backward compatibility with UNIX programs.

Telnet, on the other hand, is still commonly used, often by system and network administrators. Telnet is one of the most powerful tools for remote access to files and remote administration, allowing connections from anywhere on the Internet. Combined with an X server, remote graphical applications can be displayed locally. There is no difference between working on the local machine and using the remote machine.

Because the entire connection is unencrypted, allowing **telnet** connections involves taking high security risks. For normal remote execution of programs, Secure SHell or **ssh** is advised. We will discuss the secure method later in this section.

However, **telnet** is still used in many cases. Below are some examples in which a mail server and a web server are tested for replies:

Checking that a mail server works:

```
[jimmy@blob ~] telnet mailserver 25
Trying 192.168.42.1...
Connected to mailserver.
Escape character is '^]'.
220 m1.some.net ESMTP Sendmail 8.11.6/8.11.6; 200302281626
ehlo some.net
250-m1.some.net Hello blob.some.net [10.0.0.1], pleased to meet you
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-SIZE
250-DSN
```

```
250-ONEX
250-ETRN
250-XUSR
250 HELP
mail from: jimmy@some.net
250 2.1.0 jimmy@some.net... Sender ok
rcpt to: davy@some.net
250 2.1.5 davy@some.net... Recipient ok
data
354 Enter mail, end with "." on a line by itself
test
.
250 2.0.0 g2MA1R619237 Message accepted for delivery
quit
221 2.0.0 m1.some.net closing connection
Connection closed by foreign host.
```

Checking that a web server answers to basic requests:

```
[jimmy@blob ~] telnet www.some.net 80
Trying 64.39.151.23...
Connected to www.some.net.
Escape character is '^]'.
HEAD / ;HTTP/1.1

HTTP/1.1 200 OK
Date: Fri, 22 Mar 2002 10:05:14 GMT
Server: Apache/1.3.22 (UNIX) (Red-Hat/Linux)
 mod_ssl/2.8.5 OpenSSL/0.9.6
 DAV/1.0.2 PHP/4.0.6 mod_perl/1.24_01
Last-Modified: Fri, 04 Jan 2002 08:21:00 GMT
ETag: "70061-68-3c3565ec"
Accept-Ranges: bytes
Content-Length: 104
Connection: close
Content-Type: text/html

Connection closed by foreign host.

[jimmy@blob ~]
```

This is perfectly safe, because you never have to give a username and/or password for getting the data you want, so nobody can snoop that important information off the cable.

10.4.3. The X Window System

10.4.3.1. X features

As we already explained in Chapter 7 (see Section 7.3.3), the X Window system comes with an X server which serves graphics to clients that need a display.

It is important to realize the distinction between the X server and the X client application(s). The X server controls the display directly and is responsible for all input and output via keyboard, mouse and display. The X client, on the other hand, does not access the input and output devices directly. It communicates with the X server which handles input and output. It is the X client which does the real work, like computing values, running applications and so forth. The X server only opens windows to handle input and output for the specified client.

In normal operation (graphical mode), every Linux workstation is an X server to itself, even if it only runs client applications. All the applications you are running (for example, Gimp, a terminal window, your browser, your office application, your CD playing tool, and so on) are clients to your X server. Server and client are running on the same machine in this case.

This client/server nature of the X system makes it an ideal environment for remote execution of applications and programs. Because the process is actually being executed on the remote machine, very little CPU power is needed on the local host. Such machines, purely acting as servers for X, are called X terminals and were once very popular. More information may be found in the Remote X applications mini-HOWTO.

10.4.3.2. Telnet and X

If you would want to use **telnet** to display graphical applications running on a remote machine, you first need to give the remote machine access to your display (to your X server!) using the **xhost** command, by typing a command similar to the one below in a terminal window on your local machine:

```
davy:~> xhost +remote.machine.com
```

After that, connect to the remote host and tell it to display graphics on the local machine by setting the environment variable `DISPLAY`:

```
[davy@remote ~] export DISPLAY="local.host.com:0.0"
```

After completing this step, any application started in this terminal window will be displayed on your local desktop, using remote resources for computing, but your local graphical resources (your X server) for displaying the application.

This procedure assumes that you have some sort of X server (XFree86, X.org, Exceed, Cygwin) already set up on the machine where you want to display images. The architecture and operating system of the client machine are not important as long as they allow you to run an X server on it.

Mind that displaying a terminal window from the remote machine is also considered to be a display of an image.

10.4.4. The SSH suite

10.4.4.1. Introduction

Most UNIX and Linux systems now run Secure SHell in order to leave out the security risks that came with **telnet**. Most Linux systems will run a version of OpenSSH, an Open Source implementation of the SSH protocol, providing secure encrypted communications between untrusted hosts over an untrusted network. In the standard setup X connections are automatically forwarded, but arbitrary TCP/IP ports may also be forwarded using a secure channel.

The **ssh** client connects and logs into the specified host name. The user must provide his identity to the remote machine as specified in the `sshd_config` file, which can usually be found in `/etc/ssh`. The configuration file is rather self-explanatory and by defaults enables most common features. Should you need help, you can find it in the **sshd** man pages.

When the user's identity has been accepted by the server, the server either executes the given command, or logs into the machine and gives the user a normal shell on the remote machine. All communication with the remote command or shell will be automatically encrypted.

The session terminates when the command or shell on the remote machine exits and all X11 and TCP/IP connections have been closed.

When connecting to a host for the first time, using any of the programs that are included in the SSH collection, you need to establish the authenticity of that host and acknowledge that you want to connect:

```
lenny ~> ssh blob
The authenticity of host 'blob (10.0.0.1)' can't be established.
RSA fingerprint is 18:30:50:46:ac:98:3c:93:1a:56:35:09:8d:97:e3:1d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'blob,192.168.30.2' (RSA) to the list of
known hosts.
Last login: Sat Dec 28 13:29:19 2002 from octarine
This space for rent.

lenny is in ~
```

It is important that you type "yes", in three characters, not just "y". This edits your `~/.ssh/known_hosts` file, see Section 10.4.4.3.

If you just want to check something on a remote machine and then get your prompt back on the local host, you can give the commands that you want to execute remotely as arguments to **ssh**:

```
lenny ~> ssh blob who
jenny@blob's password:
root     tty2         Jul 24 07:19
lena     tty3         Jul 23 22:24
lena     0:           Jul 25 22:03

lenny ~> uname -n
magrat.example.com
```

10.4.4.2. X11 and TCP forwarding

If the `X11Forwarding` entry is set to *yes* on the target machine and the user is using X applications, the `DISPLAY` environment variable is set, the connection to the X11 display is automatically forwarded to the remote side in such a way that any X11 programs started from the shell will go through the encrypted channel, and the connection to the real X server will be made from the local machine. The user should not manually set `DISPLAY`. Forwarding of X11 connections can be configured on the command line or in the **sshd** configuration file.

The value for `DISPLAY` set by **ssh** will point to the server machine, but with a display number greater than zero. This is normal, and happens because **ssh** creates a *proxy* X server on the server machine (that runs the X client application) for forwarding the connections over the encrypted channel.

This is all done automatically, so when you type in the name of a graphical application, it is displayed on your local machine and not on the remote host. We use **xclock** in the example, since it is a small program which is generally installed and ideal for testing:

Figure 10-3. SSH X11 forwarding

10.4.4.4. Secure remote copying

The SSH suite provides **scp** as a secure alternative to the **rcp** command that used to be popular when only **rsh** existed. **scp** uses **ssh** for data transfer, uses the same authentication and provides the same security as **ssh**. Unlike **rcp**, **scp** will ask for passwords or passphrases if they are needed for authentication:

```
lenny /var/tmp> scp Schedule.sdc.gz blob:/var/tmp/
lenny@blob's password:
Schedule.sdc.gz  100% |*****************************| 100 KB 00:00

lenny /var/tmp>
```

Any file name may contain a host and user specification to indicate that the file is to be copied to/from that host. Copies between two remote hosts are permitted. See the Info pages for more information.

If you would rather use an FTP-like interface, use **sftp**:

```
lenny /var/tmp> sftp blob
Connecting to blob...
lenny@blob's password:

sftp> cd /var/tmp

sftp> get Sch*
Fetching /var/tmp/Schedule.sdc.gz to Schedule.sdc.gz

sftp> bye

lenny /var/tmp>
```

Secure copy or FTP GUIs

Don't feel comfortable with the command line yet? Try Konqueror's capabilities for secure remote copy, or install Putty.

10.4.4.5. Authentication keys

The **ssh-keygen** command generates, manages and converts authentication keys for **ssh**. It can create RSA keys for use by SSH protocol version 1 and RSA or DSA keys for use by SSH protocol version 2.

Normally each user wishing to use SSH with RSA or DSA authentication runs this once to create the authentication key in `$HOME/.ssh/identity`, `id_dsa` or `id_rsa`. Additionally, the system administrator may use this to generate host keys for the system.

Normally this program generates the key and asks for a file in which to store the private key. The public key is stored in a file with the same name but *.pub* appended. The program also asks for a passphrase. The passphrase may be empty to indicate no passphrase (host keys must have an empty passphrase), or it may be a string of arbitrary length.

There is no way to recover a lost passphrase. If the passphrase is lost or forgotten, a new key must be generated and copied to the corresponding public keys.

We will study SSH keys in the exercises. All information can be found in the man or Info pages.

10.4.5. VNC

VNC or Virtual Network Computing is in fact a remote display system which allows viewing a desktop environment not only on the local machine on which it is running, but from anywhere on the Internet and from a wide variety of machines and architectures, including MS Windows and several UNIX distributions. You could, for example, run MS Word on a Windows NT machine and display the output on your Linux desktop. VNC provides servers as well as clients, so the opposite also works and it may thus be used to display Linux programs on Windows clients. VNC is probably the easiest way to have X connections on a PC. The following features make VNC different from a normal X server or commercial implementations:

- No state is stored at the viewer side: you can leave your desk and resume from another machine, continuing where you left. When you are running a PC X server, and the PC crashes or is restarted, all remote applications that you were running will die. With VNC, they keep on running.
- It is small and simple, no installation needed, can be run from a floppy if needed.
- Platform independent with the Java client, runs on virtually everything that supports X.
- Sharable: one desktop may be displayed on multiple viewers.
- Free.

More information can be found in the VNC client man pages (**man *vncviewer***) or on the VNC website.

10.4.6. The rdesktop protocol

In order to ease management of MS Windows hosts, recent Linux distributions support the Remote Desktop Protocol (RDP), which is implemented in the **rdesktop** client. The protocol is used in a number of Microsoft products, including Windows NT Terminal Server, Windows 2000 Server, Windows XP and Windows 2003 Server.

Surprise your friends (or management) with the fullscreen mode, multiple types of keyboard layouts and single application mode, just like the real thing. The **man *rdesktop*** manual provides more information. The project's homepage is at http://www.rdesktop.org/.

10.4.7. Cygwin

Cygwin provides substantial UNIX functionality on MS Windows systems. Apart from providing UNIX command line tools and graphical applications, it can also be used to display a Linux desktop on an MS Windows machine, using remote X. From a Cygwin Bash shell, type the command

/usr/X11R6/bin/XWin.exe -query *your_linux_machine_name_or_IP*

The connection is by default denied. You need to change the X Display Manager (XDM) configuration and possibly the X Font Server (XFS) configuration to enable this type of connection, where you get a login screen on the remote machine. Depending on your desktop manager (Gnome, KDE, other), you might have to change some configurations there, too.

If you do not need to display the entire desktop, you can use SSH in Cygwin, just like explained in Section 10.4.4. without all the fuss of editing configuration files.

connection.

The Shoreline Firewall or Shorewall for short is a front-end for the standard firewall functionality in Linux.

More information can be found at the Netfilter/iptables project page.

10.5.4.3. TCP wrappers

TCP wrapping provides much the same results as the packet filters, but works differently. The wrapper actually accepts the connection attempt, then examines configuration files and decides whether to accept or reject the connection request. It controls connections at the application level rather than at the network level.

TCP wrappers are typically used with **xinetd** to provide host name and IP-address-based access control. In addition, these tools include logging and utilization management capabilities that are easy to configure.

The advantages of TCP wrappers are that the connecting client is unaware that wrappers are used, and that they operate separately from the applications they protect.

The host based access is controlled in the `hosts.allow` and `hosts.deny` files. More information can be found in the TCP wrapper documentation files in `/usr/share/doc/tcp_wrappers[-<version>/]` or `/usr/share/doc/tcp` and in the man pages for the host based access control files, which contain examples.

10.5.4.4. Proxies

Proxies can perform various duties, not all of which have much to do with security. But the fact that they are an intermediary make proxies a good place to enforce access control policies, limit direct connections through a firewall, and control how the network behind the proxy looks to the Internet.

Usually in combination with a packet filter, but sometimes all by themselves, proxies provide an extra level of control. More information can be found in the Firewall HOWTO or on the Squid website.

10.5.4.5. Access to individual applications

Some servers may have their own access control features. Common examples include Samba, X Window, Bind, Apache and CUPS. For every service you want to offer check which configuration files apply.

10.5.4.6. Log files

If anything, the UNIX way of logging all kinds of activities into all kinds of files confirms that "it is doing something." Of course, log files should be checked regularly, manually or automatically. Firewalls and other means of access control tend to create huge amounts of log files, so the trick is to try and only log abnormal activities.

10.5.5. Intrusion detection

Intrusion Detection Systems are designed to catch what might have gotten past the firewall. They can either be designed to catch an active break-in attempt in progress, or to detect a successful break-in after the fact. In the latter case, it is too late to prevent any damage, but at least we have early awareness of a problem. There are two basic types of IDS: those protecting networks, and those protecting individual hosts.

For host based IDS, this is done with utilities that monitor the file system for changes. System files that have changed in some way, but should not change, are a dead give-away that something is amiss. Anyone who gets in and gets root access will presumably make changes to the system somewhere. This is usually the very first thing done, either so he can get back in through a backdoor, or to launch an attack against someone else, in which case, he has to change or add files to the system. Some systems come with the **tripwire** monitoring system, which is documented at the Tripwire Open Source Project website.

Network intrusion detection is handled by a system that sees all the traffic that passes the firewall (not by portscanners, which advertise usable ports). Snort is an Open Source example of such a program. Whitehats.com features an open Intrusion detection database, arachNIDS.

10.5.6. More tips

Some general things you should keep in mind:

- Do not allow root logins. UNIX developers came up with the **su** over two decades ago for extra security.
- Direct root access is always dangerous and susceptible to human errors, be it by allowing root login or by using the **su -** command. Rather than using **su**, it is even better to use **sudo** to only execute the command that you need extra permissions for, and to return afterwards to your own environment.
- Take passwords seriously. Use shadow passwords. Change your passwords regularly.
- Try to always use SSH or SSL. Avoid **telnet**, FTP and E-mail clients and other client programs which send unencrypted passwords over the network. Security is not only about securing your computer, it is also about securing your passwords.
- Limit resources using **quota** and/or **ulimit**.
- The mail for root should be delivered to, or at least read by, an actual person.
- The SANS institute has more tips and tricks, sorted per distribution, with mailing list service.
- Check the origin of new software, get it from a trusted place/site. Verify new packages before installing.
- When using a non-permanent Internet connection, shut it down as soon as you don't need it anymore.
- Run private services on odd ports instead of the ones expected by possible hackers.
- Know your system. After a while, you can almost feel when something is happening.

10.5.7. Have I been hacked?

How can you tell? This is a checklist of suspicious events:

- Mysterious open ports, strange processes.
- System utilities (common commands) behaving strange.
- Login problems.
- Unexplained bandwidth usage.
- Damaged or missing log files, syslog daemon behaving strange.
- Interfaces in unusual modes.
- Unexpectedly modified configuration files.
- Strange entries in shell history files.
- Unidentified temporary files.

10.5.8. Recovering from intrusion

In short, stay calm. Then take the following actions in this order:

- Disconnect the machine from the network.
- Try to find out as much as you can about how your security was breached.
- Backup important non-system data. If possible, check these data against existing backups, made before the system was compromised, to ensure data integrity.
- Re-install the system.
- Use new passwords.
- Restore from system and data backups.
- Apply all available updates.
- Re-examine the system: block off unnecessary services, check firewall rules and other access policies.
- Reconnect.

10.6. Summary

Linux and networking go hand in hand. The Linux kernel has support for all common and most uncommon network protocols. The standard UNIX networking tools are provided in each distribution. Next to those, most distributions offer tools for easy network installation and management.

Linux is well known as a stable platform for running various Internet services, the amount of Internet software is endless. Like UNIX, Linux can be just as well used and administered from a remote location, using one of several solutions for remote execution of programs.

We briefly touched the subject of security. Linux is an ideal firewall system, light and cheap, but can be used in several other network functions such as routers and proxy servers.

Increasing network security is mainly done by applying frequent updates and common sense.

Here is an overview of network related commands:

Table 10-2. New commands in chapter 10: Networking

Command	Meaning
ftp	Transfer files to another host (insecure).
host	Get information about networked hosts.
ifconfig	Display IP address information.
ip	Display IP address information.
netstat	Display routing information and network statistics.
ping	Send answer requests to other hosts.
rdesktop	Display and MS Windows desktop on your Linux system.
route	Show routing information.
scp	Secure copy files to and from other hosts.
sftp	Secure FTP files to and from other hosts.
ssh	Make an encrypted connection to another host.

ssh-keygen	Generate authentication keys for Secure SHell.
telnet	Make an insecure connection to another hosts.
tracepath/traceroute	Print the route that packets follow to another host.
whois	Get information abotu a domain name.
xclock	X Window clock application, handy for testing remote display.
xhost	X Window access control tool.

10.7. Exercises

10.7.1. General networking

- Display network information for your workstation: IP address, routes, name servers.
- Suppose no DNS is available. What would you do to reach your neighbour's machine without typing the IP address all the time?
- How would you permanently store proxy information for a text mode browser such as **links**?
- Which name servers handle the redhat.com domain?
- Send an E-mail to your local account. Try two different ways to send and read it. How can you check that it really arrived?
- Does your machine accept anonymous FTP connections? How do you use the **ncftp** program to authenticate with your user name and password?
- Does your machine run a web server? If not, make it do so. Check the log files!

10.7.2. Remote connections

- From your local workstation, display a graphical application, such as **xclock** on your neighbour's screen. The necessary accounts will have to be set up. Use a secure connection!
- Set up SSH keys so you can connect to your neighbour's machine without having to enter a password.
- Make a backup copy of your home directory in `/var/tmp` on your neighbour's "backup server," using **scp**. Archive and compress before starting the data transfer! Connect to the remote host using **ssh**, unpack the backup, and put one file back on the original machine using **sftp**.

10.7.3. Security

- Make a list of open (listening) ports on your machine.
- Supposing you want to run a web server. Which services would you deactivate? How would you do that?
- Install available updates.
- How can you see who connected to your system?
- Make a repetitive job that reminds you to change your password every month, and preferably the *root* password as well.

Some distributions don't allow you to play MP3's without modifying your configuration, this is due to license restrictions on the MP3 tools. You might need to install extra software to be able to play your music.

In text mode, you can use the **mplayer** command:

```
[tille@octarine ~]$ mplayer /opt/mp3/oriental/*.mp3
MPlayer 1.0pre7-RPM-3.4.2 (C) 2000-2005 MPlayer Team
CPU: Advanced Micro Devices Duron Spitfire (Family: 6, Stepping: 1)
Detected cache-line size is 64 bytes
CPUflags:  MMX: 1 MMX2: 1 3DNow: 1 3DNow2: 1 SSE: 0 SSE2: 0
Playing /opt/oldopt/mp3/oriental/Mazika_Diana-Krozon_Super-Star_Ensani-Ma-
Bansak.mp3.
Cache fill:  1.17% (98304 bytes)    Audio file detected.
Clip info:
Title: Ensani-Ma-Bansak.mp3
Artist: Diana-Krozon
Album: Super-Star
Year:
Comment:
Genre: Unknown
==========================================================================
Opening audio decoder: [mp3lib] MPEG layer-2, layer-3
mpg123: Can't rewind stream by 450 bits!
AUDIO: 44100 Hz, 2 ch, s16le, 160.0 kbit/11.34% (ratio: 20000->176400)
Selected audio codec: [mp3] afm:mp3lib (mp3lib MPEG layer-2, layer-3)
==========================================================================
Checking audio filter chain for 44100Hz/2ch/s16le -> 44100Hz/2ch/s16le...
AF_pre: 44100Hz/2ch/s16le
AO: [oss] 44100Hz 2ch s16le (2 bps)
Building audio filter chain for 44100Hz/2ch/s16le -> 44100Hz/2ch/s16le...
Video: no video
Starting playback...
A: 227.8 (03:23:.1) 1.8% 12%
```

11.2.2.2. Other formats

It would lead us too far to discuss all possible audio formats and ways to play them. An (incomplete) overview of other common sound playing and manipulating software:

- Ogg Vorbis: Free audio format: see the GNU audio directory for tools - they might be included in your distribution as well. The format was developed because MP3 is patented.
- Real audio and video: **realplay** from RealNetworks.
- SoX or Sound eXchange: actually a sound converter, comes with th e **play** program. Plays `.wav`, `.ogg` and various other formats, including raw binary formats.
- Playmidi: a MIDI player, see the GNU directory.
- AlsaPlayer: from the Advanced Linux Sound Architecture project, see the AlsaPlayer web site.
- **mplayer**: plays just about anything, including mp3 files. More info on the MPlayerHQ website.
- **hxplay**: supports RealAudio and RealVideo, mp3, mp4 audio, Flash, wav and more, see HelixDNA (not all components of this software are completely free).
- **rhythmbox**: based on the GStreamer framework, can play everything supported in GStreamer, which claims to be able to play everything, see the Rhythmbox and GStreamer sites.

Check your system documentation and man pages for particular tools and detailed explanations on how to use them.

I don't have these applications on my system!

A lot of the tools and applications discussed in the above sections are optional software. It is possible that such applications are not installed on your system by default, but that you can find them in your distribution as additional packages. It might also very well be that the application that you are looking for is not in your distribution at all. In that case, you need to download it from the application's web site.

11.2.2.3. Volume control

aumix and **alsamixer** are two common text tools for adjusting audio controls. Use the arrow keys to toggle settings. The **alsamixer** has a graphical interface when started from the Gnome menu or as **gnome-alsamixer** from the command line. The **kmix** tool does the same in KDE.

Regardless of how you choose to listen to music or other sounds, remember that there may be other people who may not be interested in hearing you or your computer. Try to be courteous, especially in office environments. Use a quality head-set, rather than the ones with the small ear pieces. This is better for your ears and causes less distraction for your colleagues.

11.2.3. Recording

Various tools are again available that allow you to record voice and music. For recording voice you can use **arecord** on the command line:

```
alexey@russia:~> arecord /var/tmp/myvoice.wav
Recording WAVE '/var/tmp/myvoice.wav' : Unsigned 8 bit, Rate 8000 Hz, Mono
Aborted by signal Interrups...
```

"Interrupt" means that the application has caught a **Ctrl**+**C**. Play the sample using the simple **play** command.

This is a good test that you can execute prior to testing applications that need voice input, like Voice over IP (VoIP). Keep in mind that the microphone input should be activated. If you don't hear your own voice, check your sound settings. It often happens that the microphone is muted or on verry low volume. This can be easily adjusted using **alsamixer** or your distribution-specific graphical interface to the sound system.

In KDE you can start the **krec** utility, Gnome provides the **gnome-sound-recorder**.

11.3. Video playing, streams and television watching

Various players are available:

- **xine**: a free video player
- **ogle**: DVD player.
- **okle**: KDE version of **ogle**
- **mplayer**: Movie Player for Linux
- **totem**: plays both audio and video files, audio CDs, VCD and DVD.
- **realplay**: from RealNetworks.
- **hxplay**: a Real alternative, see HelixDNA.
- **kaffeine**: media player for KDE3.

Most likely, you will find one of these in your graphical menus.

Keep in mind that all codecs necessary for viewing different types of video might not be on your system by default. You can get a long way downloading W32codecs and libdvdcss.

The LDP released a document that is very appropriate for this section. It is entitled DVD Playback HOWTO and describes the different tools available for viewing movies on a system that has a DVD drive. It is a fine addition to the DVD HOWTO that explains installation of the drive.

For watching TV there is choice of the following tools, among many others for watching and capturing TV, video and other streams:

- **tvtime**: great program with station management, interaction with teletext, film mode and much more.
- **zapping**: Gnome-specific TV viewer.
- **xawtv**: X11 TV viewer.

11.4. Internet Telephony

11.4.1. What is it?

Internet telephony, or more common, Voice over IP (VoIP) or digital telephony allows parties to exchange voice data flows over the network. The big difference is that the data flows over a general purpose network, the Internet, contrary to conventional telephony, that uses a dedicated network of voice transmission lines. The two networks can be connected, however, under special circumstances, but for now this is certainly not a standard. In other words: it is very likely that you will not be able to call people who are using a conventional telephone. If it is possible at all, it is likely that you will need to pay for a subscription.

While there are currently various applications available for free download, both free and proprietary, there are some major drawbacks to telephony over the Internet. Most noticably, the system is unreliable, it can be slow or there can be a lot of noise on the connection, and it can thus certainly not be used to replace conventional telephony - think about emergency calls. While some providers take their precautions, there is no guarantee that you can reach the party that you want to call.

Most applications currently do not use encryption, so be aware that it is potentially easy for someone to eavesdrop on your conversations. If security is a concern for you, read the documentation that comes with your VoIP client. Additionally, if you are using a firewall, it should be configured to allow incoming connections from anywhere, so using VoIP also includes taking risks on the level of site security.

11.4.2. What do you need?

11.4.2.1. Server Side

First of all, you need a provider offering the service. This service might integrate traditional telephony and it might or might not be free. Among others are SIPphone, Vonage, Lingo, AOL TotalTalk and many locally accessible providers offering the so-called "full phone service". Internet phone service only is offered by Skype, SIP Broker, Google and many others.

If you want to set up a server of your own, you might want to look into Asterisk.

11.4.2.2. Client Side

On the client side, the applications that you can use depend on your network configuration. If you have a direct Internet connection, there won't be any problems, provided that you know on what server you can connect, and usually that you also have a username and password to authenticate to the service.

If you are behind a firewall that does Network Address Translation (NAT), however, some services might not work, as they will only see the IP address of the firewall and not the address of your computer, which might well be unroutable over the Internet, for instance when you are in a company network and your IP address starts with 10., 192.168. or another non-routable subnet prefix. This depends on the protocol that is used by the application.

Also, available bandwidth might be a blocking factor: some applications are optimized for low bandwidth consumption, while others might require high bandwidth connections. This depends on the codec that is used by the application.

Among the most common applications are the Skype client, which has an interface that reminds of instant messaging, and X-Lite, the free version of the XTen softphone, which looks like a mobile telephone. However, while these programs are available for free download and very popular, they are not free as in free speech: they use proprietary protocols and/or are only available in binary packages, not in source format.

Free *and open* VoIP clients are for instance Gizmo, Linphone, GnomeMeeting and KPhone.

Client hardware

While your computer, especially if it is a laptop PC, might have a built-in microphone, the result will be far better if you connect a headset. If you have the choice, opt for a USB headset, as it functions independently from existing audio hardware. Use **alsamixer** to configure input and output sound levels to your taste.

VoIP applications are definitely a booming market. Volunteers try to document the current status at http://www.voip-info.org/.

11.5. Summary

The GNU/Linux platform is fully multi-media enabled. A wide variety of devices like sound cards, tv-cards, headsets, microphones, CD and DVD players is supported. The list of applications is sheer endless, that is why we needed to shortenthe list of new commands below and limit ourselves to general audio commands.

Table 11-1. New commands in chapter 11: Audio

Command	Meaning
alsaconf	Configure the ALSA sound system.
alsamixer	Tune output levels of ALSA driver.
arecord	Record a sound sample.
aumix	Audio mixer tool.
cdp	Play an audio CD.
cdparanoia	Rip an audio CD.
cdplay	Play an audio CD.

gnome-alsamixer	Gnome ALSA front-end.
gnome-cd	Gnome front-end for playing audio CDs.
gnome-sound-recorder	Gnome front-end for recording sound samples.
kaudiocreator	KDE front-end for creating audio CDs.
kmix	KDE front-end for sound settings.
krec	KDE front-end for recording sound samples.
mplayer	Multi-media player.
play	Command line tool for playing sound samples.

11.6. Exercises

1. From the Gnome or KDE menu, open your sound configuration panel. Make sure audio boxes or headset are connected to your system and find an output level that is comfortable for you. Make sure, when your system is ALSA-compatible, that you use the appropriate panel.
2. If you have a microphone, try recording a sample of your own voice. Make sure the input volume is not too high, as this will result in high-pitched tones when you communicate with others, or in transfering background noise to the other party. On the command line, you might even try to use **arecord** and **aplay** for recording and playing sound.
3. Locate sound files on your system and try to play them.
4. Insert an audio CD and try to play it.
5. Find a chat partner and configure a VoIP program. (You might need to install one first.)
6. Can you listen to Internet radio?
7. If you have a DVD player and a movie on a DVD disk, try to play it.

Appendix A. Where to go from here?

This document gives an overview of useful books and sites.

A.1. Useful Books

A.1.1. General Linux

- "Linux in a Nutshell" by Ellen Siever, Jessica P. Hackman, Stephen Spainhour, Stephen Figgins, O'Reilly UK, ISBN 0596000251
- "Running Linux" by Matt Welsh, Matthias Kalle Dalheimer, Lar Kaufman, O'Reilly UK, ISBN 156592469X
- "Linux Unleashed" by Tim Parker, Bill Ball, David Pitts, Sams, ISBN 0672316889
- "When You Can't Find Your System Administrator" by Linda Mui, O'Reilly UK, ISBN 1565921046
- When you actually buy a distribution, it will contain a very decent user manual.

A.1.2. Editors

- "Learning the Vi Editor" by Linda Lamb and Arnold Robbins, O'Reilly UK, ISBN 1565924266
- "GNU Emacs Manual" by Richard M.Stallman, iUniverse.Com Inc., ISBN 0595100333
- "Learning GNU Emacs" by Debra Cameron, Bill Rosenblatt and Eric Raymond, O'Reilly UK, ISBN 1565921526
- "Perl Cookbook" by Tom Christiansen and Nathan Torkington, O'Reilly UK, ISBN 1565922433

A.1.3. Shells

- "Unix Shell Programming" by Stephen G.Kochan and Patrick H.Wood, Sams Publishing, ISBN 067248448X
- "Learning the Bash Shell" by Cameron Newham and Bill Rosenblatt, O'Reilly UK, ISBN 1565923472
- "The Complete Linux Shell Programming Training Course" by Ellie Quigley and Scott Hawkins, Prentice Hall PTR, ISBN 0130406767
- "Linux and Unix Shell Programming" by David Tansley, Addison Wesley Publishing Company, ISBN 0201674726
- "Unix C Shell Field Guide" by Gail and Paul Anderson, Prentice Hall, ISBN 013937468X

A.1.4. X Window

- "Gnome User's Guide" by the Gnome Community, iUniverse.Com Inc., ISBN 0595132251
- "KDE Bible" by Dave Nash, Hungry Minds Inc., ISBN 0764546929
- "The Concise Guide to XFree86 for Linux" by Aron HSiao, Que, ISBN 0789721821
- "The New XFree86" by Bill Ball, Prima Publishing, ISBN 0761531521
- "Beginning GTK+ and Gnome" by Peter Wright, Wrox Press, ISBN 1861003811
- "KDE 2.0 Development" by David Sweet and Matthias Ettrich, Sams Publishing, ISBN 0672318911
- "GTK+/Gnome Application Development" by Havoc Pennington, New Riders Publishing, ISBN 0735700788

A.1.5. Networking

- "TCP/IP Illustrated, Volume I: The Protocols" by W. Richard Stevens, Addison-Wesley Professional Computing Series, ISBN 0-201-63346-9
- "DNS and BIND" by Paul Albitz, Cricket Liu, Mike Loukides and Deborah Russell, O'Reilly & Associates, ISBN 0596001584
- "The Concise Guide to DNS and BIND" by Nicolai Langfeldt, Que, ISBN 0789722739
- "Implementing LDAP" by Mark Wilcox, Wrox Press, ISBN 1861002211
- "Understanding and deploying LDAP directory services" by Tim Howes and co., Sams, ISBN 0672323168
- "Sendmail" by Brian Costales and Eric Allman, O'Reilly UK, ISBN 1565922220
- "Removing the Spam : Email Processing and Filtering" by Geoff Mulligan, Addison Wesley Publishing Company, ISBN 0201379570
- "Managing IMAP" by Dianna & Kevin Mullet, O'Reilly UK, ISBN 059600012X

A.2. Useful sites

A.2.1. General information

- The Linux documentation project: all docs, manpages, HOWTOs, FAQs
- LinuxQuestions.org: forum, downloads, docs and much more
- Google for Linux: the specialized search engine
- Google Groups: an archive of all newsgroup postings, including the comp.os.linux hierarchy
- Slashdot: daily news
- http://www.oreilly.com: books on Linux System and Network administration, Perl, Java, ...
- POSIX: the standard
- Linux HQ: Maintains a complete database of source, patches and documentation for various versions of the Linux kernel.

A.2.2. Architecture Specific References

- AlphaLinux: Linux on Alpha architecture (e.g. Digital Workstation)
- Linux-MIPS: Linux on MIPS (e.g. SGI Indy)
- Linux on the Road: Specific guidelines for installing and running Linux on laptops, PDAs, mobile phones and so on. Configuration files for various models.
- MkLinux: Linux on Apple

A.2.3. Distributions

- The Fedora Project: RedHat-sponsored community effort OS.
- Mandriva
- Ubuntu
- Debian
- TurboLinux
- Slackware
- SuSE
- LinuxISO.org: CD images for all distributions.

Appendix B. DOS versus Linux commands

In this appendix, we matched DOS commands with their Linux equivalent.

As an extra means of orientation for new users with a Windows background, the table below lists MS-DOS commands with their Linux counterparts. Keep in mind that Linux commands usually have a number of options. Read the Info or man pages on the command to find out more.

Table B-1. Overview of DOS/Linux commands

DOS commands	Linux command
<command> /?	**man <command>** or **command `--help`**
cd	**cd**
chdir	**pwd**
cls	**clear**
copy	**cp**
date	**date**
del	**rm**
dir	**ls**
echo	**echo**
edit	**vim (or other editor)**
exit	**exit**
fc	**diff**
find	**grep**
format	**mke2fs or mformat**
mem	**free**
mkdir	**mkdir**
more	**more or even less**
move	**mv**
ren	**mv**
time	**date**

Appendix C. Shell Features

This document gives an overview of common shell features (the same in every shell flavour) and differing shell features (shell specific features).

C.1. Common features

The following features are standard in every shell. Note that the stop, suspend, jobs, bg and fg commands are only available on systems that support job control.

Table C-1. Common Shell Features

Command	Meaning
>	Redirect output
>>	Append to file
<	Redirect input
<<	"Here" document (redirect input)
|	Pipe output
&	Run process in background.
;	Separate commands on same line
*	Match any character(s) in filename
?	Match single character in filename
[ ]	Match any characters enclosed
( )	Execute in subshell
` `	Substitute output of enclosed command
" "	Partial quote (allows variable and command expansion)
' '	Full quote (no expansion)
\	Quote following character
$var	Use value for variable
$$	Process id
$0	Command name
$n	nth argument (n from 0 to 9)
$*	All arguments as a simple word
#	Begin comment
bg	Background execution
break	Break from loop statements
cd	Change directories
continue	Resume a program loop
echo	Display output
eval	Evaluate arguments
exec	Execute a new shell