

Mandrake Linux 8.2

Reference Manual

MandrakeSoft

March 2002

<http://www.mandrakelinux.com/>

Mandrake Linux 8.2 : Reference Manual
by MandrakeSoft

Copyright © 1999-2002 by **MandrakeSoft**

Table of Contents

Preface	i
1. Legal Notice	i
2. About Mandrake Linux	i
2.1. Contact Mandrake Community	i
2.2. Support Mandrake	ii
2.3. Purchasing Mandrake Products	ii
3. Authors And Translators	ii
4. Tools Used in The Making of This Manual	iii
5. Note From The Editor	iii
6. Conventions Used in This Book	iii
6.1. Typing Conventions	iii
6.2. General Conventions	iv
7. Introduction	v
I. Introduction to Linux	1
1. Basic UNIX System Concepts	1
1.1. Users and Groups	1
1.2. File Basics	2
1.3. Processes	4
1.4. Small Introduction to the Command Line	4
2. Introduction to The Command Line	9
2.1. File-Handling Utilities	9
2.2. Handling File Attributes	11
2.3. Shell Globbing Patterns	12
2.4. Redirections and Pipes	13
2.5. Command-Line Completion	14
2.6. Starting and Handling Background Processes: Job Control	15
2.7. A Final Word	16
3. Text Editing: <i>Emacs</i> and <i>Vi</i>	17
3.1. <i>Emacs</i>	17
3.2. <i>Vi</i> : the ancestor	19
3.3. A last word... ..	23
4. Command Line Utilities	25
4.1. <i>grep</i> : Locate Strings in Files	25
4.2. <i>find</i> : Find Files According to Certain Criteria	26
4.3. <i>crontab</i> : reporting or editing your <i>crontab</i> file	27
4.4. <i>at</i> : schedule a command, but only once	28
4.5. <i>tar</i> : Tape ARchiver	29
4.6. <i>bzip2</i> and <i>gzip</i> : Data Compression Programs	30
4.7. Many, many more... ..	31
5. Process control	33
5.1. More about processes	33
5.2. Information on processes: <i>ps</i> and <i>pstree</i>	33
5.3. Sending signals to processes: <i>kill</i> , <i>killall</i> and <i>top</i>	34
II. Linux in Depth	39
6. File Tree Organization	39
6.1. Shareable/Unshareable, Static/Variable Data	39
6.2. The root Directory: /	39
6.3. /usr: The Big One	40
6.4. /var: Modifiable Data During Use	40
6.5. /etc: Configuration Files	40
7. Filesystems and Mount Points	41
7.1. Principles	41
7.2. Partitioning a Hard Disk, Formatting a Partition	42
7.3. The mount And umount Commands	42
7.4. The /etc/fstab File	43
7.5. A Note About The Supermount Feature	44
8. The Linux Filesystem	45
8.1. Comparison of a Few Filesystems	45
8.2. Everything is a File	46

8.3. Links	48
8.4. "Anonymous" Pipes and Named Pipes	48
8.5. "Special" Files: Character Mode and Block Mode Files	50
8.6. Symbolic Links, Limitation of "Hard" Links	50
8.7. File Attributes	51
9. The /proc Filesystem	53
9.1. Information About Processes	53
9.2. Information on The Hardware	54
9.3. The /proc/sys Sub-Directory	56
10. The Start-Up Files: init sysv	57
10.1. In The Beginning Was init	57
10.2. Runlevels	57
III. Advanced Uses	61
11. Printing	61
11.1. Installing And Managing Printers	61
11.2. Printing Documents	66
12. msec – Mandrake Security Tools	73
12.1. Introducing msec	73
12.2. Setting Your Security Level	73
12.3. Security Levels Features	74
12.4. Customization	79
13. Building and installing free software	81
13.1. Introduction	81
13.2. Decompression	82
13.3. Configuration	84
13.4. Compilation	87
13.5. Installation	92
13.6. Support	92
13.7. Acknowledgments	93
14. Compiling And Installing New Kernels	95
14.1. Where to Find Kernel Sources	95
14.2. Unpacking Sources, Patching The Kernel (if Necessary)	95
14.3. Configuring The Kernel	96
14.4. Saving, Reusing Your Kernel Configuration Files	97
14.5. Compiling Kernel And Modules, Installing The Beast	98
14.6. Installing The New Kernel Manually	98
15. Troubleshooting	103
15.1. Introduction	103
15.2. A Boot Disk	103
15.3. Backup	105
15.4. Restore	111
15.5. My System Freezes at Boot Time	111
15.6. Bootloader Reinstall	112
15.7. Runlevels	113
15.8. Recovering Deleted Files	114
15.9. Recovering From a System Freeze	114
15.10. Killing Misbehaved Apps	115
15.11. Mandrake's Specific Troubleshooting Tools	116
15.12. Final Thoughts	116
A. The GNU General Public License	117
A.1. Preamble	117
A.2. Terms and conditions for copying, distribution and modification	117
B. GNU Free Documentation License	121
B.1. GNU Free Documentation License	121
0. PREAMBLE	121
1. APPLICABILITY AND DEFINITIONS	121
2. VERBATIM COPYING	122
3. COPYING IN QUANTITY	122
4. MODIFICATIONS	122
5. COMBINING DOCUMENTS	123

6. COLLECTIONS OF DOCUMENTS	124
7. AGGREGATION WITH INDEPENDENT WORKS.....	124
8. TRANSLATION.....	124
9. TERMINATION.....	124
10. FUTURE REVISIONS OF THIS LICENSE	124
B.2. How to use this License for your documents.....	124
Glossary	127
.....	

List of Tables

8-1. Filesystem Characteristics	46
---------------------------------------	----

List of Figures

1-1. Graphical-Mode Login Session	1
1-2. Console Mode Login Session	1
1-3. The Terminal Icon on the KDE Panel	4
3-1. <i>Emacs</i> , editing two files at once	17
3-2. <i>Emacs</i> , before copying the text block	18
3-3. <i>Emacs</i> , after having copied the text block	19
3-4. Starting position in <i>vim</i>	20
3-5. <i>vim</i> , before copying the text block	22
3-6. <i>vim</i> , after having copied the text block	22
5-1. Example of execution of <i>top</i>	34
7-1. A Not Yet Mounted Filesystem	41
7-2. Filesystem Is Now Mounted	41
11-1. The CUPS Welcome Page	61
11-2. The Empty CUPS Printers List	62
11-3. The CUPS Login Dialog	62
11-4. Adding a New Printer, Step 1	62
11-5. Adding a New Printer, Step 2	63
11-6. Adding a New Printer, Step 3	64
11-7. Adding a New Printer, Step 4	64
11-8. The Printer Status Page	65
11-9. The XPP Main Window	66
11-10. The XPP File Selection	67
11-12. The XPP Basic Options Dialog	68
11-13. The XPP Text Options Dialog	69
11-14. The XPP Advanced Options Dialog	70
15-1. Enter your root password	104
15-2. <i>drakfloppy</i> 's main window	104
15-3. Making a customized boot disk	104

Preface

1. Legal Notice

This manual (except the chapter listed below) is protected under **MandrakeSoft** intellectual property rights. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the invariant sections being *About Mandrake Linux*, page i, with the front-cover texts being listed below, and with no Back-Cover Texts. A copy of the license is included in the section *GNU Free Documentation License*, page 121.

Front-cover texts:

MandrakeSoft March 2002

<http://www.mandrakesoft.com/>

Copyright © 1999,2000,2001,2002 by MandrakeSoft S.A. and MandrakeSoft Inc.



The chapter quoted in the table below is subject to another copyright owner than the whole manual and a slightly different license:

	Original Copyright	License
<i>"Building and installing free software"</i> , page 81	Benjamin Drieu, APRIL (http://www.april.org/)	GNU Free Documentation License with no Invariant Sections no Front-Cover Texts nor Back-Cover Texts.

"Mandrake", "Mandrake Linux" and "MandrakeSoft" are registered trademarks of **MandrakeSoft S.A.**; Linux is a registered trademark of Linus Torvalds; *UNIX* is a registered trademark of The Open Group in the United States and other countries. All other trademarks and copyrights are the property of their respective owners.

2. About Mandrake Linux

Mandrake Linux is a *GNU/Linux* distribution supported by **MandrakeSoft S.A.** **MandrakeSoft** was born in the Internet in 1998 with the main goal to provide an easy-to-use and friendly *GNU/Linux* system. The two pillars of **MandrakeSoft** are open-source and collaborative work.

2.1. Contact Mandrake Community

Following are various Internet links pointing you to various **Mandrake Linux** related sources. If you wish to know more about the **MandrakeSoft** company, connect to its web site (<http://www.mandrakesoft.com/>). There is also the **Mandrake Linux** distribution (<http://www.mandrakelinux.com/>) web site and all its derivatives.

First of all, **MandrakeSoft** is proud to present its new open help platform. MandrakeExpert (<http://www.mandrakeexpert.com/>) isn't just another web site where people help others with their computer problems in exchange for up-front fees, payable regardless of the quality of the service received. It offers a new experience based on trust and the pleasure of rewarding others for their contributions.

In addition, MandrakeCampus (<http://mandrakecampus.com/>) provides the *GNU/Linux* community with open education and training courses on all open-software-related technologies and issues. It also gives teachers, tutors and learners a place where they can share knowledge.

There is a site for the "mandrakeholic" called Mandrake Forum (<http://www.mandrakeforum.com/>): a primary site for **Mandrake Linux** related tips, tricks, rumors, pre-announcements, semi-official news, and more. This is also the only interactive web site hosted by **MandrakeSoft**, so if you have something to tell us, or something you want to share with other users, search no longer: this is a place to do it!

In the philosophy of open source, **MandrakeSoft** is offering many means of support (<http://www.mandrakelinux.com/en/ffreesup.php3>) for the **Mandrake Linux** distributions. You are invited in particular to participate in the various Mailing lists (<http://www.mandrakelinux.com/en/flists.php3>), where the **Mandrake Linux** community demonstrates its vivacity and keenness.

Finally, do not forget to connect to MandrakeSecure (<http://www.mandrakesecure.net/>). This site gathers all security related material about **Mandrake Linux** distributions. You'll notably find there security and bug advisories, as well as security and privacy-related articles. A must for any server administrator or user concerned about security.

2.2. Support Mandrake

By popular request, **MandrakeSoft** proposes that its happy customers make a donation (<http://www.mandrakelinux.com/donations/>) to support the forth-coming developments of the **Mandrake Linux** system. Your contribution will help **MandrakeSoft** provide its users with an ever better distribution, ever safer, easier, up-to-date, and with more supported languages.

For the many talented, your skills will be very useful for one of the many tasks required in the making of a **Mandrake Linux** system:

- Packaging: a *GNU/Linux* system is mainly made of programs picked up on the Internet. These programs have to be packaged so that they will hopefully work together.
- Programming: there are many many projects directly supported by **MandrakeSoft**: find the one that most appeals to you, and offer your help to the main developer.
- Internationalization: translation of the web pages, programs and their respective documentation.
- Documentation: last but not least, the book you are currently reading requires a lot of effort to stay up-to-date with the rapid evolution of the system.

Consult the contributors page (<http://www.mandrakesoft.com/labs/>) to learn more about the way you can contribute to the evolution of **Mandrake Linux**.

On August 3rd 2001, after having established itself as one of the world leaders in Open Source and *GNU/Linux* software, **MandrakeSoft** became the first *Linux* company listed on a European stock market. Whether you're already a **MandrakeSoft** shareholder or wish to become one, our Investor pages (<http://www.mandrakesoft.com/company/investors>) provide the best financial information related to the company.

2.3. Purchasing Mandrake Products

For **Mandrake Linux** fans wishing to benefit from the ease of on-line purchasing, **MandrakeSoft** now sells its products worldwide from its MandrakeStore (<http://www.mandrakestore.com/>) e-commerce web site. You will find not only **Mandrake Linux** software — operating systems and network tools (Single Network Firewall), but also special subscription offers, support, third party software and licenses, training documentation, *GNU/Linux* related books, as well as other goodies related to **MandrakeSoft**.

3. Authors And Translators

The following people contributed to the making of the **Mandrake Linux** manuals:

- Yves Bailly
- Camille Bégnis
- Marco De Vitis
- Francis Galiègue
- Hinrich Göhlmann
- Carsten Heimig
- Fabian Mandelbaum
- Joël Pomerleau
- Peter Rait
- Roberto Rosselli Del Turco
- Christian Roy

- Stefan Siegel

These people also participated at various degrees: Philippe Ambon, Jay Beale, Hoyt Duff, Joël Flores-Carpio, Giuseppe Ghibò, Till Kampetter, Alexander Sasha Kirillov, Damien Dams Krotkine, Robert Kulagowski, Kevin Lecouvey, François Pons, Guillaume Poulin, Pascal Pixel Rigaux, John Rye and Laurence Tricon.

4. Tools Used in The Making of This Manual

This manual was written in *DocBook*. *perl* and *GNU make* were used to manage the sets of files involved. The XML source files were processed by *openjade* and *jadetex* using custom Norman Walsh's stylesheets. Screen-shots were taken using *xwd* or *GIMP* and converted with *convert* (from the *ImageMagick* package). All this software is available on your **Mandrake Linux** distribution, and all parts of it are free software.

5. Note From The Editor

As you may notice while you go from one chapter to another, this book is a composite document from various authors. Even though much care has been taken in insuring the technical and vocabulary consistency, the style of each author is obviously preserved.

Some of the authors write in English even though it is not their native language. Therefore, you may notice strange sentence constructions; do not hesitate to let us know if something is not clear to you.

In the open-source philosophy, contributors are always welcomed! You may provide help to this documentation project by many different means. If you have a lot of time, you can write a whole chapter. If you speak a foreign language, you can help with the internationalization of this book. If you have ideas on how to improve the content, let us know - even advice on typos is welcomed!

For any information about the **Mandrake Linux** documentation project, please contact the documentation administrator (<mailto:documentation@mandrakesoft.com>).

6. Conventions Used in This Book

6.1. Typing Conventions

In order to clearly differentiate special words from the text flow, the documentation team uses different renderings. The following table shows an example of each special word or group of words with its actual rendering and what this means.

Formatted Example	Meaning
<i>inode</i>	This formatting is used to stress a technical term, explained in the <i>Glossary</i> .
<code>ls -lta</code>	Indicates commands or arguments to a command. This formatting is applied to commands, options and file names. Also see the section about “ <i>Commands Synopsis</i> , page iv”.
<code>ls(1)</code>	Reference to a man page. To get the page in a <i>shell</i> (or command line), simply type <code>man 1 ls</code> .
<pre>\$ ls *.pid inwheel.pid</pre>	The documentation team uses this formatting for text snapshots of what you may see on your screen. It includes computer interactions, program listings, etc.
<code>localhost</code>	This is literal data that does not generally fit in with any of the previously defined categories. For example, a key word taken from a configuration file.
<i>Apache</i>	This is used for application names. The example used is not a command name but, in particular contexts, the application and command name may be the same but formatted in different ways.
<u>Files</u>	This is used for menu entries or graphical interface labels in general. The underlined letter indicates the keyboard shortcut, if applicable.
<i>SCSI-Bus</i>	It denotes a computer part or a computer itself.

Formatted Example	Meaning
<i>Le petit chaperon rouge</i>	This formatting identifies foreign language words.
Warning!	Of course, this is reserved for special warnings in order to stress the importance of words; read out loud :-)



This icon highlights a note. Generally, it is a remark in the current context, giving additional information.



This icon represents a tip. It can be a general advice on how to perform a specific action, or a nice feature that can make your life easier.



Be very careful when you see this icon. It always means that very important information about a specific subject will be dealt with.

6.2. General Conventions

6.2.1. Commands Synopsis

The example below shows you the symbols you will find when the writer describes the arguments of a command:

```
command <non literal argument>
[--option={arg1,arg2,arg3}] [optional arg. ...]
```

These conventions are standard and you may find them at other places such as the man pages.

The “<” (lesser than) and “>” (greater than) symbols denote a **mandatory** argument not to be copied verbatim, but to be replaced according to your needs. For example, <filename> refers to the actual name of a file. If this name is foo.txt, you should type foo.txt, and not <foo.txt> or <filename>.

The square brackets “[]” denote optional arguments, which you may or may not include in the command.

The ellipsis “...” mean an arbitrary number of items can be included.

The curly brackets “{ }” contain the arguments authorized at this specific place. One of them is to be placed here.


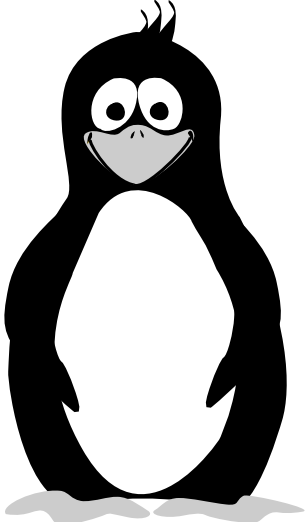
6.2.2. Special Notations

From time to time, you will be directed to press, for example, the keys Ctrl+R, which means you need to press and hold the Ctrl and tap the R key as well. The same applies for the Alt and Shift keys.

Also about menus, going to menu item **File→Reload user config (Ctrl+R)** means: click on the **File** text displayed on the menu (generally horizontal on the top of the window). Then in the pull-down menu, click on the **Reload user config** item. Additionally, you are informed that you can use the key combination Ctrl+R, as described above, to achieve the same result.

6.2.3. System Generic Users

Whenever possible, we used two generic users in our examples:

Queen Pingusa		This user is created at installation time.
Peter Pingus		This user is created afterwards by the system administrator.

7. Introduction

Welcome, and thank you for using **Mandrake Linux**! This book is aimed at people wishing to dive into the depths of their *GNU/Linux* system, and exploiting its huge capabilities. This book is made up of three parts:

- *Introduction to Linux*: an introduction to the command line, its various uses, and to text-editing basics, which are essential under *GNU/Linux*.

The first chapter introduces you to the *UNIX* and, more specifically, *GNU/Linux* worlds. It is mandatory to fully understand the concepts presented in this chapter before going on to the next one, which is dedicated to the command line. The latter exposes the standard utilities for manipulating files as well as some useful features provided by the *shell*.

Next, we cover text editing. As most *UNIX* configuration files are text files, you will eventually want to edit them in a *text editor*. You will learn how to use two of the most famous text editors in the *UNIX* and *GNU/Linux* worlds: the mighty *Emacs* and the modern (!) *Vi*.

You should now be able to perform basic maintenance on your system. The following two chapters present practical uses of the command line, and process control in general.

- *Linux in Depth*: we discuss details about the *Linux* kernel and filesystem architecture.

The first chapter explores the organization of the file tree. *UNIX* systems tend to grow very large, but every file has its place in a specific directory. After reading this chapter, you will know where to look for files depending on their role in the system.

Then, we cover the topics of *filesystems* and *mount points*. We define both these terms. We also explain them with practical examples.

The next chapter is dedicated to *GNU/Linux* filesystems. After presenting the available filesystems, we discuss file types and some additional concepts that may be new to you (. Another chapter will introduce the special *GNU/Linux* filesystem, that is the */proc* filesystem.

Following is the **Mandrake Linux** boot-up procedure, and how to use it efficiently.

- *Advanced Uses*: we close out with topics for advanced users, including a very useful troubleshooting chapter.

We begin by two practical chapters dedicated to printing and printer management.

We continue concerning security levels available on a **Mandrake Linux** system. It is followed by two chapters for geek apprentices: how to compile and install a new kernel and free software.

We close out our guide with troubleshooting; if you start experiencing problems with your system, this is a good starting point.

Enjoy!

I. Introduction to Linux

Chapter 1. Basic UNIX System Concepts

The name “*UNIX*” may be familiar to some of you. You may even use a *UNIX* system at work, in which case this chapter may not be very interesting to you.

For those of you who never used it, reading this chapter is absolutely necessary. Knowing the concepts which will be introduced here answers a surprisingly high amount of questions commonly asked by beginners in the GNU/Linux world. Similarly, some of these concepts will likely answer most of the problems you may encounter in the future.

1.1. Users and Groups

The users and groups concepts are extremely important, for they have a direct influence on all other concepts this chapter will introduce.

Linux is a true *multiuser* system, and in order to use your *GNU/Linux* machine, you must have an *account* on the said machine. When you created a user during the installation, you actually created a user account. You were prompted for the following items:

- the “real name” of the user (whatever you want, in fact);
- a *login* name;
- a *password* (you did enter one, didn’t you?).

The two important parameters here are the login name (commonly abbreviated to login) and password. In order to access the system, you absolutely need these.

When you create a user, a default group is also generated . As we will see later, groups are useful when you have to share files between several people. A group may contain as many users as you wish, and it is very common to see such a separation in large systems. For example, in a university, you could have one group per department, another group for teachers, and so on. The opposite is also true: a user can be a member of one or more groups, with a maximum of thirty-two. A math teacher, for example, can be a member of the teachers’ group and also in his beloved math students’ group.

All this does not tell you how to log in, however. So, here it comes.

If you chose to use the graphical interface upon boot-up, your startup screen will look similar to figure 1-1.



Figure 1-1. Graphical-Mode Login Session

In order to login, you must enter your login name into the **Login:** text field, then enter your password into the password field. Note that you must type your password blindly; there will be no *echo* into the text field.

If you are in console mode, your screen will look similar to figure 1-2.

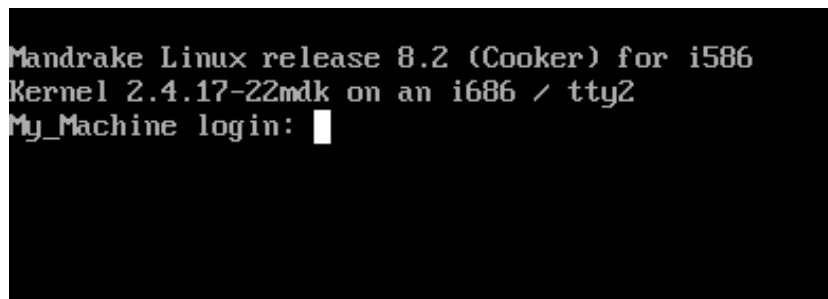


Figure 1-2. Console Mode Login Session

You will then have to enter your login name at the **Login:** prompt and press Enter, after which the login program (called, what a surprise, *login*) will display a **Password:** prompt, where you will enter the password for this account. Because the console login does not echo characters that represent the password, be careful when typing in your password... blindly.

Note that you can login several times with the same account on additional *consoles* and under *X*. Each session you open is independent from others, and it is even possible to have several *X* sessions opened concurrently. By default, **Mandrake Linux** has six *virtual consoles* in addition to the one reserved for the graphical interface. You can switch to any of them by typing the key sequence Alt-F<n>, where <n> is the number of the console which you want to switch to. By default, the graphical interface is on console number 7.

During the installation, *DrakX* also prompted you for the password of a very special user: *root*. *root* is the system administrator, which will most likely be you. For your system's security, it is very important that the *root* account always be protected with a good password!

If you regularly login as *root*, it is very easy to make a mistake which can render your system unusable; one single mistake can do it. In particular, if you have not set a password for the *root* account, then **any** user can alter **any** part of your system (even other operating systems on your machine!). This is obviously not a very good idea.

It is worth mentioning that, internally, the system does not identify you with your login name but with a unique number assigned to this login name: the *User ID* (UID for short). Similarly, every group is identified by its *Group ID* (GID) and not its name.

1.2. File Basics

Files are another topic where *GNU/Linux* differs greatly from *Windows* and most other *operating systems*. We will cover the most obvious differences here. For more information, see the The Linux Filesystem chapter, which offers greater detail.

The major differences result directly from the fact that *Linux* is a multiuser system: every file is the exclusive property of one user and one group. And one thing we did not mention about users and groups is that every one of them possesses a personal directory (called *home directory*). He is the owner of this directory, and of all files he will subsequently create.

However, this would not be very useful if there were only that notion of file ownership. But there is more: as the file owner, a user can set **permissions** on the files. These permissions distinguish between three categories of users: the owner of the file, every user who is a member of the group associated with the file (also called the *owner group*) but who is not the owner, and others, which includes every other user who is neither the owner nor member of the owners' group.

There are three different permissions:

1. *Read* permission (r): it enables the contents of a file to be read. For a directory, this allows its contents (i.e. the files in this directory) to be listed.
2. *Write* permission (w): it allows the modification of a file's contents. For a directory, the write permission allows a user to add and/or remove files from this directory, even if they are not the owner of these files.
3. *eXecute* permission (x): it enables a file's execution (as a consequence, only executable files should normally have this permission set). For a directory, this allows a user to *traverse* it (which means going into or through that directory). Note that this is separated from the read access: it may very well be that you can traverse a directory but cannot read its contents!

Every combination of these permissions is possible. For example, you can allow only yourself to read the file and forbid it to all other users. You can even do the opposite, even if it's not very logical at a first glance... As the file owner, you can also change the owner group (if and only if you are a member of the new group), and even deprive yourself of the file (that is, change its owner). Of course, if you deprive yourself of the file, you will lose all your rights to it...

Let's take the example of a file and a directory. The display below represents entering the `ls -l` command from the *command line*:

```
toi$ ls -l
total 1
-rw-r----- 1 queen    users      0 Jul  8 14:11 a_file
drwxr-xr--  2 peter    users    1024 Jul  8 14:11 a_directory/
$
```

The results of the `ls -l` command are (from left to right):

- the first ten characters represent the file's type and the permissions associated to it. The first character is the file's type: if it is a regular file, it will contain a dash (-). If it's a directory, you will see this character: d. There are other file types, which we will talk about in the *Reference Manual*. The nine following characters represent the permissions associated to that file. Here you can see the distinction which is made between different users for the same file: the first three characters represent the rights associated to the file owner, the next three apply to all users belonging to the group but who are not the owner, and the last three apply to others. A dash (-) means that the permission is not set;
- following this are the number of links for the file. We will see in the *Reference Manual* that the unique identifier of a file is not its name, but a number (the *inode number*), and that it is possible for one file on disk to have several names. For a directory, the number of links has a special meaning, which we will also discuss in the *Reference Manual*;
- following this is the name of the file owner and the name of the owner group;
- finally, the size of the file (in *bytes*) and its last modification time are displayed, followed lastly by the name of the file or directory itself.

Let us now look closely at the permissions associated to each of these files: first of all, we must strip off the first character representing the type, and for the file `a_file`, we get the following rights: `rw-r-----`. The interpretation of these permissions is as follows:

- the first three characters (`rw-`) are the file owner's rights, in this case queen. Therefore, queen has the right to read the file (`r`), to modify its contents (`w`) but not to execute it (`-`);
- the next three characters (`r--`) apply to any user who is not queen but who is a member of the `users` group : such a user will be able to read the file (`r`), but neither write nor execute it (`--`);
- the last three characters (`---`) apply to any user who is not queen and is not a member of the `users` group: such a user will simply have no rights on the file at all.

For the directory `a_directory`, the rights are `rwxr-xr--`, and as such:

- peter, as the directory owner, can list files contained inside (`r`), add or remove files from that directory (`w`), and he can traverse it (`x`);
- each user who is not peter, but a member of the `users` group, will be able to list files in this directory (`r`), but not remove nor add files (`-`), and will be able to traverse it (`x`);
- every other user will only be able to list the contents of this directory (`r`), but that's all. He will not even be able to enter the directory.

There is **one** exception to these rules: `root`. `root` can change attributes (permissions, owner and group owner) of all files, even if he's not the owner. This means that he can also grant himself the ownership! He can read files on which he has no read permission, traverse directories which he would normally have no access to, and so on. And if he lacks a permission, he just has to add it...

Lastly, it is worth noting the differences between file names in the *UNIX* and the *Windows* worlds. For one, *UNIX* allows for a much greater flexibility and has less limitations:

- a file name may contain any character (except ASCII character 0, which is the end of a string, and /, which is the directory separator), even non-printable ones. Moreover, *UNIX* is case sensitive: the files *readme* and *Readme* are different, because *r* and *R* are read as two different **characters** under *UNIX*-based systems.
- As you may have noticed, a file name does not have to include an extension unless you prefer it that way. File extensions do not identify the contents of files under *GNU/Linux*, and neither do they on any operating system for that matter. So-called “file extensions” are always very convenient, though. The period (.) under *UNIX* is just one character among others. It is worth mentioning that file names beginning with a period under *UNIX* are “hidden files”;

1.3. Processes

A *process* defines an instance of a program being executed and its *environment*. As for files, we will only mention the most important differences; you should refer to the *Reference Manual* for a more in-depth discussion on this subject.

The most important difference is, once again, directly related to the user concept: each process is executed with the rights of the user whom launched it. Internally, the system identifies processes in a unique way: with a number. This number is called the *process ID*, or *PID*. From this *PID*, the system knows, among other things, who (which user, that is) has launched the process. Then, it just has to verify the process’ “validity”. Therefore, if we take our *a_file* example, a process launched by the *peter* will be able to open this file in *read-only mode*, but not in *read-write mode*, as the rights associated to the file forbid it. Once again, the exception to the rule is *root*...

thanks to this, *GNU/Linux* is virtually immune against viruses. In order to operate, viruses must infect executable files. As a user, you do not have any write access to vulnerable system files, so the risk is greatly reduced. Add to this the fact that generally speaking, viruses are very rare in the *UNIX* world. So far, there have been only three known viruses for *Linux*, and they were completely harmless when launched by a normal user. Only one user can damage a system by activating these viruses, and once again, it is *root*.

Interestingly enough, anti-virus software does exist for *GNU/Linux*, but for *DOS/Windows* files... The reason for this is that, more and more often, you will see *GNU/Linux* file servers serving *Windows* machines with the help of the *Samba* software package.

Linux makes it easy to control processes. One way to control them are signals. With signals, you can, for example, suspend a process or kill it. Just send the corresponding signal to the process and you are done. However, you are limited to sending signals to your own processes, not processes launched by any other user. The exception to this rule is *root*. Yes, him again! In “*Process control*”, page 33, you will learn how to obtain the *PID* of a process and send it signals.

1.4. Small Introduction to the Command Line

The command line is the most direct way to send commands to your machine. If you use the *GNU/Linux* command line, you will soon find that it is much more powerful and capable than other command prompts you may have previously used. The reason for this is that you have direct access, not only to all *X* applications, but also to thousands of utilities in console mode (as opposed to graphical mode) which do not have graphical equivalents, or the many options and possible combinations of which would be hard to access in the form of buttons or menus.

But, admittedly, it requires a little help to get started. The first thing to do is to launch a terminal emulator. While accessing either your *GNOME* or *KDE* application menu, you will find these in **Terminal**. Then, choose the one you want, for example *Terminal* or *xterm*. You also have an icon which clearly identifies it on the panel (figure 1-3).

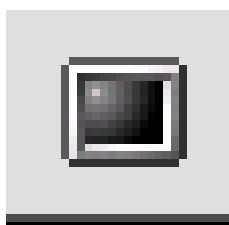


Figure 1-3. The Terminal Icon on the KDE Panel

When you launch a this terminal emulator, you actually use a *shell*. This is the name of the program with which you interact. You find yourself in front of the *prompt*:

```
[queen@localhost] ~ $
```

This assumes that your user name is queen and that your machine's name is localhost (this is the case if your machine is not part of an existing network). Typically, after the prompt is space for you to type whatever you need to type. Note that when you are root, the prompt's \$ character turns into a # (this is true only in the default configuration, since you can customize all such details in *GNU/Linux*). In order to become root, type su after launching a *shell*.

```
# Enter the root password; it will not appear on the screen
[queen@localhost] ~ $ su
Password:
# exit will make you come back to your normal user account
[root@localhost] queen # exit
[queen@localhost] ~ $
```

Anywhere else in the book, the prompt will be symbolically represented by a \$, whether you are a normal user or root. You will be told when you have to be root to execute a command, so please remember the su. However, when the # character is placed at the beginning of a line of code, it represents a comment.

When you *launch* a *shell* for the first time, you normally find yourself in your home directory. To display the directory you are currently in, type pwd (which stands for *Print Working Directory*):

```
$ pwd
/home/queen
```

Now, we will look at a few basic commands, and you will soon find that you cannot live without them!

1.4.1. cd: Change Directory

The cd command is just like *DOS*', with a few extras. It does just what its acronym states, changes the working directory. You can use . and .., which stand respectively for the current and parent directories. Typing cd alone will take you back to your home directory. Typing cd - will take you back to the last directory you visited. And lastly, you can specify the peter's home directory by typing cd ~peter (~ alone means your own home directory). Note that as a normal user, you usually cannot go into another user's home directory (unless she explicitly authorized it or if this is the default configuration on the system), except if you are root, so let's be root and practice:

```
$ pwd
/root
$ cd /usr/share/doc/HOWTO
$ pwd
/usr/share/doc/HOWTO
$ cd ../FAQ-Linux
$ pwd
/usr/share/doc/FAQ-Linux
$ cd ../../../../lib
$ pwd
/usr/lib
$ cd ~peter
$ pwd
/home/peter
$ cd
$ pwd
/root
```

Now, go back to being a normal user again.

1.4.2. Some Environment Variables and the echo Command

All processes have their *environment variables* and the *shell* allows you to view them directly with the `echo` command. Some interesting variables are:

1. HOME: this environment variable contains a string which represents your home directory.
2. PATH: this variable holds the list of all directories in which the *shell* should look for executables when you type a command. Note that unlike *DOS*, by default, a *shell* will **not** look for commands in the current directory!
3. USERNAME: this variable contains your login name.
4. UID: this one holds your user ID.
5. PS1: this variable contains your prompt's definition. It is often a combination of special sequences. You may read the `bash(1)` *manual page* for more information.

To have the *shell* print a variable's value, you must put a `$` in front of its name. Here, the `echo` command will help you:

```
$ echo Hello
Hello
$ echo $HOME
/home/queen
$ echo $USERNAME
queen
$ echo Hello $USERNAME
Hello queen
$ cd /usr
$ pwd
/usr
$ cd $HOME
$ pwd
/home/queen
```

As you can see, the *shell* substitutes the variable's value before it executes the command. Otherwise, our `cd $HOME` would not have worked here. In fact, the shell first replaced `$HOME` by its value, `/home/queen`. Therefore, the line became `cd /home/queen`, which is what we wanted. It is the same thing for `echo $USERNAME` and so on.

1.4.3. cat: Print the Contents of One or More Files to the Screen

Nothing much to say, this command does just that: print the contents of one or more files to the standard output, normally the screen:

```
$ cat /etc/fstab
/dev/hda5 / ext2 defaults 1 1
/dev/hda6 /home ext2 defaults 1 2
/dev/hda7 swap swap defaults 0 0
/dev/hda8 /usr ext2 defaults 1 2
/dev/fd0 /mnt/floppy auto sync,user,noauto,nosuid,nodev 0 0
none /proc proc defaults 0 0
none /dev/pts devpts mode=0620 0 0
/dev/cdrom /mnt/cdrom auto user,noauto,nosuid,exec,nodev,ro 0 0
$ cd /etc
$ cat modules.conf shells
alias parport_lowlevel parport_pc
pre-install plip modprobe parport_pc ; echo 7 > /proc/parport/0/irq
#pre-install pcmcia_core /etc/rc.d/init.d/pcmcia start
#alias char-major-14 sound
alias sound esssolo1
keep
/bin/zsh
/bin/bash
/bin/sh
/bin/tcsh
/bin/csh
/bin/ash
/bin/bsh
```

```
/usr/bin/zsh
```

1.4.4. less: a Pager

Its name is a play on words related to the first pager ever used under *UNIX*, which was called *more*. A *pager* is a program which allows a user to view long files page by page (more accurately, screen by screen). We speak here about *less* rather than *more* because its use is much more intuitive. You should use *less* to view large files, which do not fit on a screen. For example:

```
less /etc/termcap
```

To browse through this file, use the up and down arrow keys. Use **q** to quit. However, *less* can do far more than just that. In fact, just type **h** for help, and take a look. But anyway, the goal of this section is just to enable you to read long files, so we already reached our goal :-)

1.4.5. ls: Listing Files

The *ls* (*LiSt*) command is equivalent to *DOS*' *dir* command, but it can do much much more. In fact, this is largely due to the fact that files can do more too. The command syntax for *ls* is:

```
ls [options] [file|directory] [file|directory...]
```

If no file or directory is specified on the command line, *ls* will list files in the current directory. Its options are very numerous and we will only explain a few of them:

- **-a**: lists all files, including *hidden files* (remember that in *UNIX*, hidden files are those whose names begin with a **.**); the **-A** option lists "almost" all files, which means every file the **-a** option would print except **."** and **.."**;
- **-R**: lists recursively, i.e. all files and subdirectories of directories mentioned on the command line;
- **-s**: prints the file size in kilobytes next to each file;
- **-l**: prints additional information about the files;
- **-i**: prints the inode number (the file's unique number on a file system, see chapter The Linux Filesystem) next to each file;
- **-d**: treats directories on the command line as if they were normal files, instead of listing their contents.

Some examples:

- **ls -R**: lists the contents of the current directory recursively;
- **ls -is images/ ..**: lists files in the *images/* directory and in the parent directory of the current one. Then, it prints, for each file, the inode number and its size in kilobytes;
- **ls -al images/*.png**: lists all files (including any hidden files) in the *images/* directory whose names end with **.png**. Note that this also includes the file **.png**, if one exists.

1.4.6. Useful Keyboard Shortcuts

Many keystrokes are available and their main advantage is that they save you a lot of typing time. This section assumes you are using the default *shell* provided with **Mandrake Linux**, *bash*, but these keystrokes should work with other shells too.

First: the arrow keys. *bash* maintains a history of previous commands which you can view with the up and down arrow keys. You can scroll up to a maximum number of lines defined in the HISTSIZE environment variable. Moreover, the history is persistent from one session to another, so you will not lose the commands you typed in previous session.

The left and right arrow keys move the cursor left and right on the current line, so you can edit your commands this way. But there is more to editing: Ctrl+A and Ctrl+E, for example, will bring you respectively to the beginning and the end of the current line. The Backspace and Del keys work as expected. An equivalent to Backspace is Ctrl+H and an equivalent to Del is Ctrl+D. Ctrl+K will delete all the line from the position of the cursor to the end of line, and Ctrl+W will delete the word before the cursor.

Typing Ctrl+D on a blank line will make you close the current session, which is much shorter than having to type *exit*. Ctrl+C will interrupt the currently running command, except if you were in the process of editing your command line, in which case it will cancel the editing and get you back to the prompt. Ctrl+L clears the screen.

Finally, there is the case of Ctrl+S and Ctrl+Q: these keystrokes respectively suspend and restore the flow of characters on a terminal. They are very seldom used, but it may happen that you type Ctrl+S by mistake (after all, **S** and **D** are very close to each other on a keyboard...). So, if you strike keys but do not see any character appearing on the *Terminal*, try Ctrl+Q first and beware: all characters you typed between the unwanted Ctrl+S and Ctrl+Q will be printed to the screen all at once.

Chapter 2. Introduction to The Command Line

In the chapter “*Basic UNIX System Concepts*”, page 1, you were shown how to launch a *shell*. In this chapter, we will show you how to work with it.

The *shell*'s main asset is the number of existing utilities: there are thousands of them, and each one is devoted to a particular task. We will only look at a (very) small number of them here. One of *UNIX*'s greatest assets is the ability to combine these utilities, as we shall see later.

2.1. File-Handling Utilities

In this context, file handling means copying, moving and deleting files. Later, we will look at ways of changing their attributes (owner, permissions).

2.1.1. mkdir, touch: Creating Empty Directories And Files

`mkdir` (*MaKe DIRectory*) is used to create directories. Its syntax is simple:

```
mkdir [options] <directory> [directory ...]
```

Only one option is worth noting: the `-p` option. It does two things:

1. it will create parent directories if these did not exist before. Without this option, `mkdir` would just fail, complaining that the said parent directories do not exist;
2. it will return silently if the directory which you want to create already exists. Similarly, if you did not specify the `-p` option, `mkdir` will send back an error message, complaining that the directory already exists.

Here are some examples:

- `mkdir foo`: creates a directory `foo` in current directory;
- `mkdir -p images/misc docs`: creates the `misc` directory in the `images` directory. First, it creates the latter if it does not exist (`-p`); it also creates a directory named `docs` in the current directory.

Initially, the `touch` command is not intended for creating files but for updating file access and modification times¹. However, `touch` will create the files mentioned as empty files if they did not exist before. The syntax is:

```
touch [options] file [file...]
```

So running the command:

```
touch file1 images/file2
```

will create an empty file called `file1` in the current directory and an empty file `file2` in directory `images`.

2.1.2. rm: Deleting Files or Directories

The `rm` command (*ReMove*) replaces the *DOS* commands `del` and `deltree`, and adds more options. Its syntax is as follows:

```
rm [options] <file|directory> [file|directory...]
```

Options include:

- `-r`, or `-R`: delete recursively. This option is **mandatory** for deleting a directory, empty or not. However, you can also use `rmdir` to delete empty directories.
- `-i`: request confirmation before each deletion. Note that by default in **Mandrake Linux**, `rm` is an *alias* to `rm -i`, for safety reasons (similar aliases exist for `cp` and `mv`). Your mileage may vary as to the usefulness of these aliases. If you want to remove them, you can edit your `.bashrc` and add this line: `unalias rm cp mv`.

1. In *UNIX*, there are three distinct timestamps for each file: the last file access date (`atime`), i.e. the last date when the file was opened for read or write; the last date when the inode attributes were modified (`mtime`); and finally, the last date when the **contents** of the file were modified (`ctime`).

- `-f`, the opposite of `-i`, forces deletion of the files or directories, even if the user has no write access on the files².

Some examples:

- `rm -i images/*.jpg file1`: deletes all files with names ending in `.jpg` in the `images` directory and deletes the `file1` file in the current directory, requesting confirmation for each file. Answer `y` to confirm deletion, `n` to cancel.
- `rm -Rf images/misc/ file*`: deletes, without requesting confirmation, the whole directory `misc/` in the `images/` directory, together with all files in the current directory whose names begin with `file`.



Using `rm` deletes files **irrevocably**. There is no way to restore them!
Do not hesitate to use the `-i` option to ensure you do not delete something by mistake.

2.1.3. mv: Moving or Renaming Files

The syntax of the `mv` (*MoVe*) command is as follows:

```
mv [options] <file|directory> [file|directory ...] <destination>
```

Some options:

- `-f`: forces operation – no warning if an existing file is overwritten.
- `-i`: the opposite. Asks the user for confirmation before overwriting an existing file.
- `-v`: *verbose* mode, report all changes and activity.

Some examples:

- `mv -i /tmp/pics/*.png .`: move all files in the `/tmp/pics/` directory whose names end with `.png` to the current directory (`.`), but request confirmation before overwriting any files.
- `mv foo bar`: rename file `foo` to `bar`. If a `bar` directory already existed, the effect of this command would be to move the whole `foo` directory (the directory itself plus all files and directories in it, recursively) into the `bar` directory.
- `mv -vf file* images/ trash/`: move, without requesting confirmation, all files in the current directory whose names begin with `file`, together with the entire `images/` directory to the `trash/` directory, and show each operation carried out.

2.1.4. cp: Copying Files and Directories

`cp` (*CoPy*) replaces the *DOS* commands `copy`, `xcopy` and adds more options. Its syntax is as follows:

```
cp [options] <file|directory> [file|directory ...] <destination>
```

`cp` has a lot of options. Here are the most common:

- `-R`: recursive copy; **mandatory** for copying a directory, even empty.
- `-i`: request confirmation before overwriting any files which might be overwritten.
- `-f`: the opposite of `-i`, replaces any existing files without requesting confirmation.
- `-v`: verbose mode, displays all actions performed by `cp`.

Some examples:

2. It is enough for the user to have write access to the **directory** to be able to delete files in it, even if he is not the owner of the files.

- `cp -i /tmp/images/* images/`: copies all files in the `/tmp/images/` directory to the `images/` directory located in the current directory. It requests confirmation if a file is going to be overwritten.
- `cp -vR docs/ /shared/mp3s/* mystuff/`: copies the whole `docs` directory, plus all files in the `/shared/mp3s` directory to the `mystuff` directory.
- `cp foo bar`: makes a copy of the `foo` file under the name `bar` in the current directory.

2.2. Handling File Attributes

The series of commands shown here are used to change the owner or owner group of a file or its permissions. We looked at the different permissions in chapter Basic UNIX System Concepts of the *User Guide*.

2.2.1. `chown`, `chgrp`: Change The Owner And Group of One or More Files

The syntax of the `chown` (*CHange OWNer*) command is as follows:

```
chown [options] <user[.group]> <file|directory> [file|directory...]
```

The options include:

- `-R`: recursive. To change the owner of all files and subdirectories in a given directory.
- `-v`: verbose mode. Displays all actions performed by `chown`; reports which files have changed owner as a result of the command and which files have not been changed.
- `-c`: like `-v`, but only reports which files have been changed.

Some examples:

- `chown nobody /shared/book.tex`: changes the owner of the `/shared/book.tex` file to `nobody`.
- `chown -Rc queen.music *.mid concerts/`: changes the ownership of all files in the current directory whose name ends with `.mid` and all files and subdirectories in the `concerts/` directory to user `queen` and group `music`, reporting only files affected by the command.

The `chgrp` (*CHange GRouP*) command lets you change the group ownership of a file (or files); its syntax is very similar to that of `chown`:

```
chgrp [options] <group> <file|directory> [file|directory...]
```

The options for this command are the same as for `chown`, and it is used in a very similar way. Thus, the command:

```
chgrp disk /dev/hd*
```

changes the ownership of all files in directory `/dev/` with names beginning with `hd` to group `disk`.

2.2.2. `chmod`: Changing Permissions on Files And Directories

The `chmod` (*CHange MODe*) command has a very distinct syntax. The general syntax is:

```
chmod [options] <change mode> <file|directory> [file|directory...]
```

but what distinguishes it is the different forms that the mode change can take. It can be specified in two ways:

1. in octal. The owner user permissions then correspond to figures with the form `<x>00`, where `<x>` corresponds to the permission assigned: 4 for read permission, 2 for write permission and 1 for execute permission. Similarly, the owner group permissions take the form `<x>0` and permissions for “others” the form `<x>`. Then, all you need to do is add together the assigned permissions to get the right mode. Thus, the permissions `rwxr-xr--` correspond to $400+200+100$ (owner permissions, `rw`) $+40+10$ (group permissions, `r-x`) $+4$ (others’ permissions, `r--`) = 754; in this way, the permissions are expressed in absolute terms. This means that previous permissions are unconditionally replaced;

2. with expressions. Here permissions are expressed by a sequence of expressions separated by commas. Hence an expression takes the following form: `[category]<+|-|=><permissions>`.

The category may be one or more of:

- `u` (*User*, permissions for owner);
- `g` (*Group*, permissions for owner group);
- `o` (*Others*, permissions for “others”).

If no category is specified, changes will apply to all categories. A `+` sets a permission, a `-` removes the permission and a `=` sets the permission. Finally, the permission is one or more of the following:

- `r` (*Read*);
- `w` (*Write*) or;
- `x` (*eXecute*).

The main options are quite similar to those of `chown` or `chgrp`:

- `-R`: changes permissions recursively.
- `-v`: verbose mode. Displays the actions carried out for each file.
- `-c`: like `-v` but only shows files affected by the command.

Examples:

- `chmod -R o-w /shared/docs`: recursively removes write permission for others on all files and subdirectories in the `/shared/docs/` directory.
- `chmod -R og-w,o-x private/`: recursively removes write permission for group and others for the whole `private/` directory, and removes the execution permission for others.
- `chmod -c 644 misc/file*`: changes permissions of all files in the `misc/` directory whose names begin with `file` to `rw-r--r--` (i.e. read permission for everyone and write permission only for the owner), and reports only files affected by this command.

2.3. Shell Globbing Patterns

You probably already use *globbing* characters without knowing it. When you specify a file in *Windows* or when you look for a file, you use `*` to match a random string. For example, `*.txt` matches all files with names ending with `.txt`. We also used it heavily in the last section. But there is more to globbing than just `*`.

When you type a command like `ls *.txt` and press Return, the task of finding which files match the `*.txt` pattern is not done by the `ls` command, but by the *shell* itself. This requires a little explanation about how a command line is interpreted by the *shell*. When you type:

```
$ ls *.txt
README.txt  recipes.txt
```

the command line is first split into words (`ls` and `*.txt` in this example). When the shell sees a `*` in a word, it will interpret the whole word as a globbing pattern and will replace it with the names of all matching files. Therefore, the command, just before the shell executes it, has become `ls README.txt recipe.txt`, which gives the expected result. Other characters make the shell react this way:

- `?`: matches one and only one character, regardless of what that character is;
- `[...]`: matches any character found in the brackets. Characters can be referred to either as a range of characters (i.e. 1–9) or *discrete values*, or even both. Example: `[a-zA-Z0-9]` will match all characters between `a` and `z`, `A`, `B`, `a`, `E`, `5`, `6` or `7`;

- `[!...]`: matches any character **not** found in the brackets. `[!a-z]`, for example, will match any character which is not a lowercase letter³;
- `{c1,c2}`: matches `c1` or `c2`, where `c1` and `c2` are also globbing patterns, which means you can write `{[0-9]*,[acr]}` for example.

Here are some patterns and their meaning:

- `/etc/*conf`: all files in the `/etc` directory with names ending in `conf`. It can match `/etc/inetd.conf`, `/etc/conf.linuxconf`, **and also** `/etc/conf` if such a file exists. Remember that `*` can match an empty string.
- `image/{cars,space[0-9]}/*.jpg`: all file names ending with `.jpg` in directories `image/cars`, `image/space0`, (...), `image/space9`, if those directories exist.
- `/usr/share/doc/*/README`: all files named `README` in all of `/usr/share/doc`'s immediate subdirectories. This will make `/usr/share/doc/mandrake/README` match, for example, but not `/usr/share/doc/myprog/doc/README`.
- `*[!a-z]`: all files with names that do **not** end with a lowercase letter in the current directory.

2.4. Redirections and Pipes

2.4.1. A Little More About Processes

To understand the principle of redirections and pipes, we need to explain a notion about processes which has not yet been introduced. Each *UNIX* process (this also includes graphical applications) opens a minimum of three file descriptors: standard input, standard output and standard error. Their respective numbers are 0, 1 and 2. In general, these three descriptors are associated with the terminal from which the process was started, with the input being the keyboard. The aim of redirections and pipes is to redirect these descriptors. The examples in this section will help you better understand this concept.

2.4.2. Redirections

Imagine, for example, that you wanted a list of files ending with `.png`⁴ in the `images` directory. This list is very long, so you may want to store it in a file in order to look through it at your leisure. You can enter the following command:

```
$ ls images/*.png 1>file_list
```

This means that the standard output of this command (1) is redirected (`>`) to the file named `file_list`. The `>` operator is the output redirection operator. If the redirection file does not exist, it is created, but if it exists, its previous contents are overwritten. However, the default descriptor redirected by this operator is the standard output and does not need to be specified on the command line. So you can write more simply:

```
$ ls images/*.png >file_list
```

and the result will be exactly the same. Next, you can look at the file using a text file viewer such as `less`.

Now, imagine you want to know how many of these files exist. Instead of counting them by hand, you can use the utility called `wc` (*Word Count*) with the `-l` option, which writes on the standard output the number of lines in the file. One solution is as follows:

```
wc -l 0<file_list
```

and this gives the desired result. The `<` operator is the input redirection operator, and the default redirected descriptor is the standard input one, i.e. 0, and you simply need to write the line:

3. Beware! While this is true under *GNU/Linux*, this may not be true under other *UNIX*-like operating systems. This depends on the **collating order**. On some systems, `[a-z]` will match `a`, `A`, `b`, `B`, (...), `z`. And we do not even mention the fact that some languages have accentuated characters...

4. You might think it is crazy to say "files ending with `.png`" rather than "PNG images". However, once again, files under *UNIX* only have an extension by convention: extensions in no way define a file type. A file ending with `.png` could perfectly well be a JPEG image, an application file, a text file or any other type of file. The same is true under *Windows* as well!

```
wc -l <file_list
```

Now suppose you want to remove all the file “extensions” and put the result in another file. One tool for doing this is *sed* (*Stream Editor*). You simply redirect the standard input of *sed* to the `file_list` file and redirect its output to the result file, i.e. `the_list`:

```
sed -e 's/\.png$//g' <file_list >the_list
```

and your list is created, ready for you to view at your leisure with any viewer.

It can also be useful to redirect standard errors. For example, you want to know which directories in `/shared` you cannot access: one solution is to list this directory recursively and to redirect the errors to a file, while not displaying the standard output:

```
ls -R /shared >/dev/null 2>errors
```

which means that the standard output will be redirected (`>`) to `/dev/null`, a special file in which everything you write is discarded (i.e. as a side effect, the standard output is not displayed) and the standard error channel (2) is redirected (`>`) to the `errors` file.

2.4.3. Pipes

Pipes are in some way a combination of input and output redirections. The principle is that of a physical pipe, hence the name: one process sends data into one end of the pipe and another process reads the data at the other end. The pipe operator is `|`. Let us go back to the file list example above. Suppose you want to find out directly how many corresponding files there are without having to store the list in a temporary file, you would then use the following command:

```
ls images/*.png | wc -l
```

which means that the standard output of the `ls` command (i.e. the list of files) is redirected to the standard input of the `wc` command. This then gives you the desired result.

You can also directly put together a list of files “without extensions” using the following command:

```
ls images/*.png | sed -e 's/\.png$//g' >the_list
```

or, if you want to consult the list directly without storing it in a file:

```
ls images/*.png | sed -e 's/\.png$//g' | less
```

Pipes and redirections are not restricted solely to text that can be read by human beings. For example, the following command sent from a *Terminal*:

```
xwd -root | convert - ~/my_desktop.png
```

will send a screenshot of your desktop to the `my_desktop.png`⁵ file in your personal directory.

2.5. Command-Line Completion

Completion is a very handy functionality, and all modern *shells* (including *bash*) have it. Its role is to give the user as little work to do as possible. The best way to illustrate completion is to give an example.

2.5.1. Example

Suppose your personal directory contains the `file_with_very_long_name_impossible_to_type` file, and you want to look at it. Suppose you also have, in the same directory, another file called `file_text`. You are in your personal directory, so type the following sequence:

```
$ less fi<TAB>
```

(i.e., type `less fi` and then press the `TAB` key). The *shell* will then expand the command line for you:

5. Yes, it will indeed be a PNG image :-)

```
$ less file_
```

and also give the list of possible choices (in its default configuration, which can be customized). Then type the following key sequence:

```
less file_w<TAB>
```

and the *shell* will extend the command line to give you the result you want:

```
less file_with_very_long_name_impossible_to_type
```

All you need to do then is press the Enter key to confirm and read the file.

2.5.2. Other Completion Methods

The TAB key is not the only way to activate completion, although it is the most common one. As a general rule, the word to be completed will be a command name for the first word of the command line (`ns1<TAB>` will give `nslookup`), and a file name for all the others, unless the word is preceded by a “magic” character like `~`, `@` or `$`, in which case the *shell* will try to complete a user name, a machine name or an environment variable name respectively⁶. There is also a magic character for completing a command name (`!`) or a file name (`/`).

The other two ways to activate completion are the sequences `Esc-<x>` and `Ctrl+x <x>`, where `<x>` is one of the magic characters already mentioned. `Esc-<x>` will attempt to come up with a unique completion. If it fails, it will complete the word with the largest possible substring in the choice list. A *beep* means either that the choice is not unique, or simply that there is no corresponding choice. The sequence `Ctrl+x <x>` displays the list of possible choices without attempting any completion. Pressing the TAB key is the same as successively pressing `Esc-<x>` and `Ctrl+x <x>`, where the magic character depends on the context.

Thus, one way to see all the environment variables defined is to type the sequence `Ctrl+x $` on a blank line. Another example: if you want to see the *man* page for the `nslookup` command, you simply type `man ns1` then `Esc-!`, and the shell will automatically complete the command to `man nslookup`.

2.6. Starting and Handling Background Processes: Job Control

You probably noticed that when you enter a command from a *Terminal*, you normally have to wait for the command to finish before the *shell* returns control to you. This means that you have sent the command in the *foreground*. However, there are occasions when this is not desirable.

Suppose, for example, that you decide to copy a large directory recursively to another. You also decided to ignore errors, so you redirect the error channel to `/dev/null`:

```
cp -R images/ /shared/ 2>/dev/null
```

Such a command can take several minutes until it is fully executed. You then have two solutions: the first one is violent, and means stopping (killing) the command and then doing it again when you have the time. To do this, type `Ctrl+c`: this will terminate the process and take you back to the prompt. But wait, do not do it! Read on.

Suppose you want the command to run while doing something else. The solution is then to put the process into *background*. To do this, type `Ctrl+z` to suspend the process:

```
$ cp -R images/ /shared/ 2>/dev/null
# Type C-z here
[1]+  Stopped                  cp -R images/ /shared/ 2>/dev/null
$
```

and there you are again at the prompt. The process is then on standby, waiting for you to restart it (as shown by the Stopped keyword). That, of course, is what you want to do, but in the background. Type `bg` (for *Back-Ground*) to get the desired result:

```
$ bg
[1]+ cp -R images/ /shared/ 2>/dev/null &
$
```

6. Remember: *UNIX* differentiates between uppercase and lowercase. The `HOME` environment variable and the `home` variable are not the same.

The process will then start running again as a background task, as indicated by the `&` (ampersand) sign at the end of the line. You will then be back at the prompt and able to continue working. A process which runs as a background task, or in the background, is called a *job*.

Of course, you can start processes directly as background tasks by adding an `&` character at the end of the command. For example, you can start the command to copy the directory in the background by writing:

```
cp -R images/ /shared/ 2>/dev/null &
```

If you want, you can also restore this process to the foreground and wait for it to finish by typing `fg` (*ForeGround*). To put it into the background again, type the sequence `Ctrl+z`, `bg`.

You can start several jobs this way: each command will then be given a job number. The `shell` command `jobs` lists all the jobs associated with the current `shell`. The job preceded by a `+` sign indicates the last process begun as a background task. To restore a particular job to the foreground, you can then type `fg <n>` where `<n>` is the job number, i.e. `fg 5`.

Note that you can also suspend or start *full-screen* applications this way, such as `less` or a text editor like `Vi`, and restore them to the foreground when you want.

2.7. A Final Word

As you can see, the `shell` is very comprehensive and using it effectively is a matter of practice. In this relatively long chapter, we only mentioned a few of the available commands: **Mandrake Linux** has thousands of utilities, and even the most experienced users do not make use of them all.

There are utilities for all tastes and purposes: you have utilities for handling images (like `convert` mentioned above, but also `GIMP batch` mode and all *pixmap* handling utilities), sound (MP3 encoders, audio CD players), CD writing, *e-mail* clients, FTP clients and even web browsers (like `lynx` or `links`), not to mention all the administration tools.

Even if graphical applications with equivalent functions do exist, they are usually graphical interfaces built around these very same utilities. In addition, command-line utilities have the advantage of being able to operate in non-interactive mode: you can start writing a CD and then log off the system with the confidence that the writing will take place (see the `nohup(1) man` page).

Chapter 3. Text Editing: Emacs and Vi

As stated in the introduction, text editing¹ is a fundamental feature in the use of a *UNIX* system. The two editors we are going to take a quick look at are a little difficult to use initially, but once you understand the basics, both prove to be powerful tools.

3.1. Emacs

Emacs is probably the most powerful text editor in existence. It can do absolutely everything and is infinitely extendible thanks to its built-in *lisp*-based programming language. With *Emacs*, you can move around the web, read your mail, take part in usenet newsgroups, make coffee, and so on. However, what you will be able to do at the end of this section will be limited to opening *Emacs*, editing one or more files, saving them and quitting *Emacs*. Which is not too bad to start with.

3.1.1. Short presentation

Invoking *Emacs* is relatively simple:

```
emacs [file] [file...]
```

Emacs will open every file entered as an argument in a buffer, with up to a maximum of two buffers visible at the same time, and will present you with the buffer entitled **scratch** if you do not specify a file. If you are in *X*, you also have menus available, but in this chapter we will be working with the keyboard.

3.1.2. Getting started

It is time to go hands-on. By way of example, let us open two files, *file1* and *file2*. If these two files do not exist, they will be created as soon as you write something in them:

```
$ emacs file1 file2
```

You will get the window shown in figure 3-1.

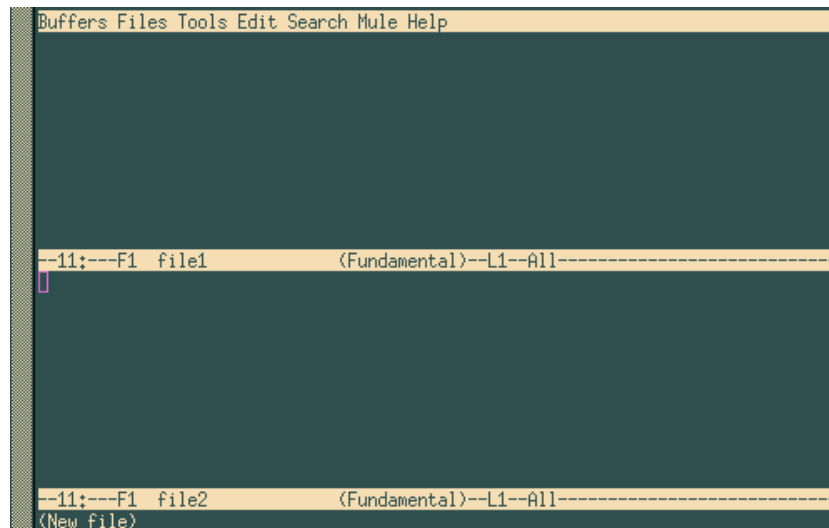


Figure 3-1. Emacs, editing two files at once

As you can see, two buffers have been created: one per file. A third is also present at the bottom of the screen (where you see *(New file)*); this is the mini-buffer. You cannot force yourself into this buffer; you must be invited by *Emacs* during interactive entries. To change the current buffer, type *Ctrl+x o*. You can type text as in a “normal” editor, deleting characters with the *DEL* or Backspace key.

1. “To edit text” means to modify the contents of a file holding only letters, digits, and punctuation signs; such files can be electronic mails, source code, documents, or even configuration files.

To move around, you can use the arrow keys, as well as these other key combinations: `Ctrl+a` to go to the beginning of the line, `Ctrl+e` to go to the end of the line, `Alt+<` to go to the beginning of the buffer and `Alt+>` to go to the end of the buffer. There are many other combinations, even for each of the arrow keys².

As soon as you want to save changes made in a file, type `Ctrl+x Ctrl+s`, or if you want to save the contents of the buffer to another file, type `Ctrl+x Ctrl+w` and *Emacs* will ask you for the name of the file to which the buffer contents are to be written. You can use *completion* to do this.

3.1.3. Handling buffers

If you want, you can show only one buffer on the screen. There are two ways of doing this:

- you are in the buffer which you want to hide: type `Ctrl+x 0`;
- you are in the buffer which you want to keep on the screen: type `Ctrl+x 1`.

There are then two ways to restore the buffer that you want on the screen:

- type `Ctrl+x b` and enter the name of the buffer you want,
- type `Ctrl+x Ctrl+b`, a new buffer will then be opened, called **Buffer List**; you can move around this buffer using the sequence `Ctrl+x o`, then select the buffer you want and press the `Enter` key, or else type the name of the buffer in the mini-buffer. The buffer **Buffer List** returns to the background once you have made your choice.

If you have finished with a file and want to get rid of the associated buffer, type `Ctrl+x k`. *Emacs* will then ask you which buffer it should close. By default, it is the name of the buffer you are currently in. If you want to get rid of a buffer other than the one suggested, enter its name directly or press `TAB`: *Emacs* will then open yet another buffer called **Completions** giving the list of possible choices. Confirm the choice with the `Enter` key.

You can also restore two visible buffers to the screen at any time. To do this type `Ctrl+x 2`. By default, the new buffer created will be a copy of the current buffer (which enables you, for example, to edit a large file in several places “at the same time”), and you simply proceed as described previously to move between buffers.

You can open other files at any time, using `Ctrl+x Ctrl+f`. *Emacs* will then ask you for the filename and here again completion is available.

3.1.4. Copy, cut, paste, search

Suppose we are in the situation of figure 3-2.

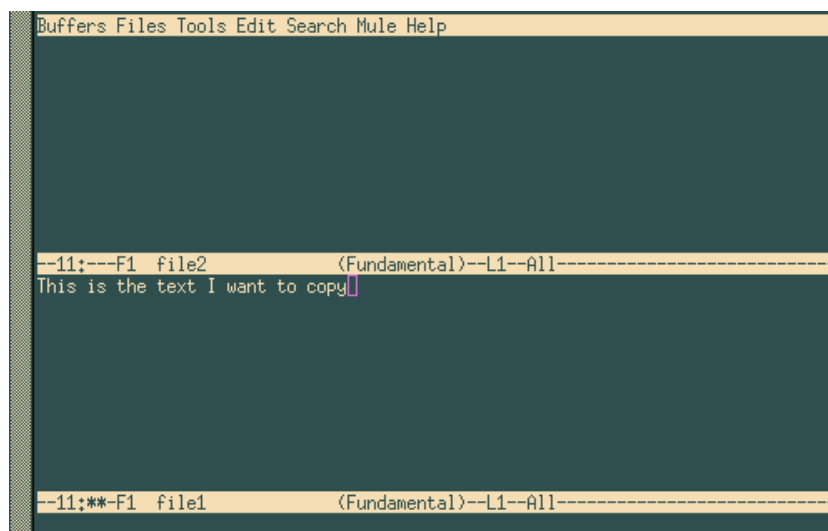


Figure 3-2. Emacs, before copying the text block

² *Emacs* has been designed to work on a great variety of machines, some of which don't even have arrow keys. This is even more true of *Vi*.

First, you need to select the text that you want to copy. In *X*, you can do it using the mouse, and the area selected will even be highlighted. But here we are in text mode :-). In this case, we want to copy the whole sentence. First, you need to place a mark to select the beginning of the area. Assuming the cursor is in the position where it is in the figure above, first type `Ctrl+SPACE` (Control + space bar); *Emacs* will then display the message `Mark set` in the mini-buffer. Then move to the beginning of the line with `Ctrl+a`. The area selected for copying or cutting is the whole area located between the mark and the cursor's current position, hence in the present case the whole line. Next type `Alt+w` (to copy) or `Ctrl+w` (to cut). If you copy, *Emacs* will return briefly to the mark position, so that you can view the selected area.

Then go to the buffer to which you want to copy the text, and type `Ctrl+y`, to obtain what is displayed in figure 3-3.

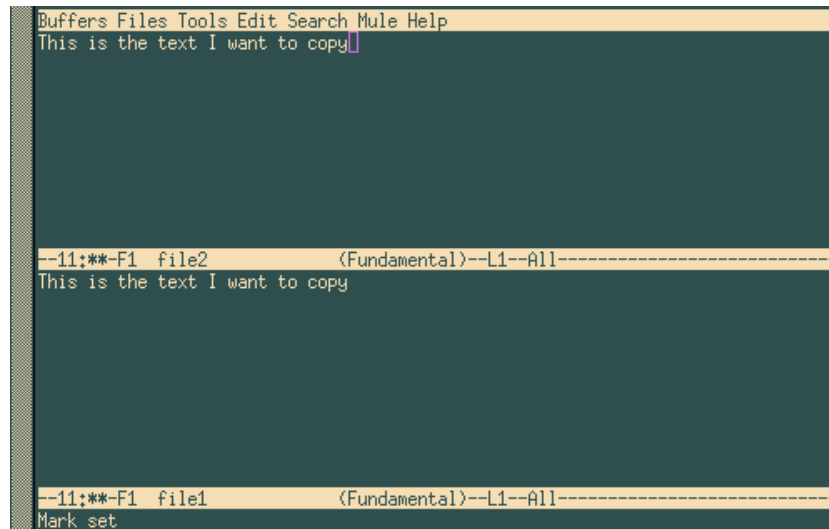


Figure 3-3. Emacs, after having copied the text block

In fact, what you have just done is copied text to the “kill ring” of *Emacs*; this kill ring contains all the areas copied or cut since *Emacs* was started. **Any** area just copied or cut is placed at the top of the kill ring. The sequence `Ctrl+y` only “pastes” the area at the top. If you want to have access to the other areas, press `Ctrl+y` then `Alt+y` until you get to the desired text.

To search for text, go into the desired buffer and type `Ctrl+s`. *Emacs* will then ask you what string to search for. To start a new search with the same string, still in the current buffer, type `Ctrl+s` again. When *Emacs* reaches the end of the buffer and finds no more occurrences, you can type `Ctrl+s` again to restart the search from the beginning of the buffer. Pressing the Enter key ends the search.

To search and replace, type `Alt+%`. *Emacs* asks you what string to search for, what to replace it with, and asks for confirmation for each occurrence it finds.

A final very useful thing: `Ctrl+x u` undoes the previous operation. You can undo as many operations as you want.

3.1.5. Quit Emacs

The shortcut for this is `Ctrl+x Ctrl+c`. *Emacs* then asks you whether you want to save the changes made to the buffers if you have not saved them.

3.2. Vi: the ancestor

Vi was the first full-screen editor in existence. *Vi* makes for one of the main objections of *UNIX* detractors, but also one of the main arguments of its defenders: while it is complicated to learn, it is also an extremely powerful tool once one gets into the habit of using it. With a few keystrokes, a *Vi* user can move mountains, and apart from *Emacs*, few text editors can boast the same.

The version supplied with **Mandrake Linux** is in fact *vim*, for *VI iMproved*, but we will call it *Vi* throughout this chapter.

3.2.1. Insert mode, command mode, ex mode...

First, we need to start *Vi*, which is identical to *Emacs*. So let us go back to our two files and type:

```
$ vi file1 file2
```

At this point, you find yourself in front of a window resembling figure 3-4.

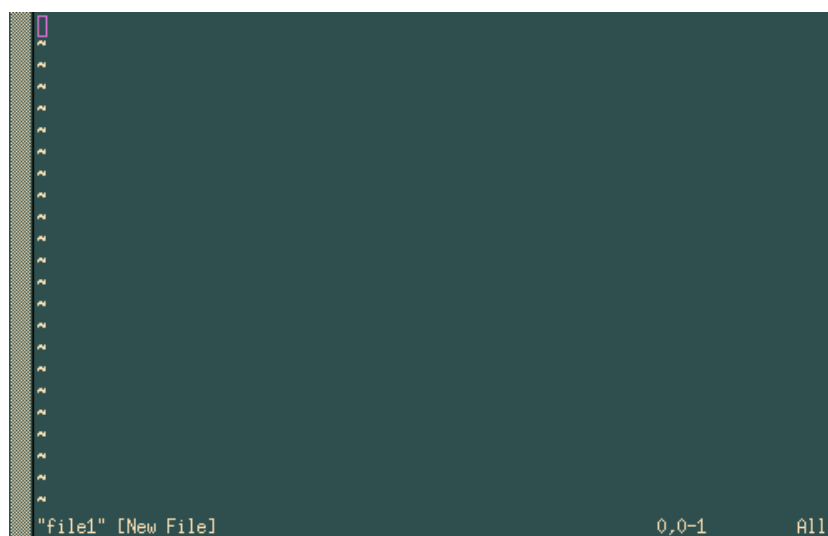


Figure 3-4. Starting position in vim

You are now in *command mode* in front of the first open file. In command mode, you cannot insert text into a file... To do this, you have to go into *insert mode*, and therefore enter one of the commands which allows you to do so:

- **a** and **i**: to insert text respectively after and before the cursor (**A** and **I** insert text at the end and at the beginning of the current line);
- **o** and **O**: to insert text respectively below and above the current line.

In insert mode, you will see the string `--INSERT--` appear at the bottom of the screen (so you know what mode you are in). It is in this and only this mode that you can enter text. To return to command mode, press the `Esc` key.

In insert mode, you can use the Backspace and `DEL` keys to delete text as you go along. To move around text, both in command mode and in insert mode, you use the arrow keys. In command mode, there are also other key combinations which we will look at later.

ex mode is accessed by pressing the `:` key in command mode. The same `:` will appear at the bottom of the screen, and the cursor will be positioned on it. Everything you type subsequently, followed by pressing `Enter`, will be considered by *Vi* to be an *ex* command. If you delete the command and the `:` you typed in, you will return to command mode and the cursor will go back to its original position.

To save changes to a file you type `:w` in command mode. If you want to save the contents of the buffer to another file, type `:w <file_name>`.

3.2.2. Handling buffers

As with *Emacs*, you can have several buffers displayed on the screen. To do this, use the `:split` command.

To move from one file to another, in a buffer, you type `:next` to move to the next file and `:prev` to move to the previous file. You can also use `:e <file_name>`, which allows you either to change to the desired file if this is already open, or to open another file. Here again completion is available.

To change buffers, type `Ctrl+w j` to go to the buffer below or `Ctrl+w k` to go to the buffer above. You can also use the up and down arrow keys instead of `j` or `k`. The `:close` command hides a buffer, the `:q` command closes it.

Watch out, *Vi* is finicky: if you try to hide or close a buffer without saving the changes, the command will not be carried out and you will get this message:

```
No write since last change (use! to override)
```

In this case, do as you are told and type `:q!` or `:close!`.

3.2.3. Editing text and move commands

Apart from the Backspace and DEL keys in edit mode, *Vi* has many other commands for deleting, copying, pasting, and replacing text – in command mode. Here, we will look at a few. All the commands shown here are in fact separated into two parts: the action to be performed and its effect. The action may be:

- **c**: to replace (*Change*); the editor deletes the text requested and goes back into insert mode after this command;
- **d**: to delete (*Delete*);
- **y**: to copy (*Yank*), we will look at this in the next section.
- **..**: repeats last action.

The effect defines which group of characters the command acts upon. These also effect commands entered as they are in command mode correspond to movements:

- **h, j, k, l**: one character left, down, up, right³ respectively;
- **e, b, w**: to the end (resp. to the beginning) of current word; to the beginning of the next word;
- **^, 0, \$**: to the first non blank character of current line, to beginning of current line, to the end of current line;
- **f<x>**: to next occurrence of character `<x>`; for example, `fe` moves the cursor to the next occurrence of the character `e`;
- **/<string>, ?<string>**: to the next occurrence of string or regexp `<string>`, and the same thing going backwards in the file; for example, `/foobar` moves the cursor until the next occurrence of the word `foobar`;
- **{, }**: to the beginning, to the end of current paragraph;
- **G, H**: to end of file, to beginning of screen.

Each of these effect characters or move commands can be preceded by a repetition number. For **G**, this references the line number in the file. On this basis, you can make all sorts of combinations.

Some examples:

- `6b`: moves 6 words backward;
- `c8fk`: delete all text until the eighth occurrence of the character `k` then goes into insert mode;
- `91G`: goes to line 91 of the file;
- `d3$`: deletes up to the end of the current line plus the next two lines.

It is true that these commands are not very intuitive, but as always the best method is practice. But you can see that the expression “move mountains with a few keys” is not much of an exaggeration :-)

3. A shortcut for `d1` (delete one character forward) is `x`; a shortcut for `dh` is `X`; `dd` deletes the current line.

3.2.4. Cut, copy, paste

Vi has a command that we have already seen for copying text: the **y** command. To cut text, simply use the **d** command. You have 27 memories for storing text: an anonymous memory and 26 memories named after the 26 lowercase letters of the alphabet.

To use the anonymous memory you enter the command as it is. So the command **y12w** copies to the anonymous memory the 12 words after the cursor⁴. Use **d12w** if you want to cut this area.

To use one of the 26 named memories, enter the sequence "<x>" before the command, where <x> gives the name of the memory. Thus, to copy the same 12 words into the memory **k**, you would write "**ky12w**", and "**kd12w**" to cut them.

To paste the contents of the anonymous memory, you use the commands **p** or **P** (for *Paste*), to insert text after or before the cursor respectively. To paste the contents of a named memory, use "<x>p" or "<x>P" in the same way (for example "**dp**" will paste the contents of memory **d** after the cursor).

Let us look at the example of figure 3-5.

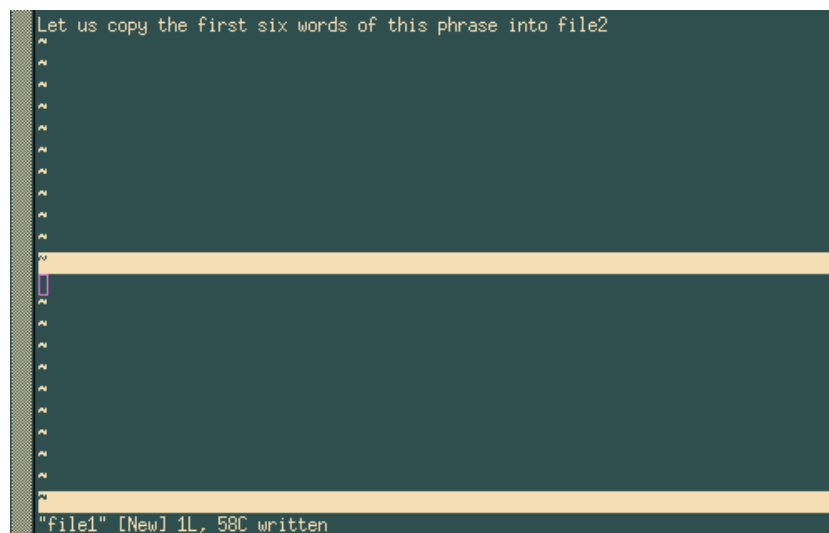


Figure 3-5. vim, before copying the text block

To carry out this action, we will:

- recopy the first 6 words of the sentence into memory **r** (for example): "**ry6w**"⁵;
- go into the buffer **file2**, which is located below: **Ctrl+w j**;
- paste the contents of memory **r** before the cursor: "**rp**".

We get the expected result, as shown in figure 3-6.

4. ... if the cursor is positioned at the beginning of the first word!

5. **y6w** literally means: "*Yank 6 words*".

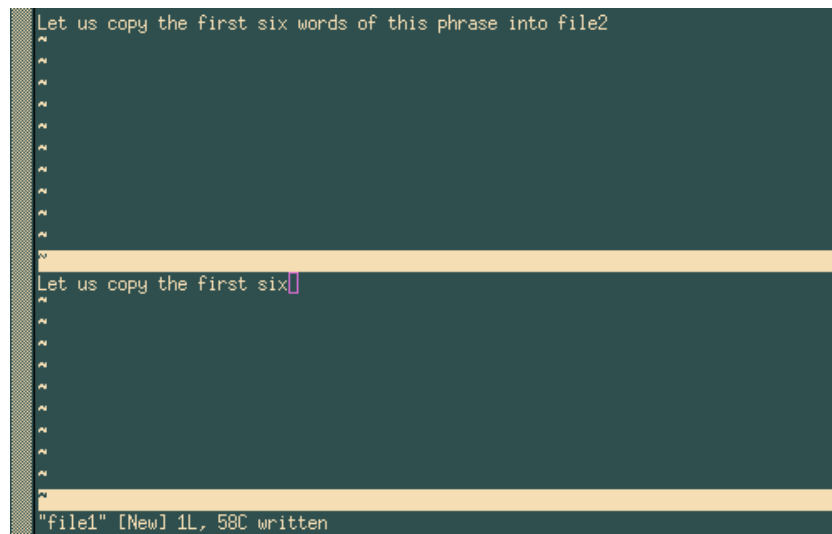


Figure 3-6. vim, after having copied the text block

Searching for text is very simple: in command mode, you simply type `/` followed by the string to search for, and then press the Enter key. For example, `/party` will search for the string `party` from the current cursor position. Pressing `n` takes you to the next occurrence, and if you reach the end of the file, the search will start again from the beginning. To search backwards, use `?` instead of `/`.

3.2.5. Quit Vi

To quit, the command is `:q` (in fact, this command closes the active buffer, as we have already seen, but if it is the only buffer present, you quit *Vi*). There is a shortcut: most of the time you edit only one file. So to quit, you will use:

- `:wq` to save changes and quit (a quicker solution is `ZZ`), or
- `:q!` to quit without saving.

You will have noticed that if you have several buffers, `:wq` will write the active buffer then close it.

3.3. A last word...

Of course, we have said much more here than was necessary (after all, the first aim was to edit a text file), but it is also to show you some of the possibilities of each of these editors. There is a great deal more to be said about them, as witnessed by the number of books dedicated to both of these editors.

Take the time to absorb all this information, opt for one of them, or learn only what you think necessary. But at least you know that when you want to go further, you can :-)

Chapter 4. Command Line Utilities

The purpose of this chapter is to introduce a small number of command line tools which may prove useful for everyday use. Of course, you may skip this chapter if you only intend to use a graphical environment, but a quick glance may change your opinion :-)

There is not really any organization in this chapter. Utilities are listed as they come, from the most commonly used to the most obscure. Each command will be illustrated by an example, but this chapter is meant as an exercise, in order for you to grasp their function and use.

4.1. `grep`: Locate Strings in Files

Okay, the name is not very intuitive, neither is its acronym (“General Regular Expression Parser”), but its use is simple: looking for a pattern given as an argument in one or more files. Its syntax is:

```
grep [options] <pattern> [one or more file(s)]
```

If several files are mentioned, their name will precede each matching line displayed in the result. Use the `-h` option to prevent the display of these names; use the `-l` option to get nothing but the matching filenames. The pattern is a regular expression, even though most of the time it consists of a simple word. The most frequently used options are the following:

- `-i`: make a case insensitive search. (i.e. ignore differences between lower and uppercase);
- `-v`: invert search: display lines which do **not** match the pattern;
- `-n`: display the line number for each line found;
- `-w`: tells *grep* that the pattern should match a whole word.

Here’s an example of how to use it:

```
$ cat my_father
Hello dad
Hi daddy
So long dad

# Search for the string "hi", no matter the case
$ grep -i hi my_father
Hi daddy

# Search for "dad" as a whole word, and print the
# line number in front of each match
$ grep -nw dad my_father
1:Hello dad
3:So long dad

# We want all lines not beginning with "H" to match
$ grep -v "^H" my_father
So long dad
$
```

In case you want to use *grep* in a pipe, you don’t have to specify the filename as, by default, it takes its input from the *standard input*. Similarly, by default, it prints the results on the *standard output*, so you can pipe the output of a *grep* to yet another program without fear. Example:

```
$ cat /usr/share/doc/HOWTO/Parallel-Processing-HOWTO | \
grep -n thread | less
```

4.2. find: Find Files According to Certain Criteria

`find` is a long-standing *UNIX* utility. Its role is to recursively scan one or more directories and find files which match a certain set of criteria in these directories. Even though it is very useful, its syntax is truly obscure, and using it requires a little work. The general syntax is:

```
find [options] [directories] [criterion] [action]
```

If you do not specify any directory, `find` will search the current directory. If you do not specify a criterion, this is equivalent to “true”, thus all files will be found. The options, criteria and actions are so numerous that we will only mention a few of each here. Let’s start with options:

- `-xdev`: do not search on directories located on other filesystems;
- `-mindepth <n>`: descend at least `<n>` levels below the specified directory before searching for files;
- `-maxdepth <n>`: search for files which are located at most `n` levels below the specified directory;
- `-follow`: follow symbolic links if they link to directories. By default, `find` does not follow them;
- `-daystart`: when using tests related to time (see below), take the beginning of current day as a timestamp instead of the default (24 hours before current time).

A criterion can be one or more of several *atomic* tests; some useful tests are:

- `-type <type>`: search for a given type of file; `<type>` can be one of: `f` (regular file), `d` (directory), `l` (symbolic link), `s` (socket), `b` (block mode file), `c` (character mode file) or `p` (named pipe);
- `-name <pattern>`: find files whose names match the given `<pattern>`. With this option, `<pattern>` is treated as a *shell globbing* pattern (see chapter *Shell Globbing Patterns*, page 12);
- `-iname <pattern>`: like `-name`, but ignore case;
- `-atime <n>`, `-amin <n>`: find files which have last been accessed `<n>` days ago (`-atime`) or `<n>` minutes ago (`-amin`). You can also specify `+<n>` or `-<n>`, in which case the search will be done for files accessed respectively at most or at least `<n>` days/minutes ago;
- `-anewer <file>`: find files which have been accessed more recently than file `<file>`;
- `-ctime <n>`, `-cmin <n>`, `-cnewer <file>`: same as for `-atime`, `-amin` and `-anewer`, but applies to the last time when the contents of the file have been modified;
- `-regex <pattern>`: same as `-name`, but pattern is treated as a *regular expression*;
- `-iregex <pattern>`: same as `-regex`, but ignore case.

There are many other tests, refer to `find(1)` for more details. To combine tests, you can use one of:

- `<c1> -a <c2>`: true if both `<c1>` and `<c2>` are true; `-a` is implicit, therefore you can type `<c1> <c2> <c3> ...` if you want all tests `<c1>`, `<c2>`, ... to match;
- `<c1> -o <c2>`: true if either `<c1>` or `<c2>` are true, or both. Note that `-o` has a lower *precedence* than `-a`, therefore if you want, say, to match files which match criteria `<c1>` or `<c2>` and match criterion `<c3>`, you will have to use parentheses and write `(<c1> -o <c2>) -a <c3>`. You must *escape* (deactivate) parentheses, as otherwise they will be interpreted by the *shell*!
- `-not <c1>`: inverts test `<c1>`, therefore `-not <c1>` is true if `<c1>` is false.

Finally, you can specify an action for each file found. The most frequently used are:

- `-print`: just prints the name of each file on standard output. This is the default action;
- `-ls`: prints on the standard output the equivalent of `ls -l` for each file found;
- `-exec <command>`: execute command `<command>` on each file found. The command line `<command>` must end with a `;`, which you must escape so that the shell does not interpret it; the file position is marked with `{}`. See the usage examples to figure this out;
- `-ok <command>`: same as `-exec` but ask confirmation for each command.

Still here? OK, now let’s practice a little, as it’s still the best way to figure out this monster. Let’s say you want to find all directories in the `/usr/share` directory. Then you will type:

```
find /usr/share -type d
```

Suppose you have an HTTP server, all your HTML files are in `/var/www/html`, which is also your current directory. You want to find all files which contents have not been modified for a month. As you got pages from several writers, some files have the `html` extension and some have the `htm` extension. You want to link these files in directory `/var/www/obsolete`. You will then type¹:

```
find \( -name "*.htm" -o -name "*.html" \) -a -ctime -30 \
-exec ln {} /var/www/obsolete \;
```

Okay, this one is a little complex and requires a little explanation. The criterion is this:

```
\( -name "*.htm" -o -name "*.html" \) -a -ctime -30
```

which does what we want: it finds all files which names end either in `.htm` or `.html` “`\(-name "*.htm" -o -name "*.html" \)`”, and `(-a)` which have not been modified in the last 30 days, which is roughly a month (`-ctime -30`). Note the parentheses: they are necessary here, because `-a` has a higher precedence. If there weren’t any, all files ending with `.htm` would have been found, plus all files ending with `.html` and which haven’t been modified for a month, which is not what we want. Also note that parentheses are escaped from the shell: if we had put `(..)` instead of `\(.. \)`, the shell would have interpreted them and tried to execute `-name "*.htm" -o -name "*.html"` in a sub-shell... Another solution would have been to put parentheses between double quotes or single quotes, but a backslash here is preferable as we only have to isolate one character.

And finally, there is the command to be executed for each file:

```
-exec ln {} /home/httpd/obsolete \;
```

Here too, you have to escape the `;` from the *shell*, as otherwise the shell interprets it as a command separator. If you don’t do so, `find` will complain that `-exec` is missing an argument.

A last example: you have a huge directory (`/shared/images`) holding all kinds of images. Regularly, you use the `touch` command to update the times of a file named `stamp` in this directory, so that you have a time reference. You want to find all **JPEG** images in it which are newer than the `stamp` file, and as you got images from various sources, these files have extensions `jpg`, `jpeg`, `JPG` or `JPEG`. You also want to avoid searching in the old directory. You want this file list to be mailed to you, and your username is `peter`:

```
find /shared/images -cnewer \
    /shared/images/stamp \
    -a -iregex ".*\.(jpe?g)" \
    -a -not -regex ".*old/.*" \
    | mail peter -s "New images"
```

And here you are! Of course, this command is not very useful if you have to type it each time, and you would like it to be executed regularly... You can do so:

4.3. crontab: reporting or editing your crontab file

`crontab` is a command which allows you to execute commands at regular time intervals, with the added bonus that you don’t have to be logged in and that the output report is mailed to you. You can specify the intervals in minutes, hours, days, and even months. Depending on the options, `crontab` will act differently:

- `-l`: Print your current crontab file.
- `-e`: Edit your crontab file.
- `-r`: Remove your current crontab file.

1. Note that this example requires that `/var/www` and `/var/www/obsolete` be on the same filesystem!

- `-u <user>`: Apply one of the above options for user `<user>`. Only root can do that.

Let's start by editing a crontab. If you type `crontab -e`, you will be in front of your favorite text editor if you have set the `EDITOR` or `VISUAL` environment variable, otherwise `Vim` will be used. A line in a crontab file is made of six fields. The first five fields are time intervals for minutes, hours, days in the month, months and days in the week. The sixth field is the command to be executed. Lines beginning with a `#` are considered to be comments and will be ignored by `crond` (the program which is responsible for executing crontab files). Here is an example of crontab:



in order to print this out in a readable font, we had to break up long lines. Therefore, some chunks must be typed on a single line. When the `\` character ends a line, this means this line has to be continued. This convention works in Makefile files and in the *shell*, as well as in other contexts.

```
# If you don't want to be sent mail, just comment
# out the following line
#MAILTO=""
#
# Report every 2 days about new images at 2 pm,
# from the example above - after that, "retouch"
# the "stamp" file. The "%" is treated as a
# newline, this allows you to put several
# commands in a same line.
0 14 */2 * * find /shared/images          \
-cnewer /shared/images/stamp              \
-a -iregex ".*\.jpe?g"                    \
-a -not -regex                             \
  ".*old/.*"touch /shared/images/stamp
#
# Every Christmas, play a melody :)
0 0 25 12 * mpg123 $HOME/sounds/merryxmas.mp3
#
# Every Tuesday at 5pm, print the shopping list...
0 17 * * 2 lpr $HOME/shopping-list.txt
```

There are several ways to specify intervals other than the ones shown in this example. For example, you can specify a set of *discrete values* separated by commas (1,14,23) or a range (1-15), or even combine both of them (1-10,12-20), optionally with a step (1-12,20-27/2). Now it's up to you to find useful commands to put in it!

4.4. at: schedule a command, but only once

You may also want to launch a command at a given day, but not regularly. For example, you want to be reminded of an appointment, today at 6pm. You run `X`, and you'd like to be notified at 5:30pm, for example, that you must go. `at` is what you want here:

```
$ at 5:30pm
# You're now in front of the "at" prompt
at> xmessage "Time to go now! Appointment at 6pm"
# Type C-d to exit
at> <EOT>
$
```

You can specify the time in different manners:

- `now +<interval>`: Means, well, now, plus an interval (optionally. No interval specified means just now). The syntax for the interval is `<n>` (minutes | hours | days | weeks | months). For example, you can specify `now + 1 hour` (an hour from now), `now + 3 days` (three days from now) and so on.
- `<time> <day>`: Fully specify the date. The `<time>` parameter is mandatory. `at` is very liberal in what it accepts: you can for example type 0100, 04:20, 2am, 0530pm, 1800, or one of three special values: noon, teatime (4pm) or midnight. The `<day>` parameter is optional. You can specify it in different manners as well: 12/20/2001 for example, which stands for December 20th, 2001, or, the European way, 20.12.2001.

You may omit the year, but then only the European notation is accepted: 20.12. You can also specify the month in full letters: Dec 20 or 20 Dec are both valid.

at also accepts different options:

- -l: Prints the list of currently queued jobs; the first field is the job number. This is equivalent to the atq command.
- -d <n>: Remove job number <n> from the queue. You can obtain job numbers from atq. This is equivalent to atrm <n>.

As usual, see the at(1) manpage for more options.

4.5. tar: Tape ARchiver

Although we have already seen a use for tar in the chapter “Building and installing free software”, page 81, we haven’t explained how it works. This is what this section is here for. As for find, tar is a long standing *UNIX* utility, and as such its syntax is a bit special. The syntax is:

```
tar [options] [files...]
```

Now, here is a list of options. Note that all of them have an equivalent long option, but you will have to refer to the manual page for this as they won’t be listed here. And of course, not all options will be listed either :-)



the initial dash (-) of short options is now deprecated with tar, except after a long option.

- c: this option is used in order to create new archives;
- x: this option is used in order to extract files from an existing archive;
- t: list files from an existing archive;
- v: this will simply list the files are they are added to an archive or extracted from an archive, or, in conjunction with the t option (see above), it outputs a long listing of files instead of a short one;
- f <file>: create archive with name <file>, extract from archive <file> or list files from archive <file>. If this parameter is omitted, the default file will be /dev/rmt0, which is generally the special file associated to a *streamer*. If the file parameter is - (a dash), the input or output (depending on whether you create an archive or extract from one) will be associated to the standard input or standard output;
- z: tells tar that the archive to create should be compressed with gzip, or that the archive to extract from is compressed with gzip;
- j: same as z, but the program used for compression is bzip2;
- p: when extracting files from an archive, preserve all file attributes, including ownership, last access time and so on. Very useful for filesystem dumps;
- r: append the list of files given on the command line to an existing archive. Note that the archive to which you want to append files should **not** be compressed!
- A: append archives given on the command line to the one submitted with the f option. Similar to r, the archives should not be compressed in order for this to work;

There are many, many, many other options, so you may want to refer to the tar(1) manual page for a whole list. See, for example, the d option. Now, on for a little practice. Say you want to create an archive of all images in /shared/images, compressed with bzip2, named images.tar.bz2 and located in your home directory. You will then type:

```
#
# Note: you must be in the directory from which
# you want to archive files!
#
$ cd /shared
$ tar cjf ~/images.tar.bz2 images/
```

As you can see, we used three options here: `c` told `tar` we wanted to create an archive, `j` to compress it with `bzip2`, and `f ~/images.tar.bz2` that the archive was to be created in our home directory, and its name will be `images.tar.bz2`. We may want to check if the archive is valid now. We can just check this out by listing its files:

```
#
# Get back to our home directory
#
$ cd
$ tar tjvf images.tar.bz2
```

Here, we told `tar` to list (`t`) files from archive `images.tar.bz2` (`f images.tar.bz2`), warned that this archive was compressed with `bzip2` (`j`), and that we wanted a long listing (`v`). Now, say you have erased the `images` directory. Fortunately, your archive is intact, and you now want to extract it back to its original place, in `/shared`. But as you don't want to break your `find` command for new `images`, you need to preserve all file attributes:

```
#
# cd to the directory where you want to extract
#
$ cd /shared
$ tar jxpf ~/images.tar.bz2
```

And here you are!

Now, let's say you want to extract the directory `images/cars` from the archive, and nothing else. Then you can type this:

```
$ tar jxf ~/images.tar.bz2 images/cars
```

In case you would worry about this, don't. If you try to back up special files, `tar` will take them as what they are, special files, and will not dump their contents. So yes, you can safely put `/dev/mem` in an archive :-). Oh, and it also deals correctly with links, so do not worry about this either. For symbolic links, also look at the `h` option in the manpage.

4.6. bzip2 and gzip: Data Compression Programs

You can see that we already have talked of these two programs when dealing with `tar`. Unlike `winzip` under *Windows*, archiving and compressing are done using two separate utilities – `tar` for archiving, and the two programs which we will now introduce for compressing data: `bzip2` and `gzip`. You might also use another compressing tool, programs like `zip`, `arj` or `rar` also exist for *GNU/Linux* (but they are rarely used).

At first, `bzip2` was written as a replacement for `gzip`. Its compression ratios are generally better, but on the other hand, it is more memory-greedy. The reason why `gzip` is still here is that it is still more widespread than `bzip2`.

Both commands have a similar syntax:

```
gzip [options] [file(s)]
```

If no filename is given, both `gzip` and `bzip2` will wait for data from the standard input and send the result to the standard output. Therefore, you can use both programs in pipes. Both programs also have a set of common options:

- `-1, ..., -9`: set the compression ratio. The higher the number, the better the compression, but better also means slower: "There's no such thing as a free lunch";
- `-d`: uncompress file(s). This is equivalent to using `gunzip` or `bunzip2`;

- `-c`: dump the result of compression/decompression of files given as parameters to the standard output.



By default, both `gzip` and `bzip2` erase the file(s) that they have compressed (or uncompressed) if you don't use the `-c` option. You can avoid it with `bzip2` by using the `-k` option, but `gzip` has no such option!

Now some examples. Let's say you want to compress all files ending with `.txt` in the current directory using `bzip2`, you will then use:

```
$ bzip2 -9 *.txt
```

Let's say you want to share your image archives with someone, but he doesn't have `bzip2`, only `gzip`. You don't need to uncompress the archive and re-compress it, you can just uncompress to the standard output, use a pipe, compress from standard input and redirect the output to the new archive:

```
bzip2 -dc images.tar.bz2 | gzip -9 >images.tar.gz
```

And here you are. You could have typed `bzcat` instead of `bzip2 -dc`. There is an equivalent for `gzip` but its name is `zcat`, not **`gzcat`**. You also have `bzless` (resp. `zless`) if you want to view compressed files right away, without having to uncompress them first. As an exercise, try and find the command you would have to type in order to view compressed files without uncompressing them, and without using `bzless` or `zless` :-)

4.7. Many, many more...

There are so many commands that a comprehensive book about them would be the size of an encyclopedia. This chapter hasn't even covered a tenth of the subject, yet you can do much with what you learned here. If you wish, you may read some manual pages: `sort(1)`, `sed(1)` and `zip(1)` (yes, that's what you think: you can extract or make `.zip` archives with *GNU/Linux*), `convert(1)`, and so on. The best way to get accustomed to these tools is to practice and experiment with them, and you will probably find a lot of uses for them, even quite unexpected ones. Have fun!

Chapter 5. Process control

5.1. More about processes

In *Processes*, page 4, we mentioned that it was possible to monitor processes; that is what we will cover in this chapter. To understand the operations we are going to perform here, it is helpful to know a bit more about them.

5.1.1. The process tree

As with files, all processes that run on a *GNU/Linux* system are organized in the form of a tree. The root of this tree is *init*. Each process has a number (its PID, *Process ID*), together with the number of its parent process (PPID, *Parent Process ID*). The PID of *init* is 1, and so is its PPID: *init* is its own father.

5.1.2. Signals

Every process in *UNIX* can react to signals sent to it. There are 64 different signals which are identified either by their number (starting from 1) or by their symbolic names. The 32 “higher” signals (33 to 64) are real-time signals, and are out of the scope of this chapter. For each of these signals, the process can define its own behavior, except for two signals: signal number 9 (KILL), and signal number 19 (STOP).

Signal 9 kills a process irrevocably, without giving it the time to terminate properly. This is the signal you send to a process which is stuck or exhibits other problems. A full list of signals is available using the command `kill -l`.

5.2. Information on processes: `ps` and `pstree`

These two commands display a list of processes currently running on the system, according to criteria set by you.

5.2.1. `ps`

Sending this command without an argument will show only processes initiated by you and attached to the terminal you are using:

```
$ ps
  PID TTY          TIME CMD
 5162 ttya1      00:00:00 zsh
 7452 ttya1      00:00:00 ps
```

As with many *UNIX* utilities, `ps` has a handful of options, we will look at the most common here:

- `a`: also displays processes started by other users;
- `x`: also displays processes with no control terminal or with a control terminal different to the one you are using;
- `u`: displays for each process the name of the user who started it and the time it was started.

There are many other options. Refer to the `ps(1)` manual page for more information.

The output of this command is divided into different fields: the one that will interest you most is the field PID, which contains the process identifier. The field CMD contains the name of the command executed. A very common way of invoking `ps` is as follows:

```
$ ps ax | less
```

This gets you a list of all processes currently running, so that you can identify one or more processes which are causing problems and subsequently kill them.

5.2.2. pstree

The command `ps` displays the processes in the form of a tree structure. One advantage is that you can immediately see which is the parent process of what: when you want to kill a whole series of processes and if they are all parents and children, you can simply kill the parent. You want to use the option `-p`, which displays the PID of each process, and the option `-u` which displays the name of the user who started off the process. As the tree structure is generally long, you want to invoke `ps` in the following way:

```
$ ps -p | less
```

This gives you an overview of the whole process tree structure.

5.3. Sending signals to processes: kill, killall and top

5.3.1. kill, killall

These two commands are used to send signals to processes. The `kill` command requires a process number as an argument, while `killall` requires a command name.

Both of these commands can optionally receive a signal number as an argument. By default, they both send the signal 15 (TERM) to the relevant process(es). For example, if you want to kill the process with PID 785, you enter the command:

```
$ kill 785
```

If you want to send it signal 9, you enter:

```
$ kill -9 785
```

Suppose that you want to kill a process for which you know the command name. Instead of finding the process number using `ps`, you can kill the process directly:

```
$ killall -9 netscape
```

Whatever happens, you will only kill your own processes (unless you are root), so don't worry about the "neighbor's" processes with the same name, they will not be affected.

5.3.2. top

`top` is a program all in one: it simultaneously fulfills the functions of `ps` and `kill`. It is a console mode program, so you start it from a terminal, as shown in figure 5-1.

```

1:22pm up 23:26, 5 users, load average: 0.01, 0.02, 0.00
50 processes: 47 sleeping, 2 running, 1 zombie, 0 stopped
CPU states: 1.5% user, 1.1% system, 0.0% nice, 97.2% idle
Mem: 128000K av, 124936K used, 3072K free, 40000K shrd, 1912K buff
Swap: 136512K av, 1772K used, 134740K free, 24700K cached

```

PID	USER	PRI	NI	SIZE	RSS	SHARE	STAT	LIB	%CPU	%MEM	TIME	COMMAND
9837	root	15	0	60716	59M	1940	R	0	1.5	47.4	3:10	%
10164	fg	9	0	1060	1060	860	R	0	0.5	0.8	0:00	top
9855	fg	1	0	2192	2192	1564	S	0	0.1	1.7	0:00	xterm
10066	fg	1	0	20592	20M	8312	S	0	0.1	16.0	0:09	ld-linux.so.
10168	fg	19	0	440	440	360	S	0	0.1	0.3	0:00	sleep
1	root	0	0	168	168	92	S	0	0.0	0.1	0:04	init
2	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kflushd
3	root	0	0	0	0	0	SW	0	0.0	0.0	0:01	kupdate
4	root	0	0	0	0	0	SW	0	0.0	0.0	0:00	kpiod
5	root	0	0	0	0	0	SW	0	0.0	0.0	0:03	kswapd
6	root	-20	-20	0	0	0	SW<	0	0.0	0.0	0:00	mdrecoveryd
133	root	0	0	72	60	0	S	0	0.0	0.0	0:00	apmd
268	bin	0	0	76	52	0	S	0	0.0	0.0	0:00	portmap
320	root	0	0	292	244	168	S	0	0.0	0.1	0:00	syslogd
330	root	0	0	496	164	120	S	0	0.0	0.1	0:00	klogd
345	root	0	0	184	176	80	S	0	0.0	0.1	0:00	crond
360	root	0	0	224	216	124	S	0	0.0	0.1	0:00	inetd

Figure 5-1. Example of execution of top

The program is entirely keyboard controlled. You can access help by pressing **h**. Here are some of the commands you can use.

- **k**: this command is used to send a signal to a process. top will then ask you for the process PID followed by the number of the signal to be sent (15 by default);
- **M**: this command is used to sort processes by the amount of memory they take up (field %MEM);
- **P**: this command is used to sort processes by the CPU time they take up (field %CPU; this is the default sort method);
- **u**: this command is used to display a given user's processes, top will ask you which one. You need to enter the user's **name**, not his UID. If you do not enter any name, all processes will be displayed;
- **i**: this command acts as a toggle; by default, all processes, even sleeping ones, are displayed; this command ensures that only processes currently running are displayed (processes whose STAT field states R, *Running*) and not the others. Using this command again takes you back to the previous situation.

II. Linux in Depth

Chapter 6. File Tree Organization

Nowadays, a *UNIX* system is big, very big. This is especially true with *GNU/Linux*: the amount of software available would make it an unmanageable system if there weren't any guidelines for the location of files in the tree.

The acknowledged standard in this respect is the FHS (*Filesystem Hierarchy Standard*), which is at version 2.2 at the time of writing this manual. The document which describes the standard is available on the Internet in different formats on The Pathname web site (<http://www.pathname.com/fhs/>). This chapter gives only a brief summary, but it should be enough to teach you in what directory to look for (or place) a given file.

6.1. Shareable/Unshareable, Static/Variable Data

Data on a *UNIX* system can be classified according to these two criteria. Their meaning is the following: shareable data can be common to several computers in a network, while unshareable can not. Static data must not be modified in normal use, while variable data can. As we explore the tree structure, we will classify the different directories into each of these categories.

Note that these classifications are only recommended. It is not mandatory follow them, but adopting these guidelines will greatly help you manage your system. Also, keep in mind that the static/variable distinction only applies to system usage, its configuration. If you install a program, you will obviously have to modify "normally" static directories, i.e.: `/usr`.

6.2. The root Directory: `/`

The root directory contains the whole system hierarchy. It cannot be classified since its subdirectories may or may not be static or shareable. Here is a list of the main directories and subdirectories, with their classifications:

- `/bin`: essential binary files. This directory contains the basic commands which will be used by all users and are necessary to the operation of the system: `ls`, `cp`, `login`, etc. Static, unshareable;
- `/boot`: contains the files required by the *GNU/Linux* bootloader (*grub* or *LILO* for **Intel**). This may or may not contain the kernel: if it is not here, it must be located in the root directory. Static, unshareable;
- `/dev`: system device files (*dev* for *DEVICES*). Static, unshareable;
- `/etc`: this directory contains all configuration files specific to the computer. Static, unshareable;
- `/home`: contains all the personal directories of the system's users. This directory may or may not be shareable (some large networks make it shareable by NFS). Variable, shareable;
- `/lib`: this directory contains libraries which are essential to the system; it also hosts kernel modules in `/lib/modules`. All libraries required by the binaries in the `/bin` and `/sbin` directories must be located here, together with the `ld.so` linker. Static, unshareable;
- `/mnt`: directory containing the mount points for temporary file systems. Variable, unshareable;
- `/opt`: holds packages not required for system operation. It is recommended to place static files (binaries, libraries, manual pages, etc.) for such packages in `/opt/package_name` and their specific configuration files in `/etc/opt`;
- `/root`: home directory for root. Variable, unshareable;
- `/usr`: see next section. Static, shareable;
- `/sbin`: contains system binaries essential to the system start-up, operable only by root. A normal user can also run them but will not get very far. Static, unshareable;
- `/tmp`: directory intended to contain temporary files which certain programs may create. Variable, unshareable;
- `/var`: location for data which may be modified in real time by programs (i.e.: e-mail servers, audit programs, print servers, etc.). All of the `/var` directory is variable, but its different subdirectories may be shareable or unshareable.

6.3. /usr: The Big One

The /usr directory is the main application-storage directory. All binary files in this directory must not be required for the system start-up or maintenance, since the /usr hierarchy is very often located on a separate filesystem. Given its often large size, /usr has its own hierarchy of subdirectories. We will mention just a few:

- /usr/X11R6: the whole *X Window System* hierarchy. All binaries required for the operation of *X* (including the *X* servers) and all necessary libraries must be located here. The /usr/X11R6/lib/X11 directory contains all aspects of *X*'s configuration which do not vary from one computer to another. Specific configurations for each computer should go in /etc/X11;
- /usr/bin: this one holds the large majority of the system's binaries. **Any** binary program which is not necessary to the maintenance of the system and is not a system administration program must be located in this directory, apart from programs you install yourself, which must be located in /usr/local;
- /usr/lib: it contains all the necessary libraries to run programs located in /usr/bin and /usr/sbin. There is also a /usr/lib/X11 symbolic link pointing to the directory which holds the *X Window System* libraries, /usr/X11R6/lib (if *X* is installed, of course);
- /usr/local: this is where you should install your personal applications. The installation program will create the necessary hierarchy: lib/, bin/, etc.;
- /usr/share: this directory contains all architecture-independent data required by applications in /usr. Among other things, you will find zone and location information (zoneinfo and locale).

Let's also mention the /usr/share/doc and /usr/share/man directories, which respectively contain application documentation and the system's manual pages.

6.4. /var: Modifiable Data During Use

The /var directory contains all operating data for programs running on the system. Unlike the working data in /tmp, this data must be kept intact in the event of a reboot. There are many subdirectories, and some are very useful:

- /var/log: contains the system log files;
- /var/spool: holds the system daemons' working files. For example, /var/spool/lpd contains the print server's working files while /var/spool/mail holds the e-mail server's working files (i.e.: all mail arriving on and leaving your system).
- /var/run: used to keep track of all processes utilized by the system, enabling you to act on them in the event of a system change *runlevel* (see chapter "The Start-Up Files: init sysv", page 57).

6.5. /etc: Configuration Files

/etc is one of *UNIX* systems' most essential directories. It holds all the basic system configuration files. **Never** delete it to save space! Likewise, if you want to extend your tree structure over several partitions, remember that /etc must not be put on a separate partition: it is needed for system initialization.

Here are some important files:

- passwd and shadow: these two are text files which contain all system users and their encrypted passwords. shadow is only there if you use shadow passwords, which is the default installation option;
- inittab: this is the configuration file for init, which plays a fundamental role when starting up the system, as we will see later on;
- services: this file contains a list of existing network services;
- profile: this is the *shell* configuration file, although certain *shells* use other files. For example, *bash* uses bashrc;
- crontab: cron's configuration file, the program responsible for periodic execution of commands.

Also, certain subdirectories exist for programs which require a large number of configuration files. This applies to the *X Window System*, for example, which stores all its files in the /etc/X11 directory.

Chapter 7. Filesystems and Mount Points

The best way to understand “how it works” is to look at a practical case, which is what we are going to do here. Suppose you just bought a brand new hard disk with no partitions on it. Your **Mandrake Linux** partition is full to bursting, and rather than starting again from scratch, you decide to move a whole section of the tree structure to your new hard disk. As this new disk is very big, you decide to move your biggest directory on it: `/usr`. But first, a bit of theory.

7.1. Principles

As we already mentioned in the *Installation Guide*, every hard disk is divided into several partitions, and each of these contains a filesystem. While *Windows* assigns a letter to each of these filesystems (actually, only to those it recognizes), *GNU/Linux* has a unique tree structure of files, and each filesystem is *mounted* at one location in the tree structure.

Just as *Windows* needs a “C: drive”, *GNU/Linux* must be able to mount the root of its file tree (`/`) somewhere, in fact on a partition which contains the *root filesystem*. Once the root is mounted, you can mount other filesystems in the tree structure, at different *mount points* in the tree structure. Any directory below the root structure can act as a mount point. Note that you can also mount the same filesystem several times.

This allows great configuration flexibility. In the case of a web server, for example, it is common to dedicate a whole partition to the directory which hosts the web-server data. Generally, the directory which contains it is `/var/www`. Hence, it acts as the mounting point for the partition. You can see in figure 7-1 and figure 7-2 the system’s situation before and after mounting the filesystem.

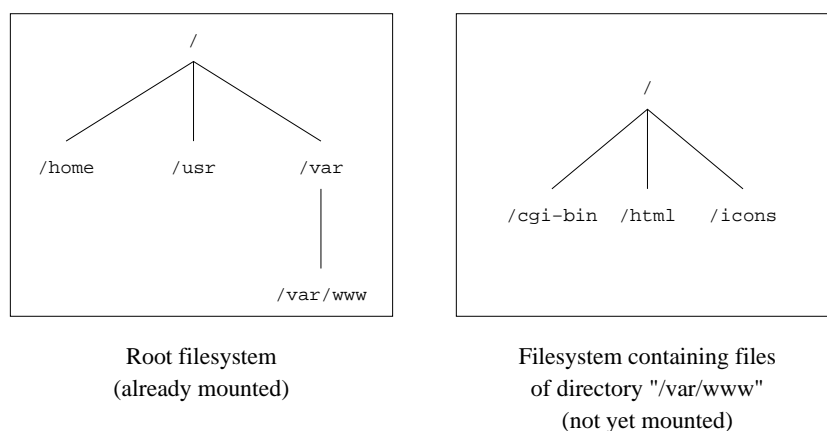


Figure 7-1. A Not Yet Mounted Filesystem

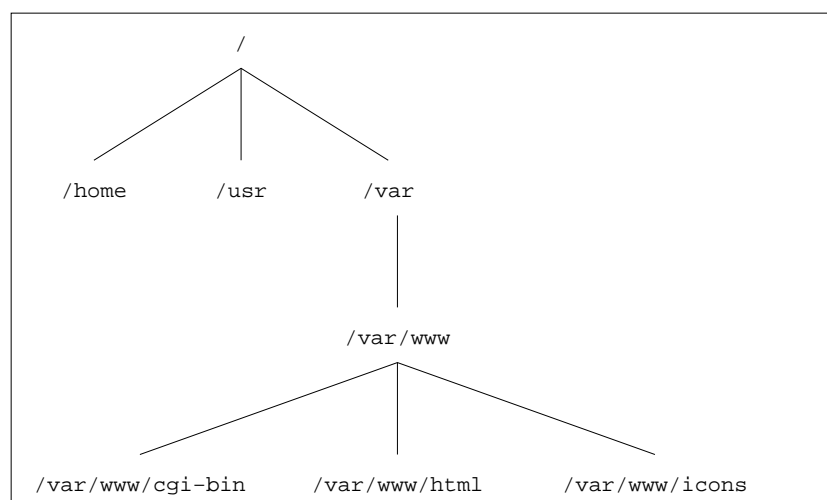


Figure 7-2. Filesystem Is Now Mounted

As you can imagine, this offers a number of advantages: the tree structure will always be the same, whether it extends over a single filesystem or several dozens¹. It is always possible to physically move a key part of the tree structure to another partition when space becomes scarce, which is what we are going to do here.

There are two things you need to know about mount points:

1. the directory which acts as a mount point must exist;
2. and this directory **should preferably be empty**: if a directory chosen as a mount point already contains files and subdirectories, these will simply be “hidden” by the newly mounted filesystem, but they will not be accessible anymore until you free the mount point.

7.2. Partitioning a Hard Disk, Formatting a Partition

Regarding the principles referred to above and as far as we are concerned in this section, there are two things to note: a hard disk is divided into partitions and each of these partitions host a filesystem. Your brand new hard disk has neither, so that is where you have to start, beginning with the partitioning. In order to do that, you must be root.

First, you have to know the “name” of your hard disk (i.e.: what file designates it). Suppose you set it up as a slave on your primary IDE interface, it will then be `/dev/hdb`.² Please refer to the *User Guide's Managing Your Partitions* section, which explains how to partition a disk. Note that *DiskDrake* will also create the filesystems for you.

7.3. The mount And umount Commands

Now that the filesystem has been created, you can mount the partition. Initially, it will be empty. The command to mount filesystems is the `mount` command, and its syntax is as follows:

```
mount [options] <-t type> [-o mount options] <device> <mounting point>
```

In this case, we want to temporarily mount our partition on `/mnt` (or any other mount point you have chosen – remember that it must exist); the command for mounting our newly created partition is:

```
$ mount -t ext2 /dev/hdb1 /mnt
```

The `-t` option is used to specify what type of file system the partition is supposed to host. Among the filesystems you will encounter most frequently are `ext2fs` (the *GNU/Linux* file system), `VFAT` (for all *DOS/Windows* partitions: `FAT 12`, `16` or `32`) and `iso9660` (`CD-ROM` filesystem). If you do not specify any type, `mount` will try and guess which filesystem is hosted by the partition by reading the superblock. It rarely fails at doing so.

The `-o` option is used to specify one or more mounting options. These options depend on the filesystem used. Refer to the `mount(8)` man page for more details.

Now that you mounted your new partition, you need to copy the entire `/usr` directory into it:

```
$ (cd /usr && tar cf - .) | (cd /mnt && tar xpvf -)
```

Now that the files are copied, we can unmount our partition. To do this, use the `umount` command. The syntax is simple:

```
umount <mount point|device>
```

So, to unmount our new partition, we can type:

```
$ umount /mnt
```

1. *GNU/Linux* can manage up to 64 mounted filesystems simultaneously.
2. How to find the name of a disk is explained in the *Installation Guide*.

or else:

```
$ umount /dev/hdb1
```

Since this partition is going to “become” our /usr directory, we need to tell this to the system. To do this, we edit:

7.4. The /etc/fstab File

The /etc/fstab file makes it possible to automate the mounting of certain filesystems, especially at system start-up. It contains a series of lines describing the filesystems, their mount points and other options. Here is an example of an /etc/fstab file:

```
/dev/hda1  /      ext2    defaults    1 1
/dev/hda5  /home  ext2    defaults    1 2
/dev/hda6  swap   swap    defaults    0 0
/dev/fd0   /mnt/floppy auto    sync,user,noauto,nosuid,nodev,unhide 0 0
/dev/cdrom /mnt/cdrom auto    user,noauto,nosuid,exec,nodev,ro 0 0
none       /proc  proc    defaults    0 0
none       /dev/pts devpts  mode=0622    0 0
```

A line contains, in order:

- the device hosting the filesystem;
- the mount point;
- the type of filesystem;
- the mounting options;
- the dump utility backup *flag*;
- fsck's (*FileSystem Check*) checking order.

There is **always** an entry for the root filesystem. The *swap* partitions are special since they are not visible in the tree structure, and the mount point field for those partitions contain the *swap* keyword. As for the /proc filesystem, we will get back to it in greater detail in “*The /proc Filesystem*”, page 53. An other special filesystem is /dev/pts.

Let's get back to the subject. You moved the whole /usr hierarchy to /dev/hdb1 and so you want this partition to be mounted as /usr at boot time. In that case you need to add an entry to the file:

```
/dev/hdb1      /usr      ext2    defaults    1 2
```

Now the partition will be mounted at each boot. It will also be checked if necessary.

There are two special options: *noauto* and *user*. The *noauto* option specifies that the filesystem should not be mounted at start-up but is to be mounted only when you tell it to. The *user* option specifies that any user can mount and unmount the filesystem. These two options are typically used for the CD-ROM and floppy drives. There are other options, and /etc/fstab even has its own man page (fstab(5)).

One of its great advantages is that it simplifies the mount command syntax. To mount a filesystem referenced in it, you can either reference the mount point or the device. To mount a floppy disk, you can type:

```
$ mount /mnt/floppy
```

or:

```
$ mount /dev/fd0
```

To finish with our partition moving example: we copied the /usr hierarchy and completed /etc/fstab, in order for the new partition to be mounted at start-up. But for the moment, the old /usr files are still there! We therefore need to delete them to free up space (which was, after all, our initial goal). To do so, you first need to go to single user mode (by issuing the *telinit 1* command on the command line), and then:

- delete all files in the /usr directory (i.e. the “old” one, since the “new” one is not yet mounted): `rm -Rf /usr/*;`
- mount the “new” /usr: `mount /usr/.`

And that's it. Now, go back to multiuser mode (`telinit 3` or `telinit 5`), and if there is no further administrative work left, you should now log off the root account.

7.5. A Note About The Supermount Feature

Newer kernels as those shipped with **Mandrake Linux** bring an interesting feature for users frequently using floppy and CD disks. Installed (or not) depending on the chosen security level, it automatically mounts and unmounts media as they are inserted or removed. This is quite handy as there is no need to run `mount` or `umount` each time.

Chapter 8. The Linux Filesystem

Naturally, your *GNU/Linux* system is contained on your hard disk within a filesystem. Here we present different aspects related to filesystems, as well as the possibilities they offer.

8.1. Comparison of a Few Filesystems

During the installation, you can choose different **file systems** for your partitions. This means you can format your partitions according to different algorithms.

Unless you are a specialist, choosing a filesystem is not obvious. We propose a rapid presentation of three of the most current file systems, which are all available under **Mandrake Linux**.

8.1.1. Different Usable Filesystems

8.1.1.1. Ext2FS

The **Second Extended Filesystem** (its abbreviated form is **Ext2FS** or simply **ext2**) has been *GNU/Linux*'s default filesystem for many years. It replaced the **Extended File System** (that's where the "Second" comes from). The "new" filesystem corrected certain problems and limitations.

Ext2FS respects the usual standards for Unix-type filesystems. Since its conception, it was destined to evolve, while still offering a great robustness and good performances.

8.1.1.2. Ext3

Like its name suggests, the **Third Extended File System** is Ext2FS' successor. It is compatible with the latter but it is enhanced by a very interesting feature: **journaling**.

One of the major flaws of "traditional" filesystems like Ext2FS is their low tolerance to abrupt system breakdowns (power failure or crashing software). Generally speaking, such events involve a very long exam of the filesystem's structure, attempts to correct errors, sometimes resulting in an extended corruption. Hence, a partial or total lost of saved data.

Journaling answers this problem. To simplify, let's say that the object is to save actions (such as the saving of a file) **before** really doing it. We could compare its functioning to the one of a boat captain whom notes in his log book daily events. The result: an always coherent filesystem. And if problems occur, the verification is very rapid and the eventual repairs, very limited. The time spent to verify a filesystem is thus proportional to its actual use and no more its size.

Hence, Ext3FS offers the journal filesystem technology, while keeping Ext2FS' structure – ensuring an excellent compatibility.

8.1.1.3. ReiserFS

Unlike to Ext3FS, **ReiserFS** is created from scratch. It is journalized like Ext3FS, but its internal structure is radically different. Specifically, it uses binary-tree concepts inspired by database software.

8.1.1.4. JFS

JFS is the journalized filesystem designed and used by IBM. Proprietary and closed at first, IBM recently decided to open the access to free software movement. Its internal structure is near to ReiserFS' one.

	Ext2FS	Ext3FS	ReiserFS	JFS
--	--------	--------	----------	-----

8.1.2. Differences Between those Filesystems

	Ext2FS	Ext3FS	ReiserFS	JFS
Stability	Excellent	Good	Good	Medium
Tools to recuperate erased files	Yes (complex)	Yes (complex)	No	No
Reboot time after crash	Long, even very long	Fast	Very fast	Very fast
Sum of the data in case of a crash	Generally speaking, good, but high risk of partial or total data loss	N/A	Very good. Complete data loss is very rare	Very good

Table 8-1. Filesystem Characteristics

About file maximum sizes, it depends on a lot of parameters (e.g. as block size for ext2/ext3), and is likely to evolve depending on the kernel version and architecture. Nonetheless, the minimum available, according to the filesystem limits, is currently generally near or superior to 2Tb (1Tb=1024 Gb) and can go up to 4Pb (1Pb=1024 Tb) for JFS. Unfortunately, these values are also limited to maximum block device size, which in current 2.4.X kernel is limited (for X86 arch only) to 2TB¹ even in RAID mode. For more information, consult Adding Support for Arbitrary File Sizes to the Single UNIX Specification (http://ftp.sas.com/standards/large.file/x_open.20Mar96.html).

8.1.3. And Performance Wise?

It is always very delicate to compare performances. Every tests have their limitations and the results must be interpreted with caution. Nowadays, Ext2FS is very mature but its development is scarce; on the other hand, journal filesystems like Ext3FS and ReiserFS evolve very rapidly. Tests done a couple of months or weeks ago are already too old. Let's not forget that today's material (specially concerning hard drive capacities) has greatly leveraged the differences between them. However JFS is currently the one showing best performances.

Each system offers advantages and disadvantages. In fact, it all depends on how you use your machine. A simple desktop machine will be happy with Ext2FS. For a server, a journaled filesystem Ext3FS is preferred. ReiserFS, perhaps because of its genesis, is more suited to a database server. JFS is itself preferred in cases where filesystem throughput is the main issue.

For a "normal" use, the four filesystems give approximately the same results. ReiserFS allows to access small files rapidly, but it is fairly slow to manipulate large files (many megabytes). In most cases, the advantages brought by ReiserFS' journaling capabilities render its drawbacks to be of minimal importance.

8.2. Everything is a File

The *User Guide* introduced the file ownership and permissions access concepts, but really understanding the *UNIX filesystem* (and this also applies to *GNU/Linux' ext2fs*) requires that we redefine the file concept itself.

Here, "everything" **really** means everything. A hard disk, a partition on a hard disk, a parallel port, a connection to a web site, an *Ethernet* card, all these are files. Even directories are files. *GNU/Linux* recognizes many types of files in addition to the standard files and directories. Note that by file type here, we do not mean the type of the **contents** of a file: for *GNU/Linux* and any *UNIX* system, a file, whether it be a PNG image, a binary file or whatever, is just a stream of bytes. Differentiating files according to their contents is left to applications.

¹. You may wonder how to achieve such capacities with hard drives that hardly reach 180Gb. In fact, using 3 RAID cards hosting each 8*128Gb drives, you reach 3Tb...

8.2.1. The Different File Types

If you remember well, when you do `ls -l`, the character before the access rights identifies the type of a file. We already saw two types of files: regular files (-) and directories (d). You can also stumble upon these other types if you wander through the file tree and list contents of directories:

1. **Character mode files:** these files are either special system files (such as `/dev/null`, which we already discussed), or peripherals (serial or parallel ports), which share the particularity that their contents (if they have any) are not *buffered* (meaning they are not kept in memory). Such files are identified by the letter `c`.
2. **Block mode files:** these files are peripherals, and as opposed to character files, their contents **are** buffered. Files entering this category are, for example, hard disks, partitions on a hard disk, floppy drives, CD-ROM drives and so on. Files `/dev/hda`, `/dev/sda5` are example of block mode files. On a `ls -l` output, these are identified by the letter `b`.
3. **Symbolic links:** these files are very common, and heavily used in the **Mandrake Linux** system startup procedure (see chapter “*The Start-Up Files: init sysv*”, page 57). As their name implies, their purpose is to link files in a symbolic way, which means that such files may or may not point to an existing file. This will be explained later in this chapter. They are very frequently (and wrongly, as we will see later) called “*soft links*”, and are identified by an ‘`l`’.
4. **Named pipes:** in case you were wondering, yes, these are very similar to pipes used in *shell* commands, but with the difference that these ones actually have names. Read on to learn more. They are very rare, however, and it is very unlikely that you will see one during your journey into the file tree. Just in case you do, the letter identifying them is ‘`p`’. To learn more about it, have a look at “*Anonymous Pipes and Named Pipes*”, page 48.
5. **Sockets:** this is the file type for all network connections. Only a few of them have names, though. What’s more, there are different types of sockets and only one can be linked, but this is way beyond the scope of this book. Such files are identified by the letter ‘`s`’.

Here is a sample of each file:

```
$ ls -l /dev/null /dev/sda /etc/rc.d/rc3.d/S20random /proc/554/maps \
/tmp/ssh-queen/ssh-510-agent
crw-rw-rw-  1 root  root      1,   3 May  5 1998 /dev/null
brw-rw----  1 root  disk      8,   0 May  5 1998 /dev/sda
lrwxrwxrwx  1 root  root      16 Dec  9 19:12 /etc/rc.d/rc3.d/
S20random -> ../init.d/random*
pr--r--r--  1 queen queen    0 Dec 10 20:23 /proc/554/maps|
srwx-----  1 queen queen    0 Dec 10 20:08 /tmp/ssh-queen/
ssh-510-agent=
$
```

8.2.2. Inodes

Inodes are, with the “Everything Is a File” paradigm, the fundamental part of any *UNIX* file system. The word “*inode*” is short for *Information NODE*.

Inodes are stored on disk in an **inode table**. They exist for all types of files which may be stored on a filesystem, and this includes directories, named pipes, character mode files and so on. Which leads to this other famous sentence: “The inode is the file”. Inodes are also the way by which *UNIX* identifies a file in a unique way.

Yes, you read well: on *UNIX*, you **do not identify a file by its name**, but by its inode number.² The reason for this is that a same file can have several names, or even no name. A file name, in *UNIX*, is just an entry in a directory inode. Such an entry is called a **link**. Let’s look at links in more detail.

2. Important: notice that inode numbers are unique **per filesystem**, which means that an inode with a same number can exist on another filesystem. Which leads to the difference between on-disk inodes and in-memory inodes. While two on-disk inodes can have the same number if they are on two different filesystems, in-memory inodes have a unique number all across the system. One solution to obtain uniqueness, for example, to hash the on-disk inode number against the block device identifier.

8.3. Links

The best way to understand what links are is to take an example. Let's create a (regular) file:

```
$ pwd
/home/queen/example
$ ls
$ touch a
$ ls -il a
32555 -rw-rw-r-- 1 queen queen 0 Dec 10 08:12 a
```

The `-i` option of the `ls` command prints the inode number, which is the first field on the output. As you can see, before we created file `a`, there were no files in the directory. The other field of interest is the third one, which is the number of file links (well, inode links, in fact).

The `touch a` command can be separated into two distinct actions:

- creation of an inode, to which the operating system has given the number 32555, and whose type is the one of a regular file;
- and creation of a link to this inode, named `a`, in the current directory, `/home/queen/example`. Therefore, the `/home/queen/example/a` file is a link to the inode numbered 32555, and it is currently the only one: the link counter shows 1.

But now, if we type:

```
$ ln a b
$ ls -il a b
32555 -rw-rw-r-- 2 queen queen 0 Dec 10 08:12 a
32555 -rw-rw-r-- 2 queen queen 0 Dec 10 08:12 b
$
```

we create another link to the same inode. As you can see, we did not create any file named `b`, but instead, we just added another link to the inode numbered 32555 in the same directory, and attributed the name `b` to this new link. You can see on the `ls -l` output that the link counter for the inode is now 2, and no longer 1.

Now, if we do:

```
$ rm a
$ ls -il b
32555 -rw-rw-r-- 1 queen queen 0 Dec 10 08:12 b
$
```

we see that even though we deleted the “original file”, the inode still exists. But now, the only link to it is the file named `/home/queen/example/b`.

Therefore, a file under *UNIX* has no name; instead, it has one or more *link(s)* in one or more directory(ies).

Directories themselves are also stored into inodes, but their link count, unlike all other file types, is the number of subdirectories within them. There are at least two links per directory: the directory itself (`.`) and its parent directory (`..`).

Typical examples of files which are not linked (i.e.: have no name) are network connections: you will never see the file corresponding to your connection to the Mandrake Linux web site (www.mandrakelinux.com) in your file tree, whichever directory you try. Similarly, when you use a *pipe* in the *shell*, the inode corresponding to the pipe does exist, but it is not linked.

8.4. “Anonymous” Pipes and Named Pipes

Let's get back to the example of pipes, as it is quite interesting and is also a good illustration of the links notion. When you use a pipe in a command line, the *shell* creates the pipe for you and operates so that the command before the pipe writes to it, whereas the command after the pipe reads from it. All pipes, whether they be anonymous (like the ones used by the *shells*) or named (see below) act like FIFOs (*First In, First Out*). We already saw examples of how to use pipes in the *shell*, but let's take one for the sake of our demonstration:

```
$ ls -d /proc/[0-9] | head -5
/proc/1/
/proc/2/
/proc/3/
/proc/4/
```



```
/proc/5/
```

One thing that you will not notice in this example (because it happens too fast for one to see) is that writes on pipes are blocking. It means that when the `ls` command writes to the pipe, it is blocked until a process at the other end reads from the pipe. In order to visualize the effect, you can create named pipes, which, as opposite to the pipes used by *shells*, have names (i.e.: they are linked, whereas *shell* pipes are not).³ The command to create such pipes is `mkfifo`:

```
$ mkfifo a_pipe
$ ls -il
total 0
 169 prw-rw-r-- 1 queen   queen      0 Dec 10 14:12 a_pipe|
#
# You can see that the link counter is 1, and that the output shows
# that the file is a pipe ('p').
#
# You can also use ln here:
#
$ ln a_pipe the_same_pipe
$ ls -il
total 0
 169 prw-rw-r-- 2 queen   queen      0 Dec 10 15:37 a_pipe|
 169 prw-rw-r-- 2 queen   queen      0 Dec 10 15:37 the_same_pipe|
$ ls -d /proc/[0-9] >a_pipe
#
# The process is blocked, as there is no reader at the other end.
# Type C-z to suspend the process...
#
zsh: 3452 suspended  ls -d /proc/[0-9] > a_pipe
#
# ...Then put in into the background:
#
$ bg
[1] + continued  ls -d /proc/[0-9] > a_pipe
#
# now read from the pipe...
#
$ head -5 <the_same_pipe
#
# ...the writing process terminates
#
[1] + 3452 done      ls -d /proc/[0-9] > a_pipe
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
#
```

Similarly, reads are also blocking. If we execute the above commands in the reverse order, we observe that `head` blocks, waiting for some process to give it something to read:

```
$ head -5 <a_pipe
#
# Program blocks, suspend it: C-z
#
zsh: 741 suspended  head -5 < a_pipe
#
# Put it into the background...
#
$ bg
[1] + continued  head -5 < a_pipe
#
# ...And give it some food :)
#
$ ls -d /proc/[0-9] >the_same_pipe
$ /proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1] + 741 done      head -5 < a_pipe
$
```

3. Other differences exist between the two kinds of pipes, but they are out of the scope of this book.

You can also see an undesired effect in the previous example: the `ls` command has terminated before the `head` command took over. The consequence is that you got back at the prompt immediately, but `head` executed only after. Therefore, it made its output only after you got back to the prompt.

8.5. “Special” Files: Character Mode and Block Mode Files

As already stated, such files are either files created by the system or peripherals on your machine. We also mentioned that the contents of block mode character files were buffered, whereas character mode files were not. In order to illustrate this, insert a floppy into the drive and type the following command twice:

```
$ dd if=/dev/fd0 of=/dev/null
```

You can observe the following: while, the first time the command was launched, the whole contents of the floppy were read, the second time, there was no access to the floppy drive at all. This is simply because the contents of the floppy were buffered when you first launched the command – and you did not change the floppy meanwhile.

But now, if you want to print a big file this way (yes it will work):

```
$ cat /a/big/printable/file/somewhere >/dev/lp0
```

the command will take as much time, whether you launch it once, twice or fifty times. This is because `/dev/lp0` is a character mode file, and its contents are not buffered.

The fact that block mode files are buffered have a nice side effect: not only are reads buffered, but writes are buffered too. This allows for writes on disks to be asynchronous: when you write a file on disk, the write operation itself is not immediate. It will only occur when *GNU/Linux* decides for it.

Finally, each special file has a *major* and *minor* number. On a `ls -l` output, they appear in place of the size, as the size for such files is irrelevant:

```
ls -l /dev/hda /dev/lp0
brw-rw---- 1 root   disk      3,   0 May  5 1998 /dev/hda
crw-rw---- 1 root   daemon    6,   0 May  5 1998 /dev/lp0
```

Here, the major and minor of `/dev/hda` are respectively 3 and 0, whereas for `/dev/lp0`, they are respectively 6 and 0. Note that these numbers are unique per file category, which means that there can be a character mode file with major 3 and minor 0 (this file actually exists: `/dev/tty0`), and similarly, there can be a block mode file with major 6 and minor 0. These numbers exist for a simple reason: it allows *GNU/Linux* to associate the correct operations to these files (that is, to the peripherals these files refer to): you do not handle a floppy drive the same way as, say, a SCSI hard drive.

8.6. Symbolic Links, Limitation of “Hard” Links

Here we have to face a very common misconception, even among *UNIX* users, which is mainly due to the fact that links as we have seen them so far (wrongly called “hard” links) are only associated with regular files (and we have seen that it is not the case – all the more that even symbolic links are “linked”). But this requires that we first explain what symbolic links (“soft” links, or even more often “symlinks”) are.

Symbolic links are files of a particular type whose sole contents is an arbitrary string, which may or may not point to an existing file. When you mention a symbolic link on the command line or in a program, in fact, you access the file it points to, if it exists. For example:

```
$ echo Hello >myfile
$ ln -s myfile mylink
$ ls -il
total 4
 169 -rw-rw-r-- 1 queen   queen           6 Dec 10 21:30 myfile
 416 lrwxrwxrwx 1 queen   queen           6 Dec 10 21:30 mylink
-> myfile
$ cat myfile
Hello
$ cat mylink
Hello
```

You can see that the file type for `mylink` is `'l'`, for symbolic *Link*. The access rights for a symbolic link are not significant: they will always be `lrwxrwxrwx`. You can also see that it is a different file from `myfile`, as its inode

number is different. But it refers to it symbolically, therefore when you type `cat mylink`, you will in fact print the contents of the `myfile` file. To demonstrate that a symbolic link contains an arbitrary string, we can do the following:

```
$ ln -s "I'm no existing file" anotherlink
$ ls -il anotherlink
    418 lrwxrwxrwx    1 queen    queen          20 Dec 10 21:43 anotherlink
-> I'm no existing file
$ cat anotherlink
cat: anotherlink: No such file or directory
$
```

But symbolic links exist because they overcome several limitations encountered by normal (“hard”) links:

- You cannot create a link to an inode in a directory which is on a different filesystem than the said inode. The reason is simple: the link counter is stored in the inode itself, and inodes can not be shared along filesystems. Symlinks allow this;
- You cannot link directories, as we have seen that the link counter for a directory has a special usage. But you can make a symlink point to a directory and use it as if it were actually a directory.

Symbolic links are therefore very useful in several circumstances, and very often, people tend to use them to link files together even when a normal link could be used instead. One advantage of normal linking, though, is that you do not lose the file if you delete the “original one”.

Lastly, if you observed carefully, you know what the size of a symbolic link is: it is simply the size of the string.

8.7. File Attributes

The same way that FAT has file attributes (archive, system file, invisible), *ext2fs* has its own, but they are different. We speak of them here for the sake of completeness, but they are very seldom used. However, if you really want a secure system, read on.

There are two commands for manipulating file attributes: `lsattr(1)` and `chattr(1)`. You probably guessed it, `lsattr` *LiSts* attributes, whereas `chattr` *CHanges* them. These attributes can only be set on directories and regular files. The following attributes are possible:

1. *A (no Access time)*: if a file or directory has this attribute set, whenever it is accessed, either for reading or for writing, its last access time will not be updated. This can be useful, for example, on files or directories which are very often accessed for reading, especially since this parameter is the only one which changes on an inode when it's open read-only.
2. *a (append only)*: if a file has this attribute set and is open for writing, the only operation possible will be to append data to its previous contents. For a directory, this means that you can only add files to it, but not rename or delete any existing file. Only root can set or clear this attribute.
3. *d (no dump)*: `dump (8)` is the standard *UNIX* utility for backups. It dumps any filesystem for which the dump counter is 1 in `/etc/fstab` (see chapter “Filesystems and Mount Points”, page 41). But if a file or directory has this attribute set, unlike others, it will not be taken into account when a dump is in progress. Note that for directories, this also includes all subdirectories and files under it.
4. *i (immutable)*: a file or directory with this attribute set simply can not be modified at all: it can not be renamed, no further link can be created to it ⁴ and it cannot be removed. Only root can set or clear this attribute. Note that this also prevents changes to access time, therefore you do not need to set the *A* attribute when *i* is set.
5. *s (secure deletion)*: when such a file or directory with this attribute set is deleted, the blocks it was occupying on disk are written back with zeroes.
6. *S (Synchronous mode)*: when a file or directory has this attribute set, all modifications on it are synchronous and written back to disk immediately.

You may want, for example, to set the ‘*i*’ attribute on essential system files in order to avoid bad surprises. Also, consider the ‘*A*’ attribute on man pages for example: this prevents a lot of disk operations and, in particular, it saves some battery life on laptops.

4. Be sure to understand what “adding a link” means, both for a file and a directory :-)

Chapter 9. The /proc Filesystem

The /proc filesystem is specific to *GNU/Linux*. It is a virtual filesystem, and as such it takes no room on your disk. It is a very convenient way to obtain information on the system, all the more that most files in this directory are human readable (well, with a little help). Many programs actually gather information from files in /proc, format it in their own way and then display it. This is the case for all programs which display information about processes, and we have already seen a few of them (*top*, *ps* and *friends*). /proc is also a good source of information about your hardware, and similarly, quite a few programs are just interfaces to the information contained in /proc.

There is also a special subdirectory, /proc/sys. It allows for changing some kernel parameters in real time or displaying them.

9.1. Information About Processes

If you list the contents of the /proc directory, you will see many directories that the name of which is a number. These are the directories holding information on all processes currently running on the system:

```
$ ls -d /proc/[0-9]*
/proc/1/      /proc/302/    /proc/451/    /proc/496/    /proc/556/    /proc/633/
/proc/127/    /proc/317/    /proc/452/    /proc/497/    /proc/557/    /proc/718/
/proc/2/      /proc/339/    /proc/453/    /proc/5/      /proc/558/    /proc/755/
/proc/250/    /proc/385/    /proc/454/    /proc/501/    /proc/559/    /proc/760/
/proc/260/    /proc/4/      /proc/455/    /proc/504/    /proc/565/    /proc/761/
/proc/275/    /proc/402/    /proc/463/    /proc/505/    /proc/569/    /proc/769/
/proc/290/    /proc/433/    /proc/487/    /proc/509/    /proc/594/    /proc/774/
/proc/3/      /proc/450/    /proc/491/    /proc/554/    /proc/595/
```

Note that as a user, you can (logically) only display information related to your own processes, but not the ones of other users. So, let's be root and see what information is available from process 127:

```
$ su
Password:
$ cd /proc/127
$ ls -l
total 0
-r--r--r-- 1 root root 0 Dec 14 19:53 cmdline
lrwx----- 1 root root 0 Dec 14 19:53 cwd -> //
-r----- 1 root root 0 Dec 14 19:53 environ
lrwx----- 1 root root 0 Dec 14 19:53 exe -> /usr/sbin/apmd*
dr-x----- 2 root root 0 Dec 14 19:53 fd/
pr--r--r-- 1 root root 0 Dec 14 19:53 maps|
-rw----- 1 root root 0 Dec 14 19:53 mem
lrwx----- 1 root root 0 Dec 14 19:53 root -> //
-r--r--r-- 1 root root 0 Dec 14 19:53 stat
-r--r--r-- 1 root root 0 Dec 14 19:53 statm
-r--r--r-- 1 root root 0 Dec 14 19:53 status
$
```

Each directory contains the same entries. Here is a brief description of some of the entries:

1. *cmdline*: this (pseudo-)file contains the whole command line used to invoke the process. It is not formatted: there is no space between the program and its arguments, and there is no newline at the end of the line either. In order to view it, you can use: `perl -ple 's,\00, ,g' cmdline`.
2. *cwd*: this symbolic link points to the current working directory (hence the name) of the process.
3. *environ*: This file contains all the environment variables defined for this process, in the form `VARIABLE=value`. Similar to *cmdline*, the output is not formatted at all: no newlines to separate between different variables, and no newline at the end either. One solution to view it: `perl -pl -e 's,\00,\n,g' environ`.
4. *exe*: this is a symlink pointing to the executable file corresponding to the process being run.
5. *fd*: this subdirectory contains the list of file descriptors currently opened by the process. See below.
6. *maps*: when you print the contents of this named pipe (with *cat* for example), you can see the parts of the process' address space which are currently mapped to a file. The fields from left to right are: the address space associated to this mapping, the permissions associated to this mapping, the offset from the beginning of the file where the mapping starts, the major and minor number (in hexadecimal) of the device

on which the mapped file is located, the inode number of the file, and finally the name of the file itself. When the device is 0 and there's no inode number and filename either, these are anonymous mappings. See `mmap(2)`.

7. `root`: this is a symbolic link which points to the root directory used by the process. Usually, it will be `/`, but see `chroot(2)`.
8. `status`: this file contains various information about the process: the name of the executable, its current state, its PID and PPID, its real and effective UID and GID, its memory usage, and other information.

If we list the contents of directory `fd`, always for our process 127, we obtain this:

```
$ ls -l fd
total 0
lrwx----- 1 root    root          64 Dec 16 22:04 0 -> /dev/console
l-wx----- 1 root    root          64 Dec 16 22:04 1 -> pipe:[128]
l-wx----- 1 root    root          64 Dec 16 22:04 2 -> pipe:[129]
l-wx----- 1 root    root          64 Dec 16 22:04 21 -> pipe:[130]
lrwx----- 1 root    root          64 Dec 16 22:04 3 -> /dev/apm_bios
lr-x----- 1 root    root          64 Dec 16 22:04 7 -> pipe:[130]
lrwx----- 1 root    root          64 Dec 16 22:04 9 ->
/dev/console
$
```

In fact, this is the list of file descriptors opened by the process. Each opened descriptor is shown by a symbolic link, the name of which is the descriptor number, and which points to the file opened by this descriptor¹. You can also notice the permissions on the symlinks: this is the only place where they make sense, as they represent the permissions with which the file corresponding to the descriptor has been opened.

9.2. Information on The Hardware

Apart from the directories associated to the different processes, `/proc` also contains a myriad of information on the hardware present in your machine. A list of files from the `/proc` directory gives the following:

```
$ ls -d [a-z]*
apm      dma      interrupts  loadavg  mounts    rtc      swaps
bus/     fb        ioports    locks    mtrr      scsi/    sys/
cmdline  filesystems kcore      meminfo  net/      self/    tty/
cpuinfo  fs/       kmsg       misc     partitions slabinfo uptime
devices  ide/      ksyms      modules  pci       stat     version
$
```

For example, if we look at the contents of `/proc/interrupts`, we can see that it contains the list of interrupts currently used by the system, along with the peripheral which holds them. Similarly, `ioports` contains the list of input/output address ranges currently busy, and lastly `dma` does the same for DMA channels. Therefore, in order to chase down a conflict, look at the contents of these three files:

```
$ cat interrupts
CPU0
0:    127648      XT-PIC  timer
1:    5191       XT-PIC  keyboard
2:      0        XT-PIC  cascade
5:    1402       XT-PIC  xirc2ps_cs
8:      1        XT-PIC  rtc
10:     0        XT-PIC  ESS Solo1
12:    2631      XT-PIC  PS/2 Mouse
13:      1       XT-PIC  fpu
14:   73434      XT-PIC  ide0
15:   80234      XT-PIC  ide1
NMI:      0
$ cat ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
```

1. If you remember what has been told in section *Redirections and Pipes*, page 13, you know what descriptors 0, 1 and 2 stand for.

```

00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0300-030f : xirc2ps_cs
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
1050-1057 : ide0
1058-105f : ide1
1080-108f : ESS Solo1
10c0-10cf : ESS Solo1
10d4-10df : ESS Solo1
10ec-10ef : ESS Solo1
$ cat dma
4: cascade
$

```

Or, more simply, use the `lsdev` command, which gathers information from these three files and sorts them by peripheral, which is undoubtedly more convenient²:

```

$ lsdev
Device          DMA   IRQ  I/O Ports
-----
cascade         4     2
dma              0080-008f
dma1             0000-001f
dma2             00c0-00df
ESS              1080-108f 10c0-10cf 10d4-10df 10ec-10ef
fpu             13  00f0-00ff
ide0            14  01f0-01f7 03f6-03f6 1050-1057
ide1            15  0170-0177 0376-0376 1058-105f
keyboard        1  0060-006f
Mouse           12
pic1             0020-003f
pic2             00a0-00bf
rtc              8  0070-007f
serial          03f8-03ff
Solo1           10
timer           0  0040-005f
vga+            03c0-03df
xirc2ps_cs      5  0300-030f
$

```

An exhaustive listing of files would be too long, but here's the description of some of them:

- `cpuinfo`: this file contains, as its name says, information on the processor(s) present in your machine.
- `modules`: this file contains the list of modules currently used by the kernel, along with the usage count for each one. In fact, this is the same information as what is reported by the `lsmod` command.
- `meminfo`: this file contains information on memory usage at the time you print its contents. A more clearly formatted output of the same information is available with the `free` command.
- `apm`: if you have a laptop, displaying the contents of this file allows you to see the state of your battery. You can see whether the AC is plugged in, the current load of your battery, and if the APM *BIOS* of your laptop supports it (unfortunately this is not the case for all), the remaining battery life in minutes. The file isn't very readable by itself, therefore you want to use the `apm` command instead, which gives the same information in a human readable format.
- `bus`: this subdirectory contains information on all peripherals found on different buses in your machine. Information inside it are generally seldom readable, and for the most part they are dealt with and reformatted with external utilities: `lspcidrake`, `lspnp`, etc.

² `lsdev` is part of the `procinfo` package.

9.3. The `/proc/sys` Sub-Directory

The role of this subdirectory is to report different kernel parameters, and to allow for changing in real time some of these parameters. As opposed to all other files in `/proc`, some files in this directory can be written to, but by root only.

A list of directories and files would be too long, all the more that they will depend in a large part on your system, and that most files will only be useful for very specialized applications. However, here are three common uses of this subdirectory:

1. Allow routing: Even if the default kernel from **Mandrake Linux** is able to route, you must explicitly allow it to do so. For this, you just have to type the following command as root:

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

Replace the 1 by a 0 if you want to forbid routing.

2. Prevent IP spoofing: IP spoofing consists in making one believe that a packet coming from the outside world comes from the interface by which it arrives. This technique is very commonly used by *crackers*³, but you can make the kernel prevent this kind of intrusion for you. You just have to type:

```
$ echo 1 >/proc/sys/net/ipv4/conf/all/rp_filter
```

and this kind of attack becomes impossible.

3. Increase the size of the table of open files and the inode table: The size of the table of open files and the inode table is dynamic under *GNU/Linux*. The default values are usually sufficient for normal use, but they may be too weak if your machine is a huge server (a database server for example). Indeed, the first obstacle is the fact that processes cannot open any more files because the table is full, therefore you must increase its size. Meanwhile, you must also increase the size of the inode table. These two lines will solve the problem:

```
$ echo 8192 >/proc/sys/fs/file-max
$ echo 16384 >/proc/sys/fs/inode-max
```

In order for these to be executed each time you boot the system, you can add all these lines to `/etc/rc.d/rc.local` so that you avoid typing them each time, but another solution is to fill in `/etc/sysctl.conf`, see `sysctl.conf(5)`.

3. And not *hackers*!

Chapter 10. The Start-Up Files: init sysv

In the *UNIX* tradition, there are two system startup schemes: the *BSD* scheme and the “*System V*” scheme, both named after the *UNIX* system which implemented them first (resp. *Berkeley Software Distribution* and *AT&T UNIX System V*). The *BSD* scheme is more simple, but the *System V* scheme, although less easy to understand (which will change once you finish this chapter), is definitely more flexible to use.

10.1. In The Beginning Was init

When the system starts, and after the kernel has configured everything and mounted the root filesystem, it executes `/sbin/init`¹. `init` is the father of all processes of the system, and it is responsible for taking the system to the desired *runlevel*. We will look at runlevels in the next section.

The `init` configuration file is `/etc/inittab`. This file has its own manual page (`inittab(5)`), but here we will describe only a few of the configuration items.

The first line which should be the focus of your attention is this one:

```
si::sysinit:/etc/rc.d/rc.sysinit
```

This instruction tells `init` that `/etc/rc.sysinit` is to be run on initialization of the system (`si` stands for *System Init*) before anything else. To determine the default runlevel, `init` then looks for the line containing the `initdefault` keyword:

```
id:5:initdefault:
```

In this case, `init` knows that the default runlevel is 5. It also knows that to enter level 5, it must run the following command:

```
l5:5:wait:/etc/rc.d/rc 5
```

As you can see, the syntax for each runlevel is similar.

`init` is also responsible for restarting (respawn) some programs, which it is the only process capable of restarting. This is the case, for example, for all login programs which run in each of the 6 virtual consoles². For the second virtual console, this gives:

```
2:2345:respawn:/sbin/mingetty tty2
```

10.2. Runlevels

All files related to system startup are located in the directory `/etc/rc.d`. Here is the list of the files:

```
$ ls /etc/rc.d
init.d/  rc.local*  rc0.d/  rc2.d/  rc4.d/  rc6.d/
rc*      rc.sysinit* rc1.d/  rc3.d/  rc5.d/
```

To begin with, as we have seen, the `rc.sysinit` file is run. This is the file responsible for setting up the basic machine configuration: keyboard type, configuration of certain devices, filesystem checking, etc.

Then the `rc` script is run, with the desired runlevel as an argument. As we have seen, the runlevel is a simple integer, and for each runlevel `<x>` defined, there must be a corresponding `rc<x>.d` directory. In a typical **Mandrake Linux** installation, you might therefore see that 6 runlevels are defined:

- 0: complete machine stop;
- 1: *single-user* mode; to be used in the event of major problems or system recovery;
- 2: *multi-user* mode, without networking;
- 3: Multi-user mode, but this time with networking;

1. Now you see why putting `/sbin` on a filesystem other than the root filesystem is a very bad idea :-)

2. So you can, if you want, add or remove virtual consoles by modifying this file, up to a maximum of 64, by following the syntax. But don't forget that `X` also runs on a virtual console! So leave it at least one free for it.

- 4: unused;
- 5: like 3, but also launches the graphical login interface;
- 6: restart.

Let us look, for example, at the contents of directory `rc5.d`:

```
$ ls rc5.d
K15postgresql@ K60atd@      S15netfs@      S60lpd@        S90xfs@
K20nfs@         K96pcmcia@    S20random@    S60nfs@        S99linuxconf@
K20rstatd@      S05apmd@      S30syslog@    S66yppasswdd@  S99local@
K20rusersd@     S10network@   S40crond@     S75keytable@
K20rwhod@       S11portmap@   S50inet@      S85gpm@
K30sendmail@    S12ypserv@    S55named@     S85httpd@
K35smb@         S13ypbind@    S55routed@    S85sound@
```

As you can see, all the files in this directory are symbolic links, and they all have a very specific form. Their general form is:

```
<S|K><order><service_name>
```

The *S* means *Start* service, and *K* means *Kill* (stop) service. The scripts are run in ascending number order, and if two scripts have the same number, alphabetical order applies. We can also see that each symbolic link points to a given script located in `/etc/rc.d/init.d` (apart from `local`), script which is responsible for controlling a specific service.

When the system goes into a given runlevel, it starts by running the *K* links in order: `rc` looks where the link is pointing, then calls up the corresponding script with the single argument `stop`. Then it runs the *S* scripts, still using the same method, apart from the fact that the script is called with the argument `start`.

Thus, without mentioning all the scripts, we can see that when the system goes into runlevel 5, it first runs `K15postgresql`, i.e. `/etc/rc.d/init.d/postgresql stop`. Then `K20nfs`, then `K20rstatd`, until the last one; next, it runs all the *S* scripts: first `S05apmd`, which then calls `/etc/rc.d/init.d/apmd start`, and so on.

Armed with all this, you can create your own entire runlevel in a few minutes, or prevent a service starting or stopping by deleting the corresponding symbolic link (there are also interface programs for doing this, notably *drakxservices* and *chkconfig*; the former is a graphical program).

III. Advanced Uses

Chapter 11. Printing

This chapter is divided in two parts: *Installing And Managing Printers*, page 61, devoted to people administering their own machines; and *Printing Documents*, page 66, which explains how to use an advanced printing tool: *XPP*.

11.1. Installing And Managing Printers

Starting from release 7.2, **Mandrake Linux** began using the new printing system based on *cups* (<http://www.cups.org/>)¹. This is a very powerful tool based on decentralized management and configuration, making all printers in a local network available to all users.

11.1.1. Install CUPS And Browse Its Web Interface

cups is now the default printer manager for **Mandrake Linux**, and all necessary packages should be installed by default. If it is not the case, make sure that at least the packages *cups*, *cups-drivers* and *xpp* are installed.



There are basically two ways to manage your printers with *cups*: a web interface and a manager part of the *KDE* Control Center (**System**→**Printing Management**). We chose to concentrate on the web interface as it is accessible from any platform and allows for remote printing management. You can get full documentation about *KDE* printing tools at their web site (<http://printing.kde.org/>).

From your preferred web browser, simply type <http://localhost:631/> in the location or URL field. It will display the main *cups* menu (figure 11-1).

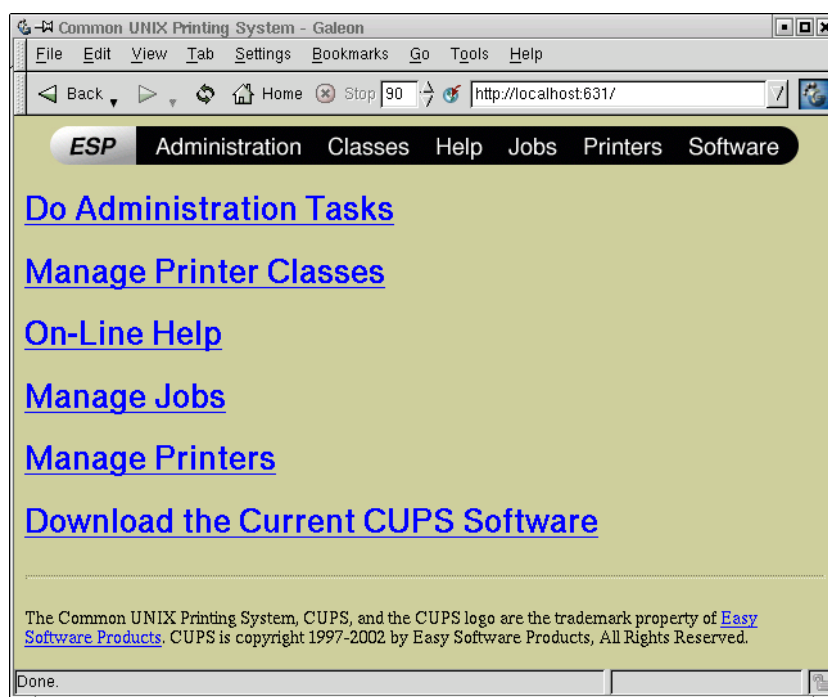


Figure 11-1. The CUPS Welcome Page

You can now browse the configuration interface like a web site.

1. Common Unix Printing System

11.1.2. Configure a New Printer

Depending whether your LAN already has machines with *cups* installed and running, you may see a list of printers under the **Manage Printers** link. We will assume that you are now installing a printer connected to your stand-alone computer. For more complex configurations consult the **On-Line Help**.

The **Manage Printers** page (figure 11-2) should look empty for now.

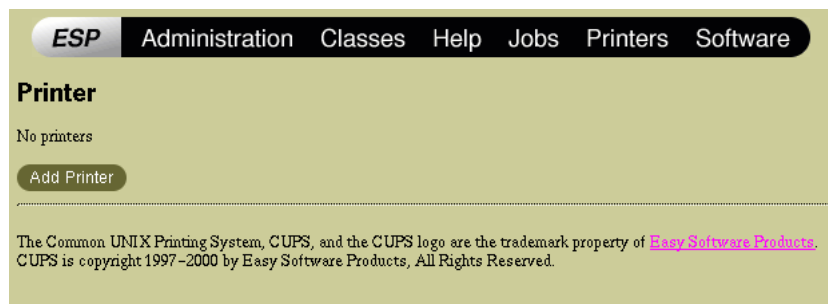


Figure 11-2. The Empty CUPS Printers List

To configure a new printer, now click on the **Add Printer** button at the bottom of the page. It will begin a four-step procedure. To go from one step to the following click the **Continue** button after filling-in all required fields on the page.



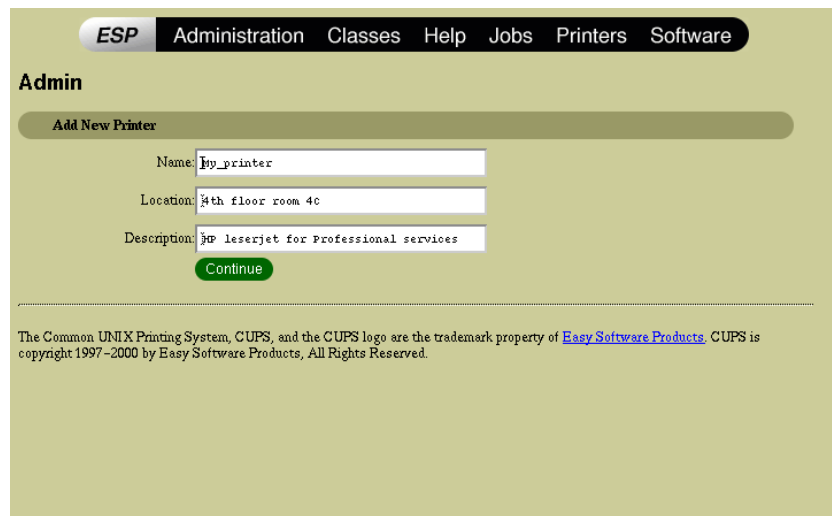
The first time you want to perform an administrative task with *cups*, it will ask you for the root password (figure 11-3). Simply provide the root login and password here.



Figure 11-3. The CUPS Login Dialog

11.1.2.1. Provide Informal Information About The Printer

This first form presents three fields you can fill-in at your convenience to help other users know which physical printer they deal with. The text has no influence on the printer's behavior, but fill them in carefully anyway, to save confusion later.



ESP Administration Classes Help Jobs Printers Software

Admin

Add New Printer

Name:

Location:

Description:

The Common UNIX Printing System, CUPS, and the CUPS logo are the trademark property of [Easy Software Products](#). CUPS is copyright 1997-2000 by Easy Software Products, All Rights Reserved.

Figure 11-4. Adding a New Printer, Step 1

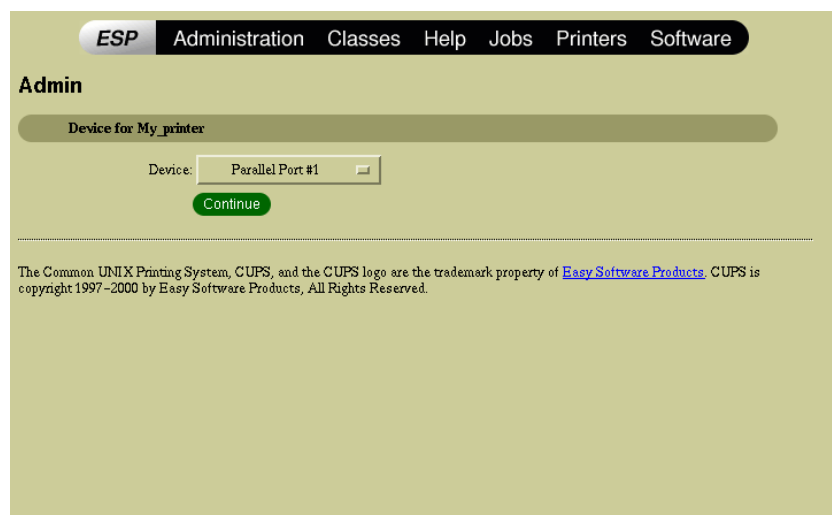
The printer's name is the only required field here.



Remember to plug in your printer and power it on, so that it will be automatically detected during the setup.

11.1.2.2. Tell Where The Printer Is Connected

You need to tell *cups* where the printer is physically located. For a printer directly connected to your computer, choose **Parallel Port**, **Serial Port**, or **USB** depending on the type of connection. Note that if you have a serial printer, you will need to install the package `cups-serial`. USB printers must also be connected and powered on.



ESP Administration Classes Help Jobs Printers Software

Admin

Device for My_printer

Device:

The Common UNIX Printing System, CUPS, and the CUPS logo are the trademark property of [Easy Software Products](#). CUPS is copyright 1997-2000 by Easy Software Products, All Rights Reserved.

Figure 11-5. Adding a New Printer, Step 2

Many other types of connections are available:

Appsocket/HP jetdirect

For printers directly connected to a local area network.

LPD/LPR

For printers that directly implement this type of behavior, or printers served by this type of queue. *UNIX* OSes, generally provide this type of connection.

Samba

For printers served by *Windows* servers. Note that to connect to that kind of printer, you need to install the *Samba* package.



We will not document those types of connection here, since their configuration is rather straight forward provided you have collected the required information previously.

11.1.2.3. Choose The Brand Name of Your Printer

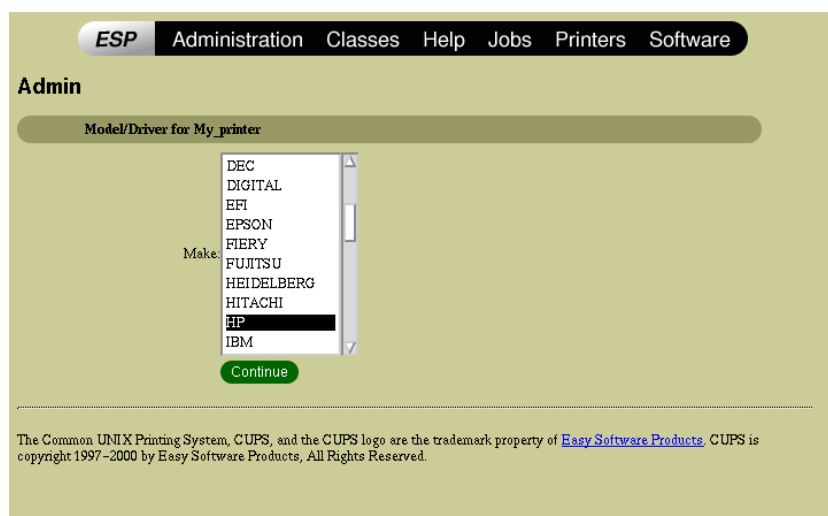


Figure 11-6. Adding a New Printer, Step 3

It is now time to tell *cups* which printer you are installing. You simply need to highlight the manufacturer's name in the list.

11.1.2.4. Choose Your Printer's Model

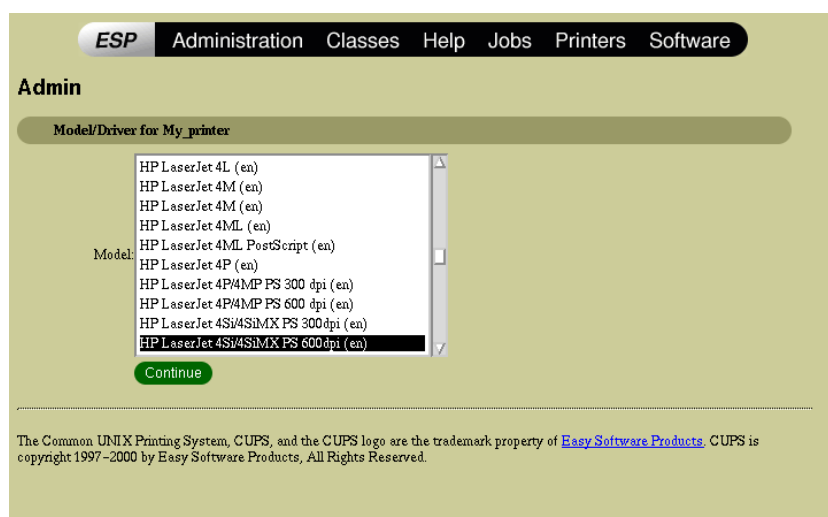


Figure 11-7. Adding a New Printer, Step 4

This is the final step, according to your previous choice, the list now shows all models from that specific manufacturer. Choose your model carefully.

If all goes well, you should now see your new printer in the **Printers** page.

11.1.2.5. Final Configuration And Test

Before testing the printer, you must ensure that the paper size configuration for that printer is correct. Go to the printer page, and click on **Configure Printer**. From the printer's parameters page, go to the **General** section and choose the appropriate **Page Size**. Some printers refuse to print if they do not have the appropriate paper loaded. For color printers it may also be necessary to select the appropriate color cartridge.



Concerning the parameters page: whenever you change a parameter in a section, you must click the corresponding **Continue** button in order to make your changes take effect.

11.1.3. A Note About Security

By default whenever you configure a printer on your machine, it becomes available for other people on your local area network. If you prefer that people shouldn't be able to print on your printer, you need to manually edit the *cups* configuration file: `/etc/cups/cupsd.conf`. You simply need to replace the line

```
#BrowseInterval 30
```

by

```
BrowseInterval 0
```

Also, this file contains a lot of options that allow you to finely tune your printer server. In particular, you can restrict access from specific machines or sub-networks. For more information consult the many comments within the configuration file or consult the online help from the web interface.



Whenever you make changes to the configuration file, do not forget to restart the *cups* daemon server by issuing:

```
service cups restart
```

as root.

11.1.4. Managing Print Jobs

This feature is mostly useful for very busy printers, but you may need it occasionally to cancel a wrong printout of 10,000 pages for example. When you send a job to a printer you can consult your jobs, and possibly all jobs from all users if you are the administrator for the machine hosting that printer, by displaying the page specific to the printer (figure 11-8).

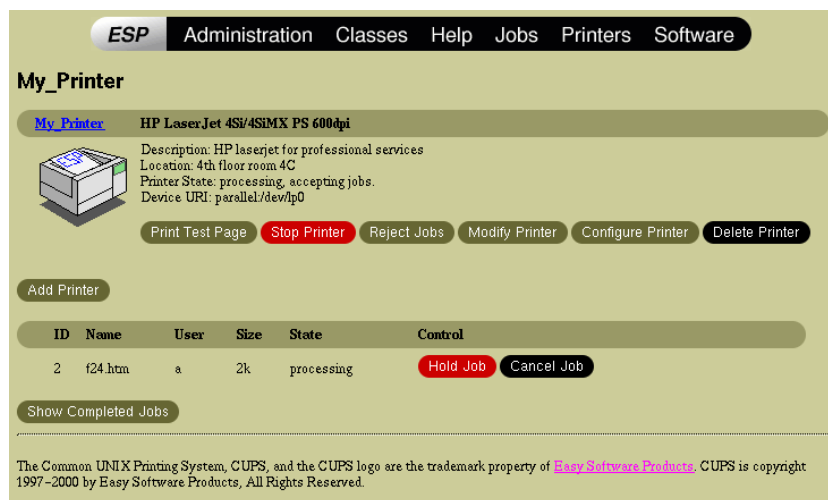


Figure 11-8. The Printer Status Page

You can then perform two different actions on a specific job:

- **Hold Job:** To put the job on a waiting list, it'll be printed only when you come back to this job and press the green **Release Job** button.
- **Cancel Job:** to definitely cancel this job (take it out of the queue).

If you wish to make your printer temporarily unavailable (to change the toner for example) you can simply click on the **Reject Jobs** button. Then when the printer is ready to accept jobs again, click on the **Accept Jobs** button.



If you are interested in job handling capabilities, please refer to the *Printing Manager* of the *KDE Control Center*.

11.2. Printing Documents

Replacing traditional printing commands `lpr` and `lp` for printing documents, there is now a nice application on **Mandrake Linux** called *XPP* that allows users to print files and configure printing parameters for all printouts. It can also be used as the “print command” in applications², so that you can comfortably access all printers from there too.

11.2.1. Just Print a File

Let's say that you have just stored on your hard drive an image from a web site and you wish to print it. First launch *XPP* from the menu **Applications+Publishing→X Printing Panel**, the main window (figure 11-9) will appear.

² You can also use for this purpose, the `kprinter` command

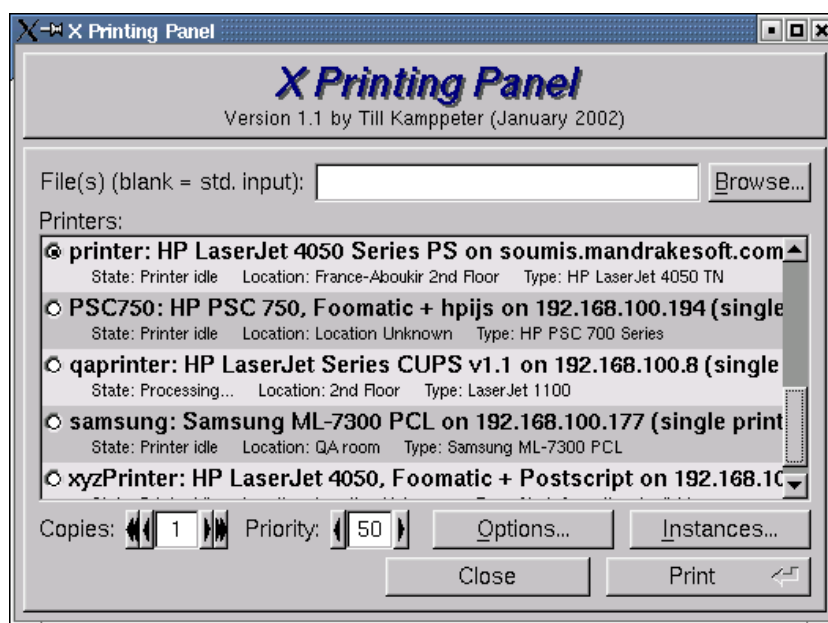


Figure 11-9. The XPP Main Window

There are three main parts on that window: a field to specify the location of the file(s) to print, the list of printers available and a set of options at the bottom.

To select the file to print, click on the **Browse...** button, it'll display a dialog (figure 11-10) where you can navigate through your disk and choose the file you wish to print. Note that if you know the location you can also type the full path by hand into the white field.

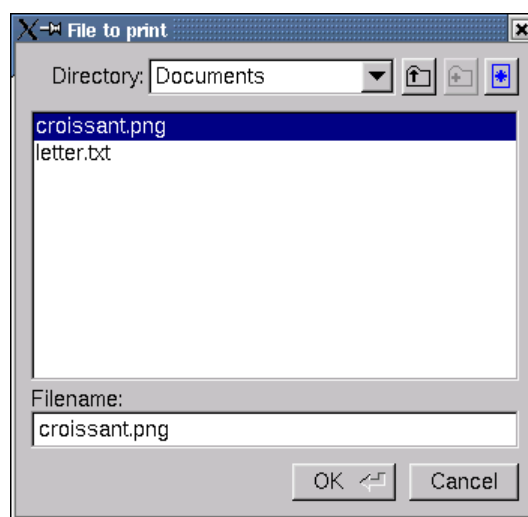


Figure 11-10. The XPP File Selection

Then make sure that the printer selected (with a black point on the left) is the correct one and click the **Print** button.



If you wish to use *XPP* as the printing program for print jobs sent from other applications such as *galeon* or many others that allow it, you simply need to change the printing command. There is usually a field in the printing options dialog that says *lpr*, just change it to *xpp*.

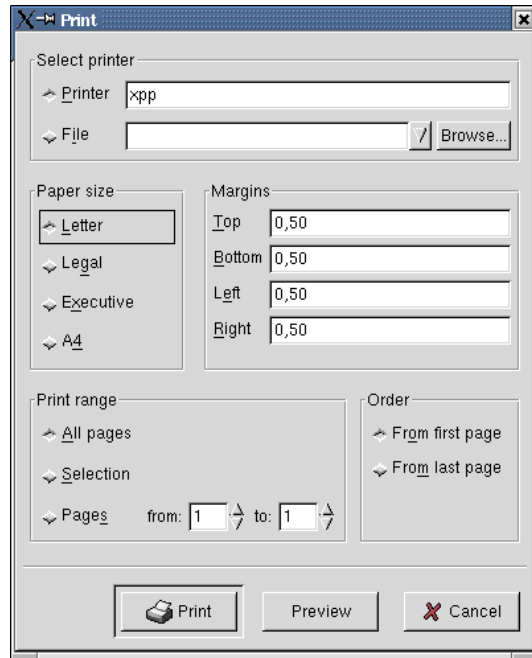


Figure 11-11. The Galeon Printing Dialog

Then when you click on the **OK** button, the *XPP* window will pop-up. You just need to change the options you wish, without worrying about the file.

11.2.2. Advanced Configuration

XPP allows you to finely tune your printouts. First, there are options on the main window that allow you to print multiple copies of the same file (**Copies** field) and to change the priority of your printing job. This latter is useful for very busy printers used by many users. If you urgently need a document to be printed, increment the priority number; if you wish to print a document you do not need immediately, lower the priority number.

Then there is the **Options...** button that displays a multi-tabs options dialog (figure 11-12).

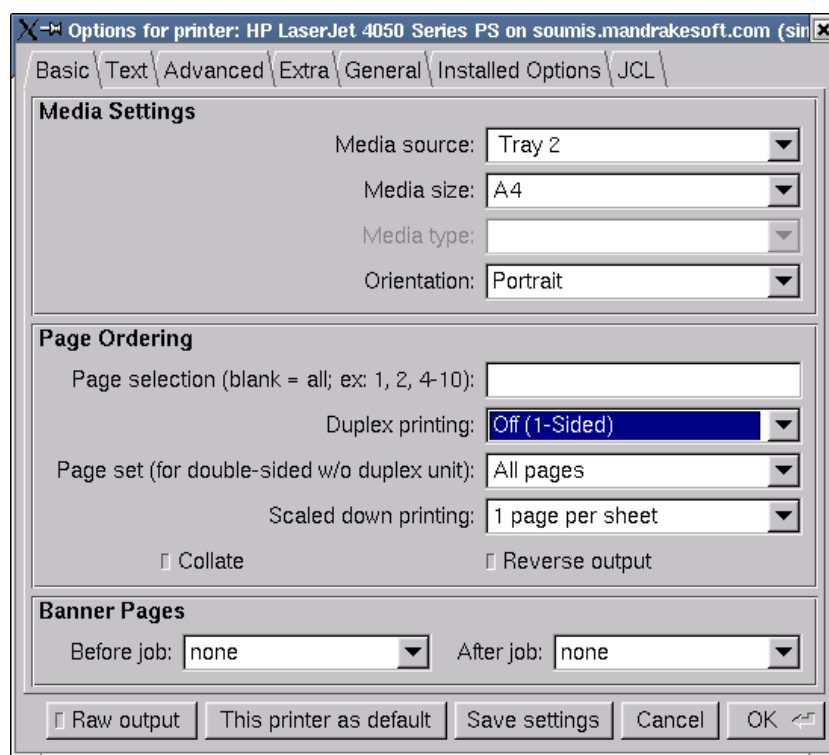


Figure 11-12. The XPP Basic Options Dialog

The first tab is divided in two sections:

Media Settings

You can choose three parameters here, although depending on the printer's capabilities, they may or may not be relevant. First is the option to choose a tray for paper, or to use manual feeding. Then the paper size, and finally the printing orientation.

Page Ordering

These options are very useful to define the way your document is printed and which part of it.

- The **Page selection** example would only print pages 1, 2, 4, 5, 6, 7, 8, 9, 10 of your document.
- The **Duplex Printing** option is only useful for printers that support it. It is also called double-sided printing.
- As stated, the **Page set** option allows you to duplex print even on printers that do not support it. Simply begin by printing the even pages, place the paper again in the tray (be careful with the orientation!) and print the odd pages on the other side.
- **Scaled down printing** is an environmentalist's option that allows printing multiple pages per sheet of paper. Combined with duplex printing, it should help to save the rain forest :-)
- The **Reverse output** option will print the pages beginning with the last one. It is useful for printers which stack the paper only face-up, so multiple-page documents come out in the correct order.
- Finally, the **Collate** option changes the page order when printing multiple copies of the same file. With the option set, the order of a 3 pages document will be 1,2,3,1,2,3. Without this setting, pages will come out in this order: 1,1,2,2,3,3.

The Text tab (figure 11-13) provides more subtle options to change the way text files are printed.

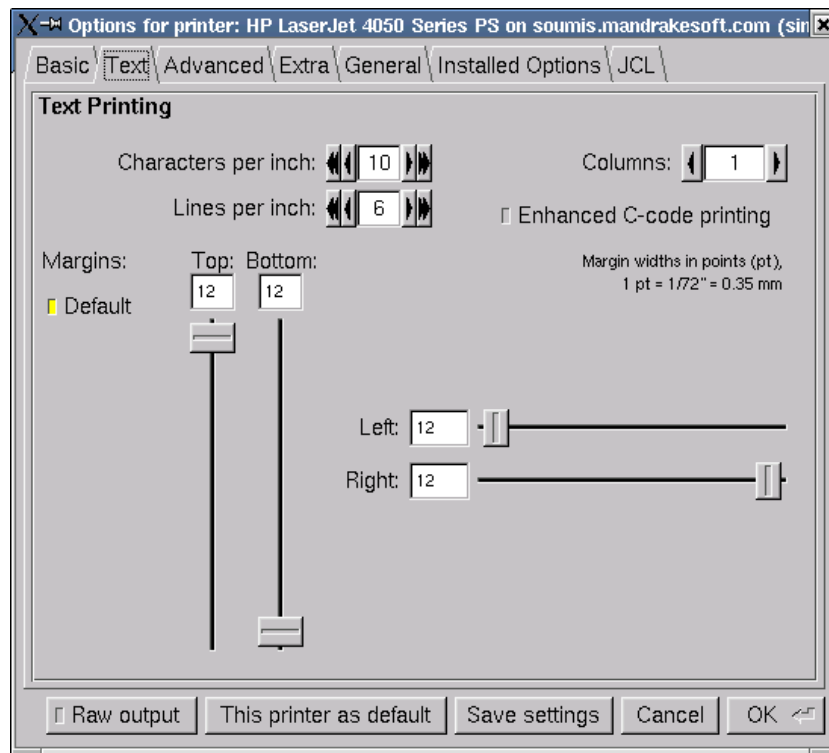


Figure 11-13. The XPP Text Options Dialog

There are many options, each one quite easy to understand. Just a note about the **Enhanced C-Code printing** option, that will print a header for each page and perform syntax highlighting for listings of *C* programs.

The **Advanced** tab (figure 11-14) provides options to change the page's visual appearance.

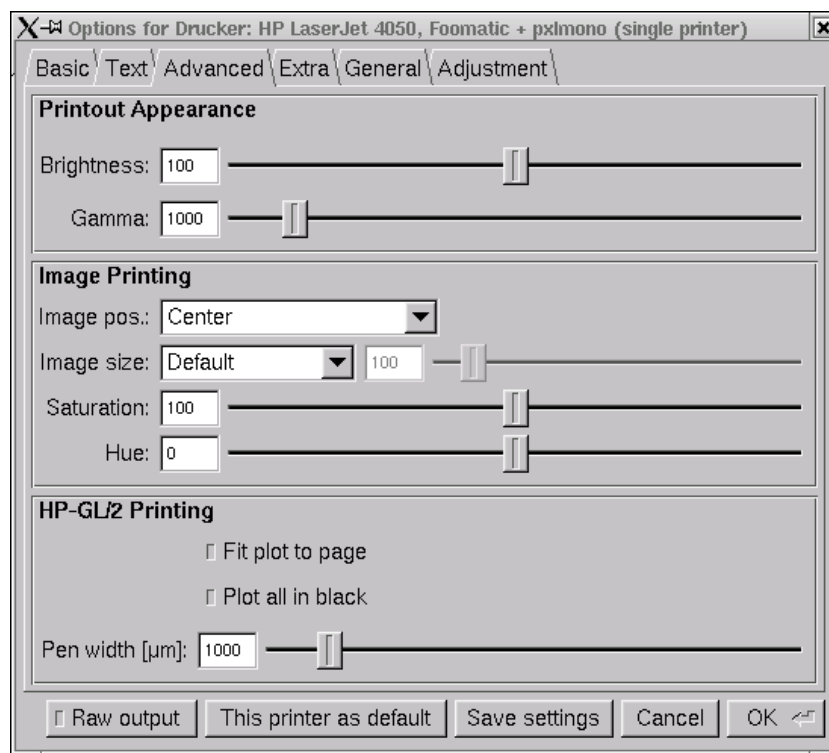


Figure 11-14. The XPP Advanced Options Dialog

This tab is divided in three sections:

Printout Appearance

Two parameters, **Brightness** and **Gamma** will mainly influence half-tone parts and images in the printout (lighten-darken).

Image Printing

The first two parameters change the relative position and size of the image on the page; the last two change the way colors are corrected.

HP-GL/2 Printing

Three parameters:

- Fit plot to page: Resize the plot so that it exactly fits the paper size.
- Plot all in black: Print in black & white only.
- Pen width: Adjust the width of the “virtual” pen plotting the document.

The following tabs present options specific to each printer: colors, resolution, dithering, etc.



When you choose options incompatible with each other, the problematic options are displayed in red so that you can easily identify the conflict. In this situation, you will get an error message when clicking on **Save settings** or **OK** - correct your settings first.

When changing the options, there are two possibilities: either save them with the **Save settings** button so that options are reused for next printouts, or simply click **OK** if these options are just valid for the current print job.

Chapter 12. msec – Mandrake Security Tools

12.1. Introducing msec

While *GNU/Linux* is being used for a very wide range of applications, from basic office work to high-availability servers, the need arose for different *security levels*. It is obvious that constraints inherent to highly secured servers do not match the needs of a secretary. On the other hand, a big public server is more sensitive to malicious people than my isolated *GNU/Linux* box.

It is with that aim that the *MSEC* package was designed. It is made of two parts:

- scripts that modify the whole system to lead it to one of the six security levels provided with *MSEC*. These levels range from very minimal security offering ease of use, to extreme configurations suitable for very sensitive applications and managed by experts;
- *cron* jobs, which will periodically check the integrity of the system according to security-level configuration, and possibly detect and warn you of a possible intrusion into the system, or of a security leak.

Note that the user may also customize his security level, adjusting parameters to his own needs.

12.2. Setting Your Security Level

MSEC is a base RPM. That means that it gets installed during the installation of your **Mandrake Linux** system. The installation created a *msec* directory into the */etc/security* directory, containing all that is needed to secure your system.

There is a graphical interface to *MSEC*, called *draksec*. It is available through *Control Center* and allows to change your system's security level. See the chapter *Setting Your Security Level* in the *User Guide*.

A command-line tool also exists. It allows better tuning. Log in as root and type *msec <x>*, *<x>* being the security level you want.

Note that whatever level you choose, your configuration will be stored in */var/lib/msec/security.conf*.

12.2.1. Level 0

This level is to be used with care. It makes your system easier to use, but extremely insecure. In particular, you should not use this security level if you answer "yes" to any of the following questions:

- is my computer connected to the Internet?
- is my computer connected to other computers through a network?
- will this computer be used by someone other than me?
- are there any private files on my computer that I don't want others to access?
- for lack of knowing *GNU/Linux* well enough, is it possible that I may harm the system?

As you can see, this security level should not be set by default, because huge data problems may occur.

12.2.2. Level 1

The main security improvement compared with level 0 is that the access to any user's data is granted via *username* and *password*. Therefore, it may be used by various people and it is less sensitive to mistakes. However, it should not be used on a computer which is connected to a modem or LAN (*Local Area Network*).

12.2.3. Level 2

Few major improvements for this security level; it mainly provides additional security warnings and checks. It is more secure for multi-user use.

12.2.4. Level 3

This is the standard security level, recommended for a computer which will be used to connect to the Internet as a client. Most of the security checks are periodically run, especially one (security check) which looks for open ports on the system. However, these open ports are kept opened and access to them is granted to everyone.

From the user's point of view, the system is now a little bit more closed, so he will need basic knowledge of the *GNU/Linux* system to achieve special operations. The security offered here is comparable with the one of a standard **Red Hat** or any previous **Mandrake Linux** distribution.

12.2.5. Level 4

With this security level, it is possible to use the system as a server. The security is now high enough and accept connections from many clients. By default, only connections from the computer itself will be granted. However, advanced services have been disabled, and the system administrator will have to activate the desired ones by hand in configuration files. He also will have to define for whom the access will be granted.

Security checks will warn the system administrator of possible system security holes or intrusions.

12.2.6. Level 5

We build on Level 4 features and now the system is entirely closed. Security features are at their maximum. The system administrator must activate ports and grant connections to give other computers access to services offered by this machine.

12.3. Security Levels Features

What follows is the description of the different security features brought by each level. These features are of two types: system hardening which changes the system, and checks run periodically which do not.

12.3.1. Hardening System

The hardening system type changes the permissions, owners, groups of the files and directories on the system according to the security level.

It also changes the configuration files to respect the security level defined and re-launch the needed programs to take the changes into account. The hardening system type is run every hour. All the changes are logged to *syslog*.

Feature \ Level	0	1	2	3	4	5
<i>umask</i> for users	002	002	002	002	077	077
<i>umask</i> for root	002	002	002	002	002	077
access without password	yes					
authorized to connect to X server	all	local	local	none	none	none
X server listens to connections from	all	all	all	all	local	local
shell timeout	none	none	none	none	3600	900
su only from members of the wheel group						yes
shell history size					10	10
sulogin in runlevel 1					yes	yes
forbid user list in the display managers					yes	yes
ignore ICMP echo					yes	yes
ignore bogus ICMP error message					yes	yes
direct root login	yes	yes	yes	yes		
enable libsafe					yes	yes
forbid at and crontab for the users					yes	yes

Feature \ Level	0	1	2	3	4	5
password aging (days)					60	30
forbid <i>autologin</i>				yes	yes	yes
allow issues (text login prompts)	all	all	all	local	local	none
. in \$PATH	yes	yes				
warnings in file /var/log/security.log		yes	yes	yes	yes	yes
warnings directly on <i>tty</i>			yes	yes	yes	yes
warnings in syslog			yes	yes	yes	yes
warnings sent by e-mail to			yes	yes	yes	yes
all system events additionally logged to /dev/tty12				yes	yes	yes
only root can Ctrl-Alt-Del					yes	yes
unknown services are disabled					yes	yes
grants connection from	all	all	all	all	local	none

12.3.1.1. umask For Users

Simply sets the *umask* for normal users to the value corresponding to the security level.

12.3.1.2. umask For Root

Same as above, but for root.

12.3.1.3. Access Without Password

Access to the accounts are granted without asking for a password.

12.3.1.4. Authorization to Connect to The X Server

1. all: everybody from everywhere can open an *X* window on your screen.
2. local: only people connected at *localhost* may open an *X* window on your screen.
3. none: nobody can.

12.3.1.5. X Server Listens to Connections From...

1. all: the *X* server listens to connections from tcp.
2. local: the *X* server listens to connections from the *UNIX* socket.

12.3.1.6. Shell Timeout

If the user is inactive for a certain amount of seconds, the shell exits.

12.3.1.7. su Only From Members of The Wheel Group

Only members of the wheel group are allowed to use *su* to take the root identity.

12.3.1.8. Shell History Size

The number of commands saved at the end of a shell run.

12.3.1.9. sulogin in Runlevel 1

The `sulogin` is used to protect access to a single user's initlevel (you have to enter the root password to have access).

12.3.1.10. Forbid User List in The Display Managers

Forbid the display of the list of users on the system in *KDM* and *GDM*.

12.3.1.11. Ignore ICMP Echo

Don't answer to ping requests.

12.3.1.12. Ignore Bogus ICMP Error Message

Don't handle bogus ICMP error message.

12.3.1.13. Direct Root Login

Allow direct root login without using the `su` program.

12.3.1.14. Enable libsafe

Enable libsafe protection (only activated if the libsafe package is installed).

12.3.1.15. Forbid at And crontab For The Users

Users cannot use the `at` and `crontab` programs. However, they can be used by the root account. If you want to authorize only some users, add them in `/etc/at.allow` and `/etc/cron.allow` respectively.

12.3.1.16. Password Aging

The passwords expire after a fixed amount of days; the users need to change them before the expiration date if they don't want to see their accounts deactivated.

12.3.1.17. Forbid autologin

The *autologin* program is forbidden.

12.3.1.18. allow Issues

If set to `none`, no banners are displayed. If set to `local`, only a login banner is displayed on the console. If set to `all`, a login banner is displayed on all login prompts.

12.3.1.19. . in \$PATH

The `.` entry is added to the `$PATH` environment variable, allowing easy execution of programs within the current working directory (it is also, to some extent, a security hole).

12.3.1.20. Warnings in The security.log File

Each warning which the daily check comes upon is logged into the `/var/log/security.log` file.

12.3.1.21. Warnings Directly on tty

Each warning issued by the daily check is directly printed to the root user's console.

12.3.1.22. Warnings in syslog

Warnings issued during the daily check are directed to the *syslog* service.

12.3.1.23. Warnings Sent by E-mail to Root

Warnings issued by the daily check are also sent by e-mail to root.

12.3.1.24. Unknown Services Are Disabled

During the installation of a package, the services are added through `chkconfig --add <service>` **only** if the name of the service appears in the `/etc/security/msec/server` file.

12.3.1.25. Grants Connection To...

1. `all`: all computers are allowed to connect to open ports.
2. `local`: only the *localhost* is allowed to connect to open ports.
3. `none`: no computers are allowed to connect to open ports.

This protection is granted by the *tcp wrappers* package. If you want to grant access to a service while none are allowed, use the `/etc/hosts.allow` file. For example, if you want to grant *ssh* connections from everywhere, add the following line to `/etc/hosts.allow`:

```
sshd: ALL
```

12.3.2. Periodic Checks

If the security level is greater than 0, the checks are run every night.

Feature \ Level	0	1	2	3	4	5
global security check		yes	yes	yes	yes	yes
<i>suid</i> root files check			yes	yes	yes	yes
<i>suid</i> root files MD5 check			yes	yes	yes	yes
writable files check			yes	yes	yes	yes
permissions check				yes	yes	yes
<i>suid</i> group files check			yes	yes	yes	yes
unowned files check					yes	yes
promiscuous check					yes	yes
listening port check				yes	yes	yes
<code>passwd</code> file integrity check				yes	yes	yes
<code>shadow</code> file integrity check				yes	yes	yes
integrity check from the RPM database				yes	yes	yes

Note that seven out of the twelve periodic checks can detect changes on the system. They store the configuration of the system during the last check (one day ago) and warn you of any changes that occurred in the

meantime in files located in the `/var/log/security/` directory. These checks are:

- *suid* root files check
- *suid* root files MD5 check
- writable files check
- *sgid* files check
- unowned files check
- listening port check
- integrity check from the RPM database

12.3.2.1. Global Security Check

1. “NFS filesystems globally exported”: this is regarded as insecure as there is no restriction as to who may mount these filesystems.
2. “NFS mounts with missing nosuid”: these filesystems are exported without the *nosuid* option, which forbids *suid* programs to work on the machine.
3. “Host trusting files contain + sign”: that means that one of the following files: `/etc/hosts.equiv`, `/etc/shosts.equiv`, `/etc/hosts.lpd` contains hosts allowed to connect without proper authentication.
4. “Executables found in the aliases files”: it issues a warning, naming the executables run through the two `/etc/aliases` and `/etc/postfix/aliases` files.

12.3.2.2. suid root Files Check

Checks for new or removed *suid* root files on the system. If such files are found, a list of these files are issued as a warning.

12.3.2.3. suid root File MD5 Check

Checks the MD5 signature of each *suid* root file on the system. If the signature has changed, it means that a modification has been made to this program, possibly a back-door. A warning is then issued.

12.3.2.4. Writable Files Check

Check whether files are world-writable on the system. If so, issues a warning containing the list of these naughty files.

12.3.2.5. Permissions Check

This one checks permissions for some special files such as `.netrc` or users’ configuration files. It also checks permissions of users’ home directories. If their permissions are too loose or the owners unusual, it issues a warning.

12.3.2.6. sgid Files Check

Check for new or removed *sgid* files on the system. If such files are found, a list of these files are issued as a warning.

12.3.2.7. Unowned Files Check

This check searches for files owned by users or groups not known by the system. If such files are found, the owner is automatically changed to the `nobody.nogroup` user/group.

12.3.2.8. Promiscuous Check

This test checks every *Ethernet* card to determine whether they are in “promiscuous” mode. This mode allows the card to intercept every packet received by the card, even those that are not directed to it. It may mean that a *sniffer* is running on your machine. **Note that this check is set up to be run every minute.**

12.3.2.9. Listening Port Check

Issues a warning with all listening ports.

12.3.2.10. passwd File Integrity Check

Verifies that each user has a password (not a blank or an easy-to-guess password) and checks that it is shadowed.

12.3.2.11. shadow File Integrity Check

Verifies that each user in the shadow file has a password (not a blank one).

12.3.2.12. Integrity Check From The RPM Database

Verifies that no file from installed packages has changed and verifies that no package has been installed/removed/updated since last run.

12.4. Customization**12.4.1. Filesystem Changes**

The changes on the filesystem are stored in the `/var/lib/msec/perm.<level>`. This can be modified by using the `/etc/security/msec/perm.local` with the same format as the file in `/var/lib/msec/perm.*`

12.4.2. Config Files Changes

The config file changes can be modified by using the `/etc/security/msec/level.local`. See the `msec lib` man page for more details.

12.4.3. Checks Changes

The current checks configuration is stored under `/var/lib/msec/security.conf`. These settings can be overridden in `/etc/security/msec/security.conf` with the same syntax. For example, the `MAIL_USER` can be used to change the e-mail address used to send the daily mail report.

Chapter 13. Building and installing free software

I am often asked how to install free software from sources. Compiling software yourself is really easy because most of the steps to follow are the same no matter what the software to install is. The aim of this document is to guide the beginner step by step and explain to him the meaning of each move. I assume that the reader has a minimal knowledge of the *UNIX* system (*ls* or *mkdir* for instance).

This guide is only a guide, not a reference manual. That is why several links are given at the end to answer any remaining questions. This guide can probably be improved, so I appreciate receiving any remarks or corrections on its contents.

13.1. Introduction

What makes the difference between free software and proprietary software is the access to the sources of the software¹. That means free software is distributed as archives of source files. It may be unfamiliar to beginners because users of free software must compile sources by themselves before they can use the software.

There are compiled versions of most of the existing free software. The user in a hurry just has to install these pre-compiled binaries. Some free software is not distributed under this form, or the earlier versions are not yet distributed under binary form. Furthermore, if you use an exotic operating system or an exotic architecture, a lot of software will not be compiled for you. More importantly, compiling software by yourself allows you to enable only the interesting options or to extend the functionality of the software by adding extensions in order to obtain a program that exactly fits your needs.

13.1.1. Requirements

To build software, you need:

- a computer with a working operating system,
- general knowledge of the operating system you use,
- some space on your disk,
- a compiler (usually for the *C* language) and an archiver (*tar*),
- some food (in difficult cases, it may last a long time). A real hacker eats pizzas - not quiches.
- something to drink (for the same reason). A real hacker drinks soda – for caffeine.
- the phone number of your techie friend who recompiles his kernel each week,
- especially patience and a lot of it!

Compiling from sources does not generally present a lot of problems, but if you are not used to it the smallest snag can throw you. The aim of this document is to show you how to escape from such a situation.

13.1.2. Compilation

13.1.2.1. Principle

In order to translate a source code into a binary file, a *compilation* must be done (usually from *C* or *C++* sources, which are the most widespread languages among the (*UNIX*) free software community). Some free software is written in languages which do not require compilation (for instance *perl* or the *shell*, but they still require some configuration).

C compilation is logically done by a *C* compiler, usually *gcc*, the free compiler written by the GNU project (<http://www.gnu.org/>). Compiling a complete software package is a complex task, which goes through the successive compilations of different source files (it is easier for various reasons for the programmer to put the different parts of his work in separate files). In order to make it easier on you, these repetitive operations are handled by a utility named *make*.

1. This is not completely true since some proprietary software also provides source code. But unlike what happens with free software, the final user is not allowed to use or modify the code as he wants.

13.1.2.2. The four steps of compilation

To understand how compilation works (in order to be able to solve possible problems), you have to know the four steps. The object is to little by little convert a textfile written in a language that is comprehensible to a trained human being (i.e. *C* language), into a language that is comprehensible to a machine (or a **very** well trained human being in a few cases). `gcc` executes four programs one after the other, each of which takes on one step:

1. `cpp`: The first step consists of replacing directives (*preprocessors*) by pure *C* instructions. Typically, this means inserting a header (`#include`) or defining a macro (`#define`). At the end of this stage, pure *C* code is generated.
2. `cc1`: This step consists in converting *C* into *assembly language*. The generated code depends on the target architecture.
3. `as`: This step consists of generating *object code* (or binary code) from the assembly language. At the end of this stage, a `.o` file is generated.
4. `ld`: The last step (*linkage*) links all the object files (`.o`) and the associated libraries, and produces an executable file.

13.1.3. Structure of a distribution

A correctly structured free software distribution always has the same organization:

- An `INSTALL` file, which describes the installation procedure.
- A `README` file, which contains general information related to the program (short description, author, URL where to fetch it, related documentation, useful links, etc). If the `INSTALL` file is missing, the `README` file usually contains a brief installation procedure.
- A `COPYING` file, which contains the license or describes the distribution conditions of the software. Sometimes a `LICENSE` file replaces it.
- A `CONTRIB` or `CREDITS` file, which contains a list of people related to the software (active participation, pertinent comments, third-party programs, etc).
- A `CHANGES` file (or less frequently, a `NEWS` file), which contains recent improvements and bugfixes.
- A `Makefile` file (see the section *make*, page 87), which allows compilation of the software (it is a necessary file for `make`). This file often does not exist at the beginning and is generated during configuration process.
- Quite often, a `configure` or `Imakefile` file, which allows one to generate a new file `Makefile`,
- A directory that contains the sources, and where the binary file is usually stored at the end of the compilation. Its name is often `src`.
- A directory that contains the documentation related to the program (usually in `man` or *Texinfo* format), whose name is often `doc`.
- Sometimes, a directory that contains data specific to the software (typically configuration files, example of produced data, or resources files).

13.2. Decompression

13.2.1. `tar.gz` archive

The standard² compression format under *UNIX* systems is the `gzip` format, developed by the GNU project, and considered as one of the best general compression tools.

`gzip` is often associated with a utility named `tar`. `tar` and is a survivor of antediluvian times, when computerists stored their data on tapes. Nowadays, floppy disks and CD-ROM have replaced tapes, but `tar` is still

2. More and more a new program, called `bzip2`, more efficient on text files (and requiring more computing power) is being used. See the later section *bzip2*, page 83 which deals specifically with this program.

being used to create archives. All the files in a directory can be appended in a single file for instance. This file can then be easily compressed with gzip.

This is the reason why much free software is available as tar archives, compressed with gzip. So, their extensions are .tar.gz (or also .tgz to shorten).

13.2.2. The use of GNU Tar

To decompress this archive, gzip and then tar can be used. But the GNU version of tar (gtar) allows us to use gzip *"on-the-fly"*, and to decompress an archive file without noticing each step (and without the need for the extra disk space).

The use of tar follows this format:

```
tar <file options> <.tar.gz file> [files]
```

The <files> option is not required. If it is omitted, processing will be made on the whole archive. This argument does not need to be specified to extract the contents of a .tar.gz archive.

For instance:

```
$ tar xvfz guile-1.3.tar.gz
-rw-r--r-- 442/1002      10555 1998-10-20 07:31 guile-1.3/Makefile.in
-rw-rw-rw- 442/1002      6668 1998-10-20 06:59 guile-1.3/README
-rw-rw-rw- 442/1002      2283 1998-02-01 22:05 guile-1.3/AUTHORS
-rw-rw-rw- 442/1002     17989 1997-05-27 00:36 guile-1.3/COPYING
-rw-rw-rw- 442/1002     28545 1998-10-20 07:05 guile-1.3/ChangeLog
-rw-rw-rw- 442/1002      9364 1997-10-25 08:34 guile-1.3/INSTALL
-rw-rw-rw- 442/1002      1223 1998-10-20 06:34 guile-1.3/Makefile.am
-rw-rw-rw- 442/1002     98432 1998-10-20 07:30 guile-1.3/NEWS
-rw-rw-rw- 442/1002      1388 1998-10-20 06:19 guile-1.3/THANKS
-rw-rw-rw- 442/1002      1151 1998-08-16 21:45 guile-1.3/TODO
...
```

Among the options of tar:

- **v** makes tar verbose. This means it will display all the files it finds in the archive on the screen. If this option is omitted, the processing will be silent.
- **f** is a required option. Without it, tar tries to use a tape instead of an archive file (i.e., the /dev/rmt0 device).
- **z** allows you to process a "gzipped" archive (with a .gz extension). If this option is forgotten, tar will produce an error. Conversely, this option must not be used with an uncompressed archive.

tar allows you to perform several actions on an archive (extract, read, create, add...). An option defines which action is used:

- **x**: allows you to extract files from the archive.
- **t**: lists the contents of the archive.
- **c**: allows you to create an archive. You may use it to backup your personal files, for instance.
- **r**: allows you to add files at the end of the archive. It cannot be used with a compressed archive.

13.2.3. bzip2

A compression format named bzip2 has begun to replace gzip in general use. bzip2 produces smaller archives than gzip does, but is not yet a standard. It is only on recently made archives that the .tar.bz2 extensions would be found.

bzip2 is used like gzip by means of the tar command. The only change is to replace the letter z by the letter j. For instance:

```
$ tar xvjf foo.tar.bz2
```

Some distributions may still use the option I instead:

```
$ tar xvfI foo.tar.bz2
```

Another way (which seems to be more portable, but is longer to type!):

```
$ tar --use-compress-program=bzip2 -xvf foo.tar.bz2
```

bzip2 must be installed and included in your PATH environment variable before you run tar.

13.2.4. Just do it!

13.2.4.1. The easiest way

Now that you are ready to decompress the archive, do not forget to do it as administrator (root). You will need to do things that a ordinary user is not allowed to do, and even if you can perform some of them as a regular user, it is simpler to just be root the whole time.

The first step is to be in the `/usr/local/src` directory and copy the archive there. You should then always be able to find the archive if you lose the installed software. If you do not have a lot of space on your disk, save the archive on a floppy disk after having installed the software. You can also delete it but be sure that you can find it on the *Web* whenever you need it.

Normally, decompressing a tar archive should create a new directory (you can check that beforehand thanks to the `t` option). Go then in that directory, you are now ready to proceed further.

13.2.4.2. The safest way

UNIX systems (of which *GNU/Linux* and *FreeBSD* are examples) can be secure systems. That means that normal users cannot perform operations that may endanger the system (format a disk, for instance) or alter other users' files. It also immunizes the system against viruses.

On the other hand, root can do everything - even running a malicious program. Having the source code available allows you to examine it for malicious code (viruses or Trojans). It is better to be cautious in this regard³.

The idea is to create a user dedicated to administration (`free` or `admin` for example) by using the `adduser` command. This user must be allowed to write in the following directories: `/usr/local/src`, `/usr/local/bin` and `/usr/local/lib`, as well as all the sub-tree of `/usr/share/man` (he also may need to be able to copy files elsewhere). I recommend that you make this user owner of the necessary directories or create a group for him and make the directories writable for the group.

Once these precautions are taken, you can follow the instructions in the section *The easiest way*, page 84.

13.3. Configuration

For purely technical interest, the fact that authors create the sources is for the *porting* of the software. Free software developed for a *UNIX* system may be used on all of the existing *UNIX* systems (whether they are free or proprietary), with some changes. That requires configuration of the software just before compiling it.

Several configuration systems exist. You have to use the one the author of the software wants (sometimes, several are needed). Usually, you can:

- Use *AutoConf* (see the section *AutoConf*, page 84) if a file named `configure` exists in the parent directory of the distribution.
- Use *imake* (see the section *imake*, page 86) if a file named `Imakefile` exists in the parent directory of the distribution.
- Run a shell *script* (for instance `install.sh`) according to the contents of the `INSTALL` file (or the `README` file)

3. A proverb from the BSD world says: "Never trust a package you don't have the sources for."

13.3.1. AutoConf

13.3.1.1. Principle

AutoConf is used to correctly configure software. It creates the files required by the compilation (*Makefile* for instance), and sometimes directly changes the sources (for instance by using a *config.h.in* file).

The principle of *AutoConf* is simple:

- The programmer of the software knows which tests are required to configure his software (eg: “which version of this *library* do you use?”). He writes them in a file named *configure.in*, following a precise syntax.
- He executes *AutoConf*, which generates a configuration script named *configure* from the *configure.in* file. This script makes the tests required when the program is configured.
- The end-user runs the script, and *AutoConf* configures everything that is needed by the compilation.

13.3.1.2. Example

An example of the use of *AutoConf*:

```
$ ./configure
loading cache ./config.cache
checking for gcc... gcc
checking whether the C compiler (gcc ) works... yes
checking whether the C compiler (gcc ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for main in -lX11... yes
checking for main in -lXpm... yes
checking for main in -lguile... yes
checking for main in -lm... yes
checking for main in -lncurses... yes
checking how to run the C preprocessor... gcc -E
checking for X... libraries /usr/X11R6/lib, headers /usr/X11R6/include
checking for ANSI C header files... yes
checking for unistd.h... yes
checking for working const... yes
updating cache ./config.cache
creating ./config.status
creating lib/Makefile
creating src/Makefile
creating Makefile
```

To have better control of what *configure* generates, some options may be added by the way of the command line or environment variables. Example:

```
$ ./configure --with-gcc --prefix=/opt/GNU
```

or (with *bash*):

```
$ export CC='which gcc'
$ export CFLAGS=-O2
$ ./configure --with-gcc
```

or:

```
$ CC=gcc CFLAGS=-O2 ./configure
```

13.3.1.3. What if... it does not work?

Typically, it is an error that looks like `configure: error: Cannot find library guile` (most of the errors of the `configure` script look like this).

This means that the `configure` script was not able to find a library (the `guile` library in the example). The principle is that the `configure` script compiles a short test program, which uses this library. If it does not succeed in compiling this program, it will not be able to compile the software. Then an error occurs.

- Look for the reason of the error by looking at the end of the `config.log` file, which contains a track of all the steps of the configuration. The `C` compiler is clear enough with its error messages. This will usually help you in solving the problem.
- Check that the said library is properly installed. If not, install it (from the sources or a compiled binary file) and run `configure` again. An efficient way to check it is to search for the file that contains the symbols of the library; which is always `lib<name>.so`. For instance,

```
$ find / -name 'libguile*'

```

or else:

```
$ locate libguile

```

- Check that the library is accessible by the compiler. That means it is in a directory among: `/usr/lib` , `/lib` , `/usr/X11R6/lib` (or among those specified by the environment variable `LD_LIBRARY_PATH`, explained *What if... it does not work?*, page 88 number 5.b. Check that this file is a library by typing `file libguile.so`.
- Check that the headers corresponding to the library are installed in the right place (usually `/usr/include` or `/usr/local/include` or `/usr/X11R6/include`). If you do not know which headers you need, check that you have installed the development version of the required library (for instance, `gtk+-devel` instead of `libgtk`). The development version of the library provides the “include” files necessary for the compilation of software using this library.
- Check that you have enough space on your disk (the `configure` script needs some space for temporary files). Use the command `df -k` to display the partitions of your system, and note the full or nearly full partitions.

If you do not understand the error message stored in the `config.log` file, do not hesitate to ask for help from the free software community (see section *Technical support*, page 93).

Furthermore, check whether `configure` answers by 100% of No or whether it answers No while you are sure that a library exists. For instance, it would be very strange that there is no curses library on your system). In that case, the `LD_LIBRARY_PATH` variable is probably wrong!

13.3.2. imake

imake allows you to configure free software by creating a `Makefile` file from simple rules. These rules determine which files need to be compiled to build the binary file, and *imake* generates the corresponding `Makefile`. These rules are specified in a file named `Imakefile`.

The interesting thing about *imake* is that it uses information that is *site* (architecture-dependent). It is quite handy for applications using *X Window System*. But *imake* is also used for many other applications.

The easiest use of *imake* is to go into the main directory of the decompressed archive, and then to run the `xmkmf` script, which calls the *imake* program:

```
$ xmkmf -a
imake -DUseInstalled -I/usr/X11R6/lib/X11/config
make Makefiles

```

If the site is not correctly installed, recompile and install *X11R6*!

13.3.3. Various shell scripts

Read the `INSTALL` or `README` files for more information. Usually, you have to run a file called `install.sh` or `configure.sh`. Then, either the installation script is non-interactive (and determines itself what it needs) or it asks you information on your system (paths, for instance).

If you can not manage to determine the file you must run, you can type `./` (under *bash*), and then press twice the **TAB** key (tabulation key). *bash* automatically (in its default configuration) completes by a possible executable file from the directory (and therefore, a possible configuration script). If several files may be executed, it gives you a list. You just have to choose the right file.

A particular case is the installation of *perl* modules (but not only). The installation of such modules is made by the execution of a configuration script, which is written in *perl*. The command to execute is usually:

```
$ perl Makefile.PL
```

13.3.4. Alternatives

Some free software distributions are badly organized, especially during the first stages of development (but the user is warned!). They sometimes require you to change “by hand” some configuration files. Usually, these files are a `Makefile` file (see section *make*, page 87) and a `config.h` file (this name is only conventional).

I advise against these manipulations except for users who really do know what they are doing. This requires real knowledge and some motivation to succeed, but practice makes perfect.

13.4. Compilation

Now that the software is correctly configured, all that remains is for it to be compiled. This stage is usually easy, and does not pose serious problems.

13.4.1. make

The favorite tool of the free software community to compile sources is *make*. It has two advantages:

- The developer saves time, because it allows him to efficiently manage the compilation of his project,
- The end-user can compile and install the software in a few command lines, even if he has no preliminary knowledge of development.

Actions that must be executed to obtain a compiled version of the sources are stored in a file often named `Makefile` or `GNUMakefile`. Actually, when *make* is called, it reads this file – if it exists – in the current directory. If not, the file may be specified by using the option `-f` with *make*.

13.4.2. Rules

make operates in accordance with a system of *dependencies*, so compiling a binary file (“*target*”) requires going through several stages (“dependencies”). For instance, to create the (imaginary) `g1loq` binary file, the `main.o` and `init.o` object files (intermediate files of the compilation) must be compiled and then linked. These object files are also targets, whose dependencies are the source files.

This text is only a minimal introduction to survive in the merciless world of *make*. If you want to learn more, I advise you to go to the web site of **APRIL** (<http://www.april.org/groupe/doc/>), where you can find more detailed documentation about *make*. For an exhaustive documentation, refer to **Managing Projects with Make**, 2nd edition, O'Reilly, by Andrew Oram and Steve Talbott.

13.4.3. Go, go, go!

Usually, the use of `make` follows several conventions. For instance:

- `make` without argument just executes the compilation of the program, without installation.
- `make install` compiles the program (but not always), and then installs the required files at the right place in the file system. Some files are not always correctly installed (`man`, `info`), they might have been copied by the user himself. Sometimes, `make install` has to be executed again in sub-directories. Usually, this happens with modules developed by third parties.
- `make clean` clears all the temporary files created by the compilation, and also the executable file in most cases.

The first stage is to compile the program, and therefore to type (imaginary example):

```
$ make
gcc -c glloq.c -o glloq.o
gcc -c init.c -o init.o
gcc -c main.c -o main.o
gcc -lgtk -lgdk -glib -lXext -lX11 -lm glloq.o init.o main.o -o glloq
```

Excellent, the binary file is correctly compiled. We are ready to go to the next stage, which is the installation of the files of the distribution (binary files, data files, etc). See section *Installation*, page 92.

13.4.4. Explanations

If you are curious enough to look in the `Makefile` file, you will find known commands (`rm`, `mv`, `cp`, etc), but also strange strings, looking like `$(CFLAGS)`.

They are *variables* which are strings that are usually set at the beginning of the `Makefile` file, and then replaced by the value they are associated with. It is quite useful when you want to use the same compilation options several times in a row.

For instance, to print the string “foo” on the screen using `make all`:

```
TEST = foo
all:
    echo $(TEST)
```

Most of the time, the following variables are set:

1. `CC`: This is the compiler. Usually, it is `cc`, which is in most of free systems synonymous with `gcc`. When in doubt, put here `gcc`.
2. `LD`: This is the program used to ensure the final compilation stage (see section *The four steps of compilation*, page 81). By default, this is `ld`.
3. `CFLAGS`: These are the additional arguments that are given to the compiler during the first compilation stages. Among them:
 - `-I<path>`: Specifies to the compiler where to search some additional headers (eg: `-I/usr/X11R6/include` allows inclusion of the header files that are in directory `/usr/X11R6/include`).
 - `-D<symbol>`: Defines an additional symbol, useful for programs whose compilation depends on the defined symbols (ex: use the `string.h` file if `HAVE_STRING_H` is defined).

There are often compilation lines like:

```
$(CC) $(CFLAGS) -c foo.c -o foo.o
```

4. `LDFLAGS` (or `LFLAGS`): These are arguments used during the final compilation stage. Among them:
 - `-L<path>`: Specifies an additional path to search for libraries (eg: `-L/usr/X11R6/lib`).
 - `-l<library>`: Specifies an additional library to use during the final compilation stage.

13.4.5. What if... it does not work?

Do not panic, it can happen to anyone. Among the most common causes:

1. `gllq.c:16: decl.h: No such file or directory`

The compiler did not manage to find the corresponding header. Yet, the software configuration step should have anticipated this error. Here is how to solve this problem:

- Check that the header really exists on the disk in one of the following directories: `/usr/include`, `/usr/local/include`, `/usr/X11R6/include` or one of their sub-directories. If not, look for it on the whole disk (with `find` or `locate`), and if you still do not find it, check that you have installed the library corresponding to this header. You can find examples of the `find` and `locate` commands in their respective manual pages.
- Check that the header is really readable (type `less <path>/<file>.h` to test this)
- If it is in a directory like `/usr/local/include` or `/usr/X11R6/include`, you sometimes have to add a new argument to the compiler. Open the corresponding `Makefile` (be careful to open the right file, those in the directory where the compilation fails ⁴) with your favorite text editor (*Emacs*, *Vi*, etc). Look for the faulty line, and add the string `-I<path>` – where `<path>` is the path where the header in question can be found – just after the call of the compiler (`gcc`, or sometimes `$(CC)`). If you do not know where to add this option, add it at the beginning of the file, after `CFLAGS=<something>` or after `CC=<something>`.
- Launch `make` again, and if it still does not work, check that this option (see the previous point) is added during compilation on the faulty line.
- If it still does not work, call for help from your local guru or call for help from the free software community to solve your problem (see section *Technical support*, page 93).

2. `gllq.c:28: 'struct foo' undeclared (first use this function)`

The structures are special data types that all programs use. A lot of them are defined by the system in headers. That means the problem is certainly caused by a missing or misused header. The correct procedure for solving the problem is:

- try to check whether the structure in question is defined by the program or by the system. A solution is to use the command `grep` in order to see whether the structure is defined in one of the headers.

For instance, when you are in the root of the distribution:

```
$ find . -name '*.h' | xargs grep 'struct foo' | less
```

Many lines may appear on the screen (each time that a function using this type of structure is defined for instance). If it exists, pick out the line where the structure is defined by looking at the header file obtained by the use of `grep`.

The definition of a structure is:

```
struct foo {
    <contents of the structure>
};
```

Check if it corresponds with what you have. If so, this means that the header is not included in the faulty `.c` file. There are two solutions:

- add the line `#include "<filename>.h"` at the beginning of the faulty `.c` file.
- or copy-paste the definition of the structure at the beginning of this file (it is not really neat, but at least it usually works).
- If not, do the same thing with the system header files (which are usually in directories `/usr/include`, `/usr/X11R6/include`, or `/usr/local/include`). But this time, use the line `#include <<filename>.h>`.

4. Analyze the error message returned by `make`. Normally, the last lines should contain a directory (a message like `make[1]: Leaving directory '/home/benj/Project/foo'`). Pick out the one with the highest number. To check that it is the good one, go to that directory and execute `make` again to obtain the same error.

- If this structure still does not exist, try to find out which library (i.e. set of functions put together in a single package) it should be defined in (look in the `INSTALL` or `README` file to see which libraries are used by the program and their required versions). If the version that the program needs is not the one installed on your system, you will need to update this library.
- If it still does not work, check that the program properly works with your architecture (some programs have not been ported yet on all the *UNIX* systems). Check also that you have correctly configured the program (when `configure` ran, for instance) for your architecture.

3. parse error

This is a problem that is quite complicated to solve, because it often is an error that appears at a certain line, but after the compiler has met it. Sometimes, it is simply a data type that is not defined. If you meet an error message like:

```
main.c:1: parse error before 'glloq_t'
main.c:1: warning: data definition has no type or storage class
```

then the problem is that the `glloq_t` type is not defined. The solution to solve the problem is more or less the same that in the previous problem.



there may be a parse error in the old curses libraries if my memory serves me right.

4. no space left on device

This problem is easy to solve: there is not enough space on the disk to generate a binary file from the source file. The solution consists of making free space on the partition that contains the install directory (delete temporary files or sources, uninstall any programs you do not use). If you decompressed it in `/tmp`, rather than in `/usr/local/src`, which avoids needlessly saturating the `/tmp` partition. Check whether there are `core>` files on your disk. If so, delete them or make them get deleted if they belong to another user.

5. `/usr/bin/ld: cannot open -lglloq: No such file or directory`

That clearly means that the `ld` program (used by `gcc` during the last compilation stage) does not manage to find a library. To include a library, `ld` searches for a file whose name is in the arguments of type `-l<library>`. This file is `lib<library>.so`. If `ld` does not manage to find it, it produces an error message. To solve the problem, follow the steps below:

- Check that the file exists on the hard disk by using the `locate` command. Usually, the graphic libraries can be found in `/usr/X11R6/lib`. For instance:

```
$ locate libglloq
```

If the search is unrewarding, you can make a search with the `find` command (eg: `find /usr -name libglloq.so*`). If you still cannot find the library, you will have to install it.

- Once the library is located, check that it is accessible by `ld`: the `/etc/ld.so.conf` file specifies where to find these libraries. Add the incriminate directory at the end (you may have to reboot your computer for this to be taken into account). You also can add this directory by changing the contents of the environment variable `LD_LIBRARY_PATH`. For instance, if the directory to add is `/usr/X11R6/lib`, type:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/X11R6/lib
```

(if your shell is *bash*).

- If it still does not work, check that the library format is an executable file (or ELF) with the `file` command). If it is a symbolic link, check that the link is good and does not point at a non-existent file (for instance, with `nm libglloq.so`). The permissions may be wrong (if you use an account than root and if the library is protected against reading for example).

6. `glloq.c(.text+0x34): undefined reference to 'glloq_init'`

It is a problem of a symbol that was not solved during the last compilation stage. Usually, it is a library problem. There may be several causes:

- the first thing to do is to find out whether the symbol is **supposed** to be in a library. For instance, if it is a symbol beginning by `gtk`, it belongs to the `gtk` library. If the name of the library is easily identifiable (`froblicate_foobar`), you may list the symbols of a library with the `nm` command. For example,

```
$ nm libglloq.so
0000000000109df0 d glloq_message_func
000000000010a984 b glloq_msg
0000000000008a58 t glloq_nearest_pow
0000000000109dd8 d glloq_free_list
0000000000109cf8 d glloq_mem_chunk
```

Adding the `-o` option to `nm` allows you to print the library name on each line, which makes the search easier. Let's imagine that we are searching for the symbol `bulgroz_max`, a crude solution is to make a search like:

```
$ nm /usr/lib/lib*.so | grep bulgroz_max
$ nm /usr/X11R6/lib/lib*.so | grep bulgroz_max
$ nm /usr/local/lib/lib*.so | grep bulgroz_max
/usr/local/lib/libfroblicate.so:0000000000004d848 T bulgroz_max
```

Wonderful! The symbol `bulgroz_max` is defined in the `froblicate` library (the capital letter `T` is before its name). Then, you only have to add the string `-lfroblicate` in the compilation line by editing the `Makefile` file: add it at the end of the line where `LDFLAGS` or `LFGLAGS` (or `CC` at worst) are defined, or on the line corresponding to the creation of the final binary file.

- the compilation is being made with a version of the library which is not the one allowed for by the software. Read the `README` or `INSTALL` files of the distribution to see which version must be used.
- not all the object files of the distribution are correctly linked. The file where this function is defined is lacking. Type `nm -o *.o` to know which one it is and add the corresponding `.o` file on the compilation line if it is missing.
- the problem function or variable may be non-existent. Try to delete it: edit the problem source file (its name is specified at the beginning of the error message). It is a desperate solution whose consequence will certainly be a chaotic execution of the program with a (*segfault* at startup, etc).

7. Segmentation fault (core dumped)

Sometimes, the compiler hangs immediately and produces this error message. I have no advise except suggesting that you install a more recent version of your compiler.

8. no space on /tmp

Compilation needs temporary workspace during the different stages; if it does not have space, it fails. So, you may have to clean the partition, but be careful some programs being executed (*X* server, pipes, etc) can hang if some files are deleted. You must know what you are doing! If `/tmp` is part of a partition that does not only contain it (for example the root), search and delete some possible core files.

9. make/configure in infinite recursion

It is often a problem of time in your system. Indeed, `make` needs to know the date in the computer and the date of the files it checks. It compares the dates and uses the result to know whether the target is more recent than the dependence.

Some date problems may cause `make` to endlessly build itself (or to build and build again a sub-tree in infinite recursion). In such a case, the use of `touch` (whose use here is to put the files in argument at the current time) usually solves the problem.

For instance:

```
$ touch *
```

Or also (cruder, but efficient):

```
$ find . | xargs touch
```

13.5. Installation

13.5.1. With `make`

Now that all is compiled, you have to copy the built files to an appropriate place (usually in one of the sub-directories of `/usr/local`).

`make` can usually perform this task. A special target is the target `install`. So, using `make install` carries out the installation of the required files.

Usually, the procedure is described in the `INSTALL` or `README` file. But sometimes, the developer has forgotten to provide one. In that case, you must install everything by yourself.

Copy then:

- The executable files (programs) into the `/usr/local/bin` directory
- The libraries (`lib*.so` files) into the `/usr/local/lib` directory
- The headers (`*.h` files) into the `/usr/local/include` directory (be careful not to delete the originals).
- The data files usually go in `/usr/local/share`. If you do not know the installation procedure, you can try to run the programs without copying the data files, and to put them at the right place when it asks you for them (in an error message like `Cannot open /usr/local/share/glloq/data.db` for example).
- The documentation is a little bit different:
 - The man files are usually put in one of the sub-directories of `/usr/local/man`. Usually, these files are in `troff` (or `groff`) format, and their extension is a figure. Their name is the name of a command (for instance, `echo.1`). If the figure is `n`, copy the file in `/usr/local/man/man<n>`.
 - The info files are put in the directory `/usr/info` or `/usr/local/info`

You are finished! Congratulations! You now are ready to compile an entire operating system!

13.5.2. Problems

If you have just installed free software, *GNU tar* for instance, and if, when you execute it, another program is started or it does not work like it did when you tested it directly from the `src` directory, it is a `PATH` problem, which finds the programs in a directory before the one where you have installed the new software. Check by executing `type -a <program>`.

The solution is to put the installation directory higher in the `PATH` and/or to delete/rename the files that were executed when they were not asked for, and/or rename your new programs (into `gtar` in this example) so that there is no more confusion.

You can also make an alias if the shell allows it (for instance, say that `tar` means `/usr/local/bin/gtar`).

13.6. Support

13.6.1. Documentation

Several documentation sources:

- *HOWTOs*, short documents on precise points (usually far from what we need here, but sometimes useful). Look on your disk in `/usr/share/doc/HOWTO` (not always, they are sometimes elsewhere; check that out with the command `locate HOWTO`),

- The manual pages. Type `man <command>` to get documentation on the command `<command>`,
- Specialized literature. Several large publishers have begun publishing books about free systems (especially on *GNU/Linux*). It is often useful if you are a beginner and if you do not understand all the terms of the present documentation.

13.6.2. Technical support

If you have bought an “official” **Mandrake Linux** distribution, you can ask the technical support staff for information on your system. I think that the technical support staff have better things to do than help all the users to install additional software, but some of them do offer a x days-installation help. Perhaps they can spend some time on compilation problems?

You can also rely on help from the free software community:

- *newsgroups* (on *Usenet*) `comp.os.linux.*` (`news:comp.os.linux.*`) answer all the questions about *GNU/Linux*. Newsgroups matching `comp.os.bsd.*` deal with BSD systems. There may be other newsgroups dealing with other *UNIX* systems. Remember to read them for some time prior to writing to them.
- Several associations or groups of enthusiasts in the free software community offer voluntary support. The best way to find the ones closest to you, is to check out the lists on specialized web sites, or to read the relevant newsgroups for a while.
- Several *IRC channels* offer a real time (but blind) assistance by *gurus*. See for instance the `#linux` channel on most of the IRC network, or `#linuxhelp` on *IRCNET*.
- As a last resort, ask the developer of the software (if he mentioned his name and his *email* address in a file of the distribution) if you are sure that you have found a bug (which may be due only to your architecture, but after all, free software is supposed to be portable).

13.6.3. How to find free software

To find free software, a lot of links may help you:

- the huge FTP site `sunsite.unc.edu` (`sunsite.unc.edu`) or one of its mirrors
- the following web sites make a catalog of many free software that can be used on *UNIX* platforms (but one can also find proprietary software on these):
 - FreshMeat (<http://www.freshmeat.net/>) is probably the most complete site,
 - Linux France (<http://www.linux-france.org/>) contains a lot of links to software working with *GNU/Linux*. Most of them work of course with other free *UNIX* platforms,
 - GNU Software (<http://www.gnu.org/software/>) for an exhaustive list of all of GNU software. Of course, all of them are free and most are licensed under the GPL.
- you can also perform a search with a search engine like Altavista/ (<http://www.altavista.com/>) and Lycos/ (<http://ftpsearch.lycos.com/>) and make a request like: `+<software> +download` or `"download software"`.

13.7. Acknowledgments

- Proof-reading and disagreeable comments (and in alphabetical order): Sébastien Blondeel, Mathieu Bois, Xavier Renaut and Kamel Sehil.
- *Beta-testing*: Laurent Bassaler
- English translation: Fanny Drieu English editing: Hoyt Duff

Chapter 14. Compiling And Installing New Kernels

Along with filesystem mounting and building from sources, compiling the kernel is undoubtedly the subject which causes the most problems for beginners. Compiling a new kernel is not generally necessary, since the kernel installed by **Mandrake Linux** contains support for a significant number of devices (in fact, more devices than you will ever need or even think of), as well as a set of trusted patches and so on. But...

It may be that you want to do it, for no other reason than to see “what it does”. Apart from making your *PC* and your coffee machine work a bit harder than usual, not a lot. The reasons for why you should want to compile your own kernel range from deactivating an option to rebuilding a brand new experimental kernel. Anyway, the aim of this chapter is to ensure that your coffee machine still works after compilation.

There are other valid reasons for recompiling the kernel. For example, you have read that the kernel you are using has a security *bug*, which is fixed in a more recent version, or a new kernel includes support for a device you need. Of course, in these cases, you have the choice of waiting for binary upgrades, but updating the kernel sources and recompiling the new kernel yourself makes for a faster solution.

Whatever you do, stock up with coffee.

14.1. Where to Find Kernel Sources

You can basically get the sources from two places:

1. **Official Mandrake Linux Kernel.** In the SRPMS directory of any of the Cooker mirrors (<http://www.MandrakeLinux.com/en/cookerdevelopment.php3>), you will find the following packages:

kernel-2.4.??-mdk-?-mdk.src.rpm

The kernel sources for compiling the kernel used in the distribution. It is highly modified for more additional functionalities.

kernel-linux2.4-2.4.??-mdk.src.rpm

The stock kernel as published by the maintainer of *GNU/Linux* kernel.

If you choose this option (recommended), just download the source RPM, install it (as root) and jump to *Configuring The Kernel*, page 96.

2. **The Official Linux Kernel Repository.** The main kernel source host site is <ftp.kernel.org> (<ftp://ftp.kernel.org>), but there are a large number of mirrors, all named [ftp.xx.kernel.org](ftp://ftp.xx.kernel.org) (<ftp://ftp.xx.kernel.org>), where xx (xx) represents the ISO country code. Following the official announcement of the availability of the kernel, you should allow at least two hours for all the mirrors to be updated.

On all of these FTP servers, the kernel sources are in the `/pub/linux/kernel` directory. Next, go to the directory with the series that interests you: it will undoubtedly be v2.4. Nothing prevents you from trying the 2.5 version, but remember that these are experimental kernels. The file containing the kernel sources is called `linux-<kernel.version>.tar.bz2`, e.g. `linux-2.4.8.tar.bz2`.

You can also apply patches to kernel sources in order to upgrade them incrementally: thus, if you already have kernel sources version 2.4.8 and want to upgrade to kernel 2.4.10, you do not need to download the whole 2.4.10 source, you can simply download the *patches* `patch-2.4.9.bz2` and `patch-2.4.10.bz2`. As a general rule, this is a good idea, since sources currently take up more than 23 MB.

14.2. Unpacking Sources, Patching The Kernel (if Necessary)

Kernel sources should be placed in `/usr/src`. So you should go into this directory then unpack the sources there:

```
$ cd /usr/src
$ mv linux linux.old
$ tar xjf /path/to/linux-2.4.8.tar.bz2
```

The command `mv linux linux.old` is required: this is because you may already have sources of another version of the kernel. This command will ensure that you do not overwrite them. Once the archive is unpacked, you have a `linux` directory with the new kernel's sources.

Now, the patches. We will assume that you do want *to patch* from version 2.4.8 to 2.4.10 and have downloaded the patches needed to do this: go to the newly created `linux` directory, then apply the patches:

```
$ cd linux
$ bzipcat /path/to/patch-2.4.9.bz2 | patch -p1
$ bzipcat /path/to/patch-2.4.10.bz2 | patch -p1
$ cd ..
```

Generally speaking, moving from a version 2.4.x to a version 2.4.y requires you to apply all the patches numbered 2.4.x+1, 2.4.x+2, ..., 2.4.y in this order. To revert from 2.4.y to 2.4.x, repeat exactly the same procedure but applying the patches in reverse order and with option `-R` from `patch` (`R` stands for *Reverse*). So, to go back from kernel 2.4.10 to kernel 2.4.8, you would do:

```
$ bzipcat /path/to/patch-2.4.10.bz2 | patch -p1 -R
$ bzipcat /path/to/patch-2.4.9.bz2 | patch -p1 -R
```



If you wish to test if a patch will correctly apply before actually applying it, add the `--dry-try` option to the `patch` command.

Next, for the sake of clarity (and so you know where you are), you can rename `linux` to reflect the kernel version and create a symbolic link:

```
$ mv linux linux-2.4.10
$ ln -s linux-2.4.10 linux
```

It is now time to move on to configuration. For this, you have to be in the source directory:

```
$ cd linux
```

14.3. Configuring The Kernel

To start, go into the `/usr/src/linux` directory.

First, a little trick: you can, if you want, customize the version of your kernel. The kernel version is determined by the four first lines of the `Makefile`:

```
$ head -4 Makefile
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 10
EXTRAVERSION =
```

Further on in the `Makefile`, you can see that the kernel version is built as:

```
KERNELRELEASE=$(VERSION) . $(PATCHLEVEL) . $(SUBLEVEL) $(EXTRAVERSION)
```

All you have to do is modify one of these fields in order to change your version. Preferably, you will only change `EXTRAVERSION`. Say you set it to `-foo`, for example. Your new kernel version will then become 2.4.10-foo. Do not hesitate to change this field each time you recompile a new kernel with different versions, so that you can test different options while keeping old tries.

Now, on to configuration. You can choose between:

- `make xconfig` for a graphical interface;
- `make menuconfig` for an interface based on `ncurses`;
- `make config` for the most rudimentary interface, line by line, section by section;
- `make oldconfig` the same as above, but based on your former configuration. See *Saving, Reusing Your Kernel Configuration Files*, page 97.

You will go through the configuration section by section, but you can skip sections and jump to the ones that interest you if you are using `menuconfig` or `xconfig`. The options are **y** for **Yes** (functionality hard-compiled into the kernel), **m** for **Module** (functionality compiled as a module), or **n** for **No** (do not include it in the kernel).

Both `make xconfig` and `make menuconfig` have the options bundled in hierarchical groups. For example, `Processor family` goes under `Processor type` and `features`.

For `xconfig`, the button **Main Menu** is used to come back to the main menu when in a hierarchical group; **Next** goes to the next group of options; and **Prev** returns to the previous group. For `menuconfig`, use the `Enter` key to select a section, and switch options with **y**, **m** or **n** to change the options status, or else, press the `Enter` key and make your choice for the multiple choice options. **Exit** will take you out of a section or out of configuration if you are in the main menu. And there is also **Help**.

We are not going to enumerate all options here, as there are several hundreds of them. Furthermore, if you have reached this chapter, you probably know what you are doing anyway. So you are left to browse through the kernel configuration and set/unset whichever options you see fit. However, here are some advices to avoid ending up with an unusable kernel:

1. unless you use an initial ramdisk, **never** compile the drivers necessary to mount your root filesystem (hardware drivers and filesystem drivers) as modules! And if you use an initial ramdisk, say `Y` to `ext2fs` support, as this is the filesystem used for ramdisks. You will also need the `initrd` support;
2. if you have network cards on your system, compile their drivers as modules. Hence, you can define which card will be the first one, which will be the second, and so on, by putting appropriate aliases in `/etc/modules.conf`. If you compile the drivers into the kernel, the order in which they will be loaded will depend on the linking order, which may not be the order you want;
3. and finally: if you don't know what an option is about, read the help! If the help text still doesn't inspire you, just leave the option as it was. (for `config` and `oldconfig` targets, press the `?` key to access the help).

You may also consult the file `/usr/src/linux/Documentation/Configure.help` which gives the help text for every option in order of appearance. On its header, you will also find links to many translations.

And voilà! Configuration is finally over. Save your configuration and quit.

14.4. Saving, Reusing Your Kernel Configuration Files

The kernel configuration is saved in the `/usr/src/linux/.config` file. It is highly recommended that you make a backup copy of this file, for example in the `/root` directory, so that you can not only reuse it later, but also save configurations for different kernels, as this is just a matter of giving different names to configuration files.

One possibility is to name configuration files after the kernel version. Say you modified your kernel version as shown *Configuring The Kernel*, page 96, then you can do:

```
$ cp .config /root/config-2.4.10-foo
```

If you decide to upgrade to 2.4.12 (for example), you will be able to reuse this file, as the differences between the configuration of these two kernels will be very small. Just use the backup copy:

```
$ cp /root/config-2.4.10-foo .config
```

But copying back the file doesn't mean that the kernel is ready to be compiled just yet. You have to invoke `make menuconfig` (or whatever else you chose to use) again, because some files needed in order for the compilation to succeed are created and/or modified by these commands.

However, apart from the chore of going through all the menus again, you can possibly miss some interesting new option(s). You can avoid this by using `make oldconfig`. It has two advantages:

1. it's fast;
2. if a new option appears in the kernel and wasn't present in your configuration file, it will stop and wait for you to enter your choice.



After you have copied your `.config` to the root home, as proposed above, run `make mrproper`. It will ensure nothing remains from the old configuration and you will get a clean kernel.

Next, time for compilation.

14.5. Compiling Kernel And Modules, Installing The Beast

Small point to begin with: if you are recompiling a kernel with exactly the same version as the one already present on your system, the old modules must be deleted first. For example, if you are recompiling 2.4.10, you must delete the `/lib/modules/2.4.10` directory.

Compiling the kernel and modules, and then installing modules, is done with the following lines:

```
make dep
make clean bzImage modules
make modules_install install
```

A little vocabulary: `dep`, `bzImage`, etc., as well as `oldconfig` and others which we used above, are called **targets**. If you specify several targets to make as shown above, they will be executed in their order of appearance. But if one target fails, `make` won't go any further¹.

Let us look at the different targets and see what they do:

- `dep`: this computes the dependencies between the different source files. It is necessary to do so each time you change your configuration, otherwise some files may not be built and the compilation will fail;
- `bzImage`: this builds the kernel. Note that this target is only valid for *Intel* processors, and so is `zImage`. The difference between `bzImage` and `zImage` is that the former will generate a kernel which can be much larger. This target also generates the `System.map` for this kernel. We will see later what this file is used for;
- `modules`: this target will generate modules for the kernel you have just built. If you have chosen not to have modules, this target will do nothing;
- `modules_install`: this will install modules. By default, modules will be installed in the `/lib/modules/<kernel-version>` directory. This target also computes module dependencies (unlike in 2.2.x);
- `install`: this last target will finally copy the kernel and modules to the right places and modify the boot loader's configurations in order for the new kernel to be available at boot time. Do not use it if you prefer to perform a manual installation as described in *Installing The New Kernel Manually*, page 98.

At this point, everything is now compiled and correctly installed, ready to be tested! Just reboot your machine and choose the new kernel in the boot menu. Note that the old kernel remains available so that you can use it if you experience problems with the new one. However, you can choose to manually install the kernel and change the boot menus by hand. We will explain that in the next section.

14.6. Installing The New Kernel Manually

The kernel is located in `arch/i386/boot/bzImage` (or `zImage` if you chose to make `zImage` instead). The standard directory in which kernels are installed is `/boot`. You also need to copy the `System.map` file to ensure that some programs (`top` is just one example) will work correctly. Good practice again: name these files after the kernel version. Let us assume that your kernel version is 2.4.10-foo. The sequence of commands you will have to type is:

```
$ cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.10-foo
$ cp System.map /boot/System.map-2.4.10-foo
```

Now you need to tell the bootloader about your new kernel. There are two possibilities: *grub* or *LILO*. Note that **Mandrake Linux** is configured with *LILO* by default.

1. In this case, if it fails, it means that there is a bug in the kernel... If this is the case, please report it!

14.6.1. Updating Grub

Obviously, retain the possibility of starting your current kernel! The simplest way of updating *grub* is to use *drakboot* (see chapter Change Your Boot-up Configuration in the *User Guide*). Alternatively, you can manually edit the configuration file as follows.

You need to edit the `/boot/grub/menu.lst` file. This is what a typical `menu.lst` looks like, after you have installed your **Mandrake Linux** distribution and before modification:

```
timeout 5
color black/cyan yellow/cyan
i18n (hd0,4)/boot/grub/messages
keytable (hd0,4)/boot/fr-latin1.klt
default 0

title linux
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5

title failsafe
kernel (hd0,4)/boot/vmlinuz root=/dev/hda5 failsafe

title Windows
root (hd0,0)
makeactive
chainloader +1

title floppy
root (fd0)
chainloader +1
```

This file is made of two parts: the header with common options (the five first lines), and the images, each one corresponding to a different *GNU/Linux* kernel or another OS. `timeout 5` defines the time (in seconds) for which *grub* will wait for input before it loads the default image (this is defined by the `default 0` directive in common options, i.e. the first image in this case). The `keytable` directive, if present, defines where to find the keymap for your keyboard. In this example, this is a French layout. If none are present, the keyboard is assumed to be a plain QWERTY keyboard. All `hd(x,y)` which you can see refer to partition number *y* on disk number *x* as seen by the *BIOS*.

Then come the different images. In this example, four images are defined: `linux`, `failsafe`, `windows`, and `floppy`.

- The `linux` section starts by telling *grub* about the kernel which is to be loaded (`kernel hd(0,4)/boot/vmlinuz`), followed by the options to pass to the kernel. In this case, `root=/dev/hda5` will tell the kernel that the root filesystem is located on `/dev/hda5`. In fact, `/dev/hda5` is the equivalent of *grub*'s `hd(0,4)`, but nothing prevents the kernel from being on a different partition than the one containing the root filesystem;
- the `failsafe` section is very similar to the previous one, except that we will pass an argument to the kernel (`failsafe`) which tells it to enter "single" or "rescue" mode;
- the `windows` section tells *grub* to simply load the first partition's boot sector, which probably contains a *Windows* boot sector;
- the `floppy` section simply boots your system from the floppy disk in the first drive, whatever the system installed on it. It can be a *Windows* bootdisk, or even a *GNU/Linux* kernel on a floppy;



Depending on the security level you use on your system, some of the entries described here may be absent from your file.

Now to the point. We need to add another section to tell *grub* about our new kernel. In this example, it will be placed before the other entries, but nothing prevents you from putting it somewhere else:

```
title foo
kernel (hd0,4)/boot/vmlinux-2.4.10-foo root=/dev/hda5
```

Don't forget to adapt the file to your configuration! The *GNU/Linux* root filesystem here is `/dev/hda5`, but it can be somewhere else on your system.

And that's it. Unlike *LILLO*, as we will see below, there is nothing else to do. Just restart your computer and the new entry you just defined will appear. Just select it from the menu and your new kernel will boot.

If you compiled your kernel with the framebuffer, you will probably want to use it: in this case, you need to add a directive to the kernel which tells it what resolution you want to start in. The list of modes is available in the `/usr/src/linux/Documentation/fb/vesafb.txt` file (only in the case of the VESA framebuffer! Otherwise, refer to the corresponding file). For the 800x600 mode in 32 bits², the mode number is 0x315, so you need to add the directive:

```
vga=0x315
```

and your entry now resembles:

```
title foo
kernel (hd0,4)/boot/vmlinuz-2.4.10-foo root=/dev/hda5 vga=0x315
```

For more information, please consult the info pages about *grub* (`info grub`).

14.6.2. Updating LILO

The simplest way of updating *LILLO* is to use *drakboot* (see chapter Change Your Boot-up Configuration in the *User Guide*). Alternatively, you can manually edit the configuration file as follows.

The *LILLO* configuration file is `/etc/lilo.conf`. This is what a typical `lilo.conf` looks like:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
image=/boot/vmlinuz-2.4.8-17mdk
    label=linux
    root=/dev/hda1
    read-only
other=/dev/hda2
    label=dos
    table=/dev/hda
```

A `lilo.conf` file consists of a main section, followed by a section for each operating system. In the example of the file above, the main section is made up of the following directives:

```
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
vga=normal
default=linux
keytable=/boot/fr-latin1.klt
lba32
prompt
timeout=50
message=/boot/message
```

2. 8 bits means 2^8 colors, i.e. 256; 16 bits means 2^{16} colors, i.e. 64k, i.e. 65536; in 24 bits as in 32 bits, color is coded on 24 bits, i.e. 2^{24} possible colors, in other words exactly 16M, or a bit more than 16 million.

The `boot=` directive tells *LILO* where to install its boot sector; in this case, it is the MBR (*Master Boot Record*) of the first IDE hard disk. If you want to make a *LILO* floppy disk, simply replace `/dev/hda` with `/dev/fd0`. The `prompt` directive asks *LILO* to show the menu on start-up. As a timeout is set, *LILO* will start the default image after 5 seconds (`timeout=50`). If you remove the `timeout` directive, *LILO* will wait until you have typed something.

Then comes a `linux` section:

```
image=/boot/vmlinuz-2.4.8-17mdk
    label=linux
    root=/dev/hda1
    read-only
```

A section to boot a *GNU/Linux* kernel always starts with an `image=` directive, followed by the full path to a valid *GNU/Linux* kernel. Like any section, it contains a `label=` directive as a unique identifier, here `linux`. The `root=` directive tells *LILO* which partition hosts the root filesystem for this kernel. It may be different in your configuration... The `read-only` directive tells *LILO* that it should mount the root filesystem as read-only on start-up: if this directive is not there, you will get a warning message.

Then comes the *Windows* section:

```
other=/dev/hda2
    label=dos
    table=/dev/hda
```

In fact, a section beginning with `other=` is used by *LILO* to start any operating system other than *GNU/Linux*: the argument of this directive is the location of this system's boot sector, and in this case, it is a *Windows* system. To find the boot sector, located at the beginning of the partition hosting this other system, *GNU/Linux* also needs to know the location of the partition table which will enable it to locate the partition in question. This is done through the `table=` directive. The `label=` directive, as with the `linux` section above, identifies the section.

Now, it's time we added a section for our new kernel. You can put this section anywhere behind the main section, but don't enclose it within another section. Here is what it can look like:

```
image=/boot/vmlinuz-2.4.10-foo
    label=foo
    root=/dev/hda1
    read-only
```

Of course, adapt it to your configuration! We took a different situation on purpose (not the same as the *grub* one showed above...)

If you compiled your kernel with the `framebuffer`, refer to the corresponding paragraph above concerning *grub*. The only difference is that the option is alone on a new line:

```
vga=0x315
```

So this is what our `lilo.conf` looks like after modification, decorated with a few additional comments (all the lines beginning with `#`), which will be ignored by *LILO*:

```
#
# Main section
#
boot=/dev/hda
map=/boot/map
install=/boot/boot.b
# At boot, we want normal VGA. The framebuffer will switch resolutions by
# itself if we use it:
vga=normal
# Our boot message...
message=/boot/message
```

```
# What should be booted by default. Let's put our own kernel as the default:
default=foo
# Show prompt...
prompt
# ... wait 5 seconds
timeout=50
#
# Our new kernel: default image
#
image=/boot/vmlinuz-2.4.10-foo
    label=foo
    root=/dev/hda1
    read-only
# If the VESA framebuffer is used:
    vga=0x315
#
# The original kernel
#
image=/boot/vmlinuz-2.4.8-17mdk
    label=linux
    root=/dev/hda1
    read-only
#
# Windows Section
#
other=/dev/hda2
    label=dos
    table=/dev/hda
```

This could well be what your `lilo.conf` will look like... but remember, again, to adapt it to your own configuration.

Now that the file has been modified appropriately, but unlike *grub* which does not need it, you must tell *LILO* to change the boot sector:

```
$ lilo
Added foo *
Added linux
Added dos
$
```

In this way, you can compile as many kernels as you want, by adding as many sections as necessary. All you need to do now is restart your machine to test your new kernel.

Chapter 15. Troubleshooting

15.1. Introduction

This chapter will guide you through some troubleshooting basics, that is: what to do when everything goes wrong or, better yet, what to do to be **prepared** for when something goes wrong and how to fix it.

How many times have you felt foolish for not having backed up that little configuration file you've just managed to break? How many times have you lost all of your program's preferences after installing misbehaving software or even after accidentally deleting some configuration file? How many times has your computer stopped booting after that bleeding edge kernel testing you were doing? I have, many times... actually more than I would like to admit :-)

There are some people who recompile their kernel or tweak their configuration files every day of the week, every week of the month, every month of the year. You might not be one of them but, believe me, some day you will want or need to do that; so let's assume that these are not uncommon scenarios in every day *GNU/Linux* life. All of them can be managed without any hassle at all if you use a little common sense and follow some practices and guidelines we will introduce you to. These will help you when **those** times come.

So, on to the basic things you need to have ready...

15.2. A Boot Disk

The very first thing you will need when your system cannot boot up, for any of the reasons we mentioned before, will be a boot disk. You should have one already, created during the installation process. A boot disk will allow you to boot your system up and, in a matter of minutes, enable you to undo the thing that has made your system unable to boot.



You can also use the Rescue Mode of **Mandrake Linux**'s installation CD-ROM to boot your machine and perform some maintenance tasks, but a boot disk might prove to be useful anyway (for example, if your machine does not support booting from the CD-ROM drive).

You have two ways of creating a boot disk under **Mandrake Linux**, either by using the console, or a graphical one. To create a boot disk you have to be root. If you fire up the graphical program as a normal user, you will be asked for your root password before continuing.

15.2.1. Creating a Boot Floppy From The Console

So, you are on a console, su to become root and type the following:

```
[root@localhost]# mkbootdisk --device /dev/fd0 'uname -r'
```

and tap **Enter**. Doing so will give you something like this:

```
Insert a disk in /dev/fd0. Any information on the disk will be
lost. Press <Enter> to continue or ^C to abort:
```

Let's explain the example given. Among other parameters, the two that `mkbootdisk` needs are `--device [device]`, which tells `mkbootdisk` the device we want to write the boot disk to. In our example, we have chosen `/dev/fd0` which is the first floppy drive in the system. In 99.9% of the cases that should work, if it doesn't work for you, well, choose the right device to use.

The other parameter needed is `[kernel-version]`, which tells `mkbootdisk` which kernel we want to put on it. In our example, we use `'uname -r'` which gives as a result the name of the current running kernel. Thus, the example given will create a boot disk in the first floppy drive with the current running kernel on it.

Please note that this will create a boot disk that is based on your current kernel (in case you choose that) with all the modules and stuff which that kernel has. If you do not want to include all the modules or want to include other modules (for example one for tape support), you'd better use our graphical tool *drakfloppy*.

15.2.2. Using drakfloppy to Create a Boot Disk

drakfloppy is a graphical tool that lets you create a highly customized boot disk. To start *drakfloppy* go to the **Configuration/Boot and Init** menu and fire it up from there. You will be asked for your root password unless you already are root as seen in figure 15-1:

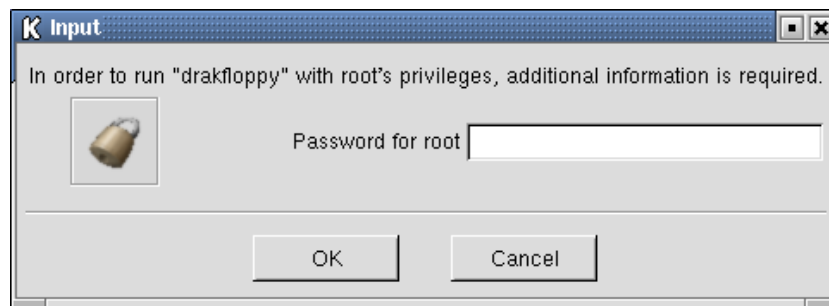


Figure 15-1. Enter your root password

After that you will be looking at *drakfloppy*'s main window (figure 15-2). If you want to create a "default" boot disk, that is, one that is the same as if made with the console example given above, all you have to do is to insert a floppy disk in the appropriate floppy drive, select that drive from the pull-down list and press **Build the disk**.

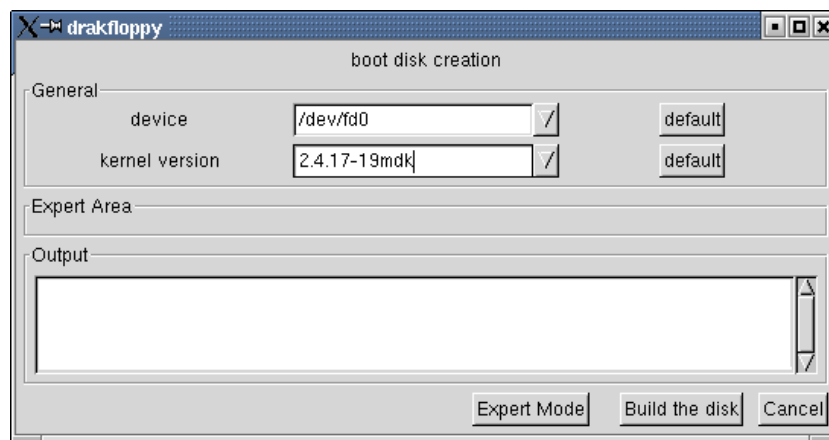


Figure 15-2. drakfloppy's main window

If you want to customize your boot disk, you will have to hit the **Expert mode** button and *drakfloppy*'s window will change as shown in figure 15-3.

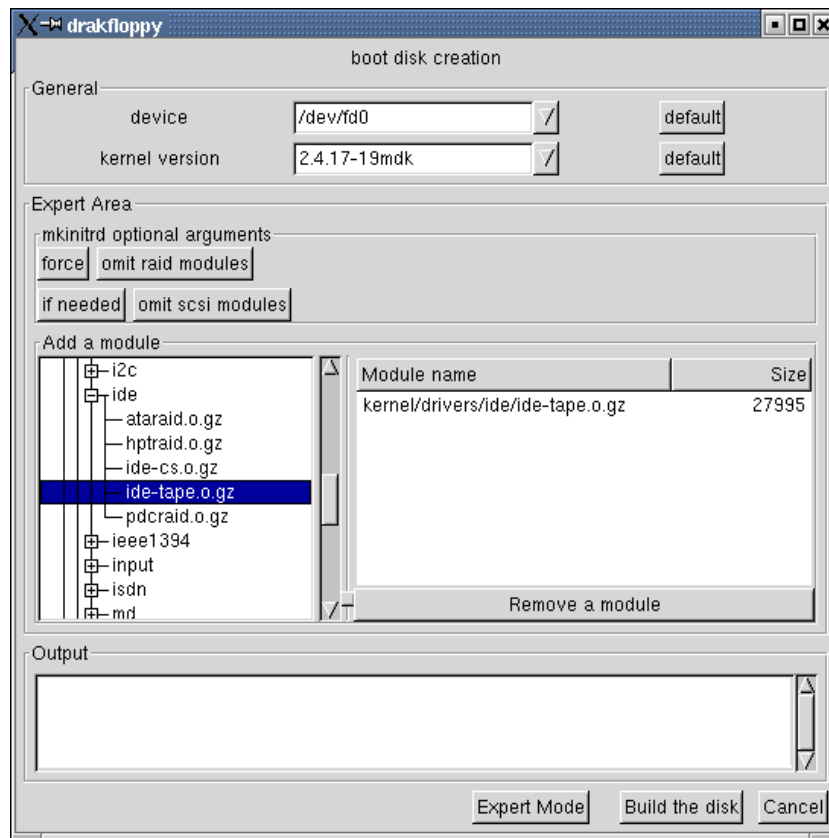


Figure 15-3. Making a customized boot disk

In the **Expert Area** you have two sections: one with some buttons with options for `mkinitrd` and another with the module “tree”. The buttons are self-explanatory. In this example we want to use the IDE tape module and pre-load it. When you are done customizing the boot disk press **Build the disk** to create it.

15.2.3. Testing the boot disk

Always test your boot floppy to make sure it **actually works**. There are few things more embarrassing than finding that the floppy will not boot because of media errors. If the floppy boots OK then...

15.2.4. You are Done!

Congratulations! You already have the most important tool in trying to recover a damaged system: a boot disk. Now, on to some important considerations on the second most important tool: backups.

15.3. Backup

15.3.1. Why Backup?

Backing up your system is the **only** means of being able to repair it if it suffers severe damage, if you accidentally delete some important system files, or if someone breaks into your system and intentionally deletes some files. You should also backup your daily use data (compressed audio, images, office documents, e-mails, address book, etc.) to be safe.

You should make your backups using an appropriate medium and keep them in a safe place. Such a place should be outside the place you usually work in, if possible. You can even have two backups, one in place, and one outside. Generally speaking, you should make sure that you will be able to restore those backups if you want all this to be really useful.

15.3.2. Preparing Your System

You probably have everything you need already installed in your system. You should also keep a boot disk near at hand (you **made** one, didn't you?). Actually, you can make backups using only tar and, optionally, a compression tool such as gzip or bzip2. See an example in *Backup Example Using TAR*, page 110.

As an alternative, you can use specialized backup programs, such as *Taper*, *Time Navigator*, *Arkeia*, etc.

15.3.3. What to Backup?

Well, this might be the single most difficult question every system administrator asks himself when the time to backup comes. The answer to it depends on things such as: are you just backing up your personal data, your configuration files, or your whole system? How much time and/or space is it going to take? Will you be restoring your backup on the same machine/OS version, or on a different one?

Since this is a troubleshooting guide, we will try to focus on doing a backup that will allow us to quickly restore our system to the state it was before that terrible thing that rendered it unusable happened. Of course, you will need to make a backup of your personal data if you don't want to lose it, but... that is another story.

As a rule of thumb, you will need to backup the following directories: `/etc`, `/home`, `/root` and `/var`. If you do a complete backup of these directories, you will have saved not only your configurations, but your data as well (in case you are wondering where your data is, it is in the `/home/your_user_name/` directory). Please bear in mind that this can take a **long** time to complete, but it is the safest bet.

A more sophisticated scheme would be to backup only the configuration files that have changed, leaving alone the ones that have not changed. This will take more "planning" time, but will lend to quicker backups (and quicker restores, too) and to backups that are "easier" to port from one machine/OS version to another.

Next, you will be presented with a list of the files you should pay the most attention to. Note that these lists are not exhaustive at all, especially if you have made lots of changes to your system¹

In the directory `/etc`:

`/etc/lilo.conf`

Holds *LILLO*'s boot loader configuration. If you use *grub* instead of *LILLO*, the files to backup are the ones in the directory `/boot/grub`.

`/etc/fstab`

Holds the disks partition table configuration and the associated mount points.

`/etc/modules.conf`

Holds the modules to load and their parameters according to your system's hardware. This might not be useful if restoring on a **very** different machine, but it might have some hints anyway.

`/etc/isapnp.conf`

Holds ISAPnP's settings if you use this to configure your ISA Plug & Play hardware.



With kernel 2.4.x you might not need this file anymore, since *plug'n'play* hardware is configured using the DevFS file system.

`/etc/X11/XF86Config`

Holds *X*'s settings. *X* is the graphical core of *GNU/Linux* and all its desktop environments and window managers.

`/etc/cups`

Holds *cups*'s settings. *cups* is **Mandrake Linux**'s default printing system.

`/etc/printcap`

If you do not use *cups* and you use the *lpr* printing system then you must backup this one instead of `/etc/cups` for your printer settings.

1. If you have made lots of changes, you probably won't need these lists anyway.

`/etc/bashrc`

Sets the *bash shell* systemwide configuration.

`/etc/profile`

Sets the system environment and some programs that are executed upon system startup.

`/etc/crontab`

Sets the cron jobs to be executed periodically, for system maintenance tasks for example.

`/etc/rc.d/*`

Sets the various runlevels of the system. Usually you will not need to backup these ones, but if you have added some personalized runlevels or changed a default one, you will need to back these up.

`/etc/inittab`

Sets the default runlevel your system will start with.

`/etc/ssh`

Keeps ssh settings. If you do secure remote access this file is **very** important to keep.

If you have a web server, an FTP server, or other servers, make a backup of their respective configuration files too.

In the directory `/root` and each user's home directory `/home/user_name`, the following directories:

`~/ .gnome/*`

Settings for the *GNOME* desktop environment.

`~/ .kde/*`

Settings for the *KDE* desktop environment.

`~/ .netscape/*`

Settings for Netscape's family of programs. Navigator's bookmarks, Messenger's mail filters, etc.

`~/ .nsmail/*`

Holds **all** your e-mail and newsgroups messages. You **definitely** do not want to loose these ones, do you?

`~/ Mail/*`

If you use *kmail* this directory holds **all** your e-mail messages. You **definitely** do not want to loose these ones, do you?

`~/ .ssh/*`

Holds personalized settings for ssh usage. If you use ssh, backup this one...

You also want to keep an eye on the following files:

`~/ .bash_profile`

Holds environment variables, aliases, and more settings for the *bash shell*.

`~/ .bashrc`

More *bash* settings.

`~/ .cshrc`

Holds environment variables, aliases, and more settings for the *CSH shell*.

`~/ .tcshrc`

Holds environment variables, aliases, and more settings for the *tcsh* shell.

Please note that we did not mention every single possible configuration file because we would need to write a whole book on that subject. For example, if you do not use *netscape* you need not backup *netscape* related files and directories, if you do not use *ssh* you need not backup *ssh* related stuff, and so on.

Summarizing, backup all of the configuration files of the programs you use and all of the configuration files you have changed. Also backup all your personal (and your system's users) data files. You will not regret it, trust me.

15.3.4. Where to Backup?

The other big question to answer. This depends on how much do you want to backup, how fast do you want to make your backups, how easy is the access to the backup media, and a large list of etceteras.

Generally speaking, you need media that is at least as large as the amount of information you want to backup, and fast enough so the whole process will not take forever to complete.

15.3.5. Backup Media

Next, we will provide you with a brief description of available backup media options. These vary in capacity, reliability, and speed. They are not given in any particular order, just as they come to mind. Please note that your backup software may or may not support all of them.



This list is not intended to be an exhaustive analysis of the different storage media available out there. In fact, some of the things written below might change in the future. Things such as expected media life were taken from the manufacturer's web sites and/or personal and community experience. Also, there might be many **personal** points of view on many matters such as price or speed, for example.

Floppy Disk

Its capacity goes up to 1.44 MB². They are easy to carry around but for today's needs they may not have enough room. Best way to carry small files. Slow. Cheap. Standard floppy disk drive in virtually every computer. Read/Write. Expected media life is 4 or 5 years.



Please bear in mind that floppy disks are not too reliable.

LS120 Floppy Disk

Its capacity is 120 MB. Identical physical dimensions than floppy disks but with almost ten times more the capacity. Not so cheap. Needs a special floppy disk drive but this drive can also read/write standard floppy disks. Might be a good floppy replacement but it's speed lags behind that of *ZIP* drives. Read/Write. Expected media life is more or less the same than that of *ZIP* drives.

ZIP Disk

Its capacity goes up to 250 MB. Although not as thin as floppy disks, they are easy to carry around, too, and are more fit for today's needs. Good balance of features, although it may be a little expensive. Read/Write. Expected media life is 10 years for 100 MB units, maybe more for 250 MB units.

2. Well, actually they can be formatted at up to 1.92 MB using programs like *SuperFormat* and your standard floppy disk drive, but that's another story...

CD-R

Its capacity goes up to 700 MB these days, although the standard is 650 MB. Very cheap and reliable media. Today many argue that its capacity is not enough, but, hey, 650 MB sounds fine to me. Its strongest feature is that almost every computer on earth has a CD-ROM drive, so they can be read almost everywhere. Write only once. Read as many as you want (well, actually, as you can...). Expected media life is 20 years, maybe more if they are stored in a safe place and read not too often.

CD-RW

Same considerations as for CD-R, but it can be formatted and re-written as many as 1000 times. All in all, cheap and reliable media. Expected media life is 15 years, maybe more if they are stored in a safe place and read not so often.

DVD recordable/rewritable

This is one of the newest additions to the available storage media. Its capacity is 4.7 GB for single sided recordable DVD discs. Drives are a little bit expensive but that is more or less compensated by the fact of being able to store 4.7 GB on a single disk. Expected media life is 15 years, maybe more if they are stored in a safe place and read not so often.

Tape

Its capacity goes from 120 MB (anyone has tape drives this old?) up to several gigabytes. Expensive and not very reliable media (hey, they **are** magnetic tapes after all). Even so, their capacity makes them ideal to backup servers and the like; if you want to backup your whole disk drive in only one piece of media, tape is the only way to go (at least, for now...). Its biggest drawback is that tape access is sequential, and this has a big performance impact, but SCSI tape drives are fast enough for today's needs and, hey, they **do** have many gigabytes of room to store your files. Read/Write. Expected media life is up to 30 years for new tape technologies.

Hard Disk

You might be wondering: Has this guy had whiskey for breakfast? No, I have not had whiskey for breakfast, it's just that today's disk prices have dropped in such a way that they might be considered seriously as a backup medium, too. They are relatively cheap, have huge capacities (up to 100 GB at the time of writing this manual), are very reliable and the fastest of all media introduced in this list. If you have a laptop system this may not be an option³, but on your desktop systems adding a spare disk drive just for backup purposes might be a good choice. Actually, you might not even need to add a new hard disk and do backups on the only hard disk you have; but this might not prove to be a good idea since it will not protect you from a hard disk crash.

Other removable media

Other removable media exist (Castlewood's *DRB*, and IOMEGA's *JAZ* come to mind) that have good price/features balance and are suited for doing backups. Some were even publicized as "hard disk replacements" (*JAZ* for example), but when used as hard disks they might not last too long due to design constraints (They **are not** hard disk drives). Anyway, Your Mileage May Vary on these matters, just make sure you choose wisely (use common sense) according to your needs, and ... good luck!

Remote directories

Well, this might not be considered strictly as "media", but we will say a little about it because it is a good backup choice provided you have enough space and bandwidth.

If your ISP provides you with some space you can use that space to place your files along with your web pages. You can find on the web many offerings for online remote storage services. If you have a network with two or more machines you can do backups on some "remote" machine on the network (other than the one you are trying to backup, of course...)

Actually, doing "remote" backups could be a security hole, so do not keep your top secret nor your most important files remotely backed up. Remember that, in the case of a major system failure, you may not even be able to connect to that remote site to recover your files...

3. If you have a relatively new laptop, you may have space to install a second hard disk. Also, using USB or parallel port, you can attach extra external USB or parallel hard disk drives.

Please bear in mind that you can also combine backup medium according to your backup strategy, for example: tapes and CD-R/DVD+RW, hard disk and tapes, hard disk and CD-R/DVD+RW, etc.

15.3.6. When to Back Up?

There are many policies for backup schedules. Here we will introduce you to a few. Please, bear in mind that these are not mandatory, nor the best ones, nor the only ones. These are just guidelines you may want to follow in rolling out your own backup schedule.

There are many backup strategies out there, they depend on the media you use, on how often your data changes, and on how critical that data is to you or your organization. For example, one strategy states that you should make a full backup each weekend, and an incremental (changed stuff only) backup every day; then make a full backup every month and store that one in at least two places. This strategy might prove useful for an enterprise, but not for a personal computer. For your personal backups you can think of something like this: make a weekly backup of your files on your disk drive and each month transfer those backups to CD-R/DVD+RW or tape.

15.3.7. Backup Example Using TAR

Next, we will introduce you to a little backup script that uses tar for making a complete backup of your home directory.



You need read permissions on the files, and read and execute permissions on directories, you are going to backup, otherwise the backup operation will fail.

```
#!/bin/bash

# Create a compressed backup of your home directory in a file named
# backup.tar.gz or backup.tar.bz2 depending on the compression scheme used.

BACKUP_DIRS=$HOME

# Uncomment the following line if you want GZipped backups
#tar cvzf backup.tar.gz $BACKUP_DIRS

# We do a BZipped backup here...
tar cvjf backup.tar.bz2 $BACKUP_DIRS
```

As you can see this is a **very** simple backup script that only does a backup of your home directory and puts the result into the very same directory. Let's enhance it a little...

```
#!/bin/bash

# Create a compressed backup of all the directories specified and put the
# resulting file in a directory of our choice.

BACKUP_DIRS="$HOME /etc /etc/rc.d"
BACKUP_FILENAME='date +%b%d%Y'
BACKUP_DEST_DIR=$HOME

# Uncomment the following line for GZipped backups, comment for BZipped backups
#tar cvzf $BACKUP_DEST_DIR/$BACKUP_FILENAME.tar.gz $BACKUP_DIRS

# We do a BZipped backup here...
# Comment the following line for GZipped backups, uncomment for BZipped backups
tar cvjf $BACKUP_DEST_DIR/$BACKUP_FILENAME.tar.bz2 $BACKUP_DIRS
```

As you can see in this last example, we have added some more directories to our backup, and we have used a naming scheme to add the date of the backup to the resulting filename.

Of course, you can later move the resulting tar.bz2 or tar.gz file to any media you want. You can even backup directly to the media you want by mounting it and changing the variable BACKUP_DEST_DIR of the script accordingly. Feel free to enhance this script and make it as flexible as you want.

To restore the backups made this way, please look at *Restore Example Using TAR*, page 111

15.4. Restore

The restoration of a backup depends on which program, media, and schedule you used to make it. We will not cover all the restore cases here, but only mention that you have to make sure that you restore the files and/or directories to the same places they were in when you made the backup in order to recover your settings, and data files.

15.4.1. Restore Example Using TAR

Now, we will see a little script to restore the backup we made with `tar` using the script introduced earlier in *Backup Example Using TAR*, page 110



You need write permissions on the files and directories you are going to restore, otherwise the restore operation will fail.

```
#!/bin/bash

# Extract a compressed backup of all the directories specified putting the
# backed up files into their original places.

BACKUP_SOURCE_DIR=$HOME
RESTORE_FILENAME=$1

# Uncomment the following line if you are restoring GZipped backups
#tar xvzf $BACKUP_SOURCE_DIR/$RESTORE_FILENAME

# Restore a BZipped backup here...
tar xvjf $BACKUP_SOURCE_DIR/$RESTORE_FILENAME
```

As you can see, this script is simple enough. All we have to do is to pass it the filename of the backup we want to restore as a parameter (just the filename, not the full path), and it restores the backed up files into their original locations.

15.4.2. Making a Recovery CD-ROM

There is way to be prepared in case of “total disaster”, and that is making a **full** backup of your system. Programs such as *mkCDrec* can be very useful to get you up and running in a matter of minutes.

If you are the proud owner of a Mandrake Linux - PowerPack Deluxe Edition, you already have this tool in the “contribs” CD-ROM. Otherwise, you can find it, together with its documentation on the *mkCDrec* web site (<http://mkcdrec.ota.be>).

mkCDrec allows you to do multiple-CD-ROM volumes, disk cloning (copying the full contents of a disk or partition to another one with similar characteristics – at least the same size), and many more things.

In order to restore a system with *mkCDrec* you just have to boot with the first CD-ROM of the multiple-CD-ROM volume and follow the on-screen instructions.

15.5. My System Freezes at Boot Time

It could happen that your system hangs during boot up. If so, don’t panic, just keep reading.



The next sections are not introduced in any particular order.

15.5.1. System Hanging During Boot

If your system hangs during Rebuilding RPM database or Finding module dependencies, just press **CTRL+C**. This will allow the system to skip this step and continue to boot. Once booted, execute `rpm --rebuilddb` as root if the hang was at the Rebuilding RPM database phase. If the hang was at the Finding module dependencies phase you have most likely been through a kernel upgrade, but have not done it properly. Check if the files in `/boot` and the `/lib/modules` directory match the current kernel version (i.e., have the current version number attached). If they do not match, please read “*Compiling And Installing New Kernels*”, page 95, to find out how to fix this.

If the boot process hangs at `RAMDISK: Compressed image found at block 0` you have messed up the `initrd` image. Either try to boot another `lilo.conf` entry or boot an emergency system and remove or change the `initrd=` section in `/etc/lilo.conf`

15.5.2. File System Check on Boot Fails

If, for any reason, you have not shutdown your box properly, the system will run a routine file system check during the next boot. It may sometimes fail to do this on its own and will drop you on a console. Execute `e2fsck -py [device]` where `[device]` is the name of the partition on which the automatic check has failed. The `-p` switch tells `e2fsck` to do all the necessary repairs without asking, `-y` assumes the answer yes to all questions. When the check and repair phase is over, press **CTRL+D** to leave the emergency console. The system will reboot.

If you get this error regularly, there might be bad blocks on your disk. Execute `e2fsck -c [device]` to find out. This command will automatically mark any bad blocks and thus prevent the file system from storing data in these blocks. `e2fsck` checks the file system automatically only if it has not been unmounted properly during the previous system shutdown; or if the `maximal mount count` has been reached. To force a check, use the `-f` option.



The check for bad blocks on a disk should only be done on unmounted file systems, and can take a tremendous amount of time to complete. It may be necessary to do it, but be warned that you'll have enough time to drink several coffees.

15.6. Bootloader Reinstall

Sometimes you make a mistake and wipe your disk's MBR (Master Boot Record), or some misbehaving program does it, or if you dual boot with *Windows* and catch a virus that does it. So, you say, I won't be able to boot my system anymore, right? **Wrong!** There are many ways to recover the boot loader.

15.6.1. Using a Boot Disk

This one is not a surprise for anybody: To recover your boot loader you will **need** a boot disk. Without a boot disk of some kind you might be completely lost⁴. You have made a boot disk, haven't you?

Well, just put the diskette in the floppy drive and reboot your computer. What you do next varies according whether you use *LILO* or *grub*. No matter which boot loader you use, all the commands you have to execute will need to be run as root.

15.6.1.1. With LILO

If you use *LILLO*, you just need to issue the following at the command prompt: `/sbin/lilo`. This will reinstall *LILLO* in your disk's boot sector and that will fix the problem.

4. Unless you make a backup of your MBR, more on that later..

15.6.1.2. With GRUB

If you use *grub* things are a little bit different than when using *LILO*... but don't be afraid, we are here to help.



The following example will assume that you are trying to install *grub* in the MBR of your first IDE drive, and that the file *stage1* is in the directory */boot/grub/*.

First, invoke *grub*'s shell by issuing the command: *grub*. Once there, issue the following command: *root (hd0,0)*; this will tell *grub* that the files it needs are in the first partition (0) of your first hard disk (*hd0*). Then issue the following command: *setup (hd0)*; this will install *grub* in the MBR of your first hard disk. That's it!

You can also try to use *grub-install /dev/hda* to install *grub* on your first hard drive's MBR, but the method described above is the preferred one.

15.6.1.3. You Are Done!

Well, that is all there is to know about reinstalling your boot loader.

15.6.2. Repairing a Damaged Super-Block



The information below only applies to ext2 and ext3 file systems. If you have another file system, please check the documentation on it for information about this.

The super-block is the first block of each ext2fs partition. It contains important data about the file system, like size, free space, etc. (It is similar to the method used by FAT partitions). A partition with a damaged super-block cannot be mounted. Fortunately, ext2fs keeps several super-block backup copies scattered over the partition.

Boot your system with the boot disk you created earlier (You **have** created one, right?). The backup copies are usually located at the beginning of each 8 KB (8192 bytes) block. So, the next backup copy is in byte number 8193. To restore the super-block from this copy, execute *e2fsck -b 8193 /dev/hda4*; change *hda4* accordingly to reflect the name of your damaged partition. If that block also happens to be damaged, try the next one at byte number 16385, and so on until you find a suitable one. Reboot your system to activate the changes.

15.7. Runlevels

15.7.1. Brief Description of What Runlevels Are

A runlevel is a configuration of the system software that only allows certain selected processes to exist. Allowed processes are defined, for each runlevel, in the file */etc/inittab*. There are eight defined runlevels: 0, 1, 2, 3, 4, 5, 6, S. You can create your own runlevel too. For a more detailed description about runlevels, please refer to "*The Start-Up Files: init sysv*", page 57.

15.7.2. What Can Runlevels do For me?

Booting into a different runlevel can help you solve certain problems, for example: suppose you have made a change into your *X* configuration that has rendered it unusable, and you boot into it **by default**. If so, you can temporarily boot into a console, fix the error and reboot into *X*. Let's see how to do that.

By default *GNU/Linux* either boots to runlevel 3 (the console) or to runlevel 5 (*X*). The default runlevel is defined in the file */etc/inittab*. Look for an entry like *id:3:initdefault:* (if your system starts in a console) or one like *id:5:initdefault:* (if your system starts in *X*).

If you want to boot into a runlevel other than the one defined in */etc/inittab*, you have to define that runlevel on the bootprompt. Under *LILO*, type *linux init 3* to boot into the console or *linux init 5* to boot into *X*. Under *grub*, press the **E** key twice, add *init 3* to boot into the console, or *init 5* to boot into *X*, press the **ENTER** key and then the **B** key to boot.

15.8. Recovering Deleted Files

In this section we discuss some ways of recovering deleted files and directories. Please bear in mind that the recovery tools are not magical, and they will work depending on how recently you deleted the file you are trying to recover.

You might be wondering “Well, I accidentally deleted this file, how can I recover it?”. Don’t fear, there are some utilities designed for *GNU/Linux*’s ext2 file system which allow you to recover deleted files and directories. However, these utilities won’t recover the files you deleted a few months ago because of disk usage, space marked as “free” will be overwritten; so the **best** way to protect against accidental or not so accidental deletions is doing backups as described above.



Please bear in mind that there are not (as yet) tools to recover files deleted on ReiserFS file systems. Keep in touch with the ReiserFS homepage (<http://www.namesys.com>) for the latest news about ReiserFS.

Anyway, on to the tools for recovering your deleted files. One such tool is *Recover*. It is an “interactive” tool. If you are the proud owner of a Mandrake Linux - PowerPack Deluxe Edition, you already have this tool in the “contribs” CD-ROM. Otherwise, you can find it on the RPMFind web site (<http://www.rpmfind.net>). Go there and download the RPM. Once you have the RPM, install it. Then, run it with `recover [command_line_opts]` and answer the questions it asks you. The questions are for setting a time span to look for deleted files and directories to minimize the time it takes to do the search.⁵

Once the tool finishes its search, it will ask you where you want to save the recovered files and directories. Pick a directory of your choice, and you will have all the files and directories recovered in it. Note that you will not be able to recover the filenames, just their contents, but you can inspect them or try to rename them with different names until you get the right one. This is better than nothing.



There are also mini-*HOWTO*s related to undeletion for ext2, look at Ext2fs-Undeletion (<http://www.linuxdoc.org/HOWTO/mini/Ext2fs-Undeletion.html>) and undeletion of whole directory structures (<http://www.linuxdoc.org/HOWTO/mini/Ext2fs-Undeletion-Dir-Struct/index.html>).

15.9. Recovering From a System Freeze

When stuck in a “freeze”, your computer will not respond to commands anymore and input devices like keyboard and mouse seem to be blocked. This is a worst case scenario and could mean that you have a very severe error in either your configuration, your software or your hardware. Here we will show you to deal with this annoying situation.

In the case of a system freeze, your top priority should be trying to shutdown your system properly. Let’s assume you are under *X*, if so, try these steps consecutively:

- Try to kill the *X* server by pressing **ALT+CTRL+BACKSPACE** simultaneously.
- Try to switch to another console with **ALT+CTRL+F2**. If you succeed, login as root and issue the command: `kill -15 $(pidof X)` or the command `kill -9 $(pidof X)`, if the first command shows no effect. (Check with `top` to see if *X* is still running).
- If you are part of a local network, try to `ssh` into your machine from another box. It is advisable to `ssh` into the remote machine as an unprivileged user and then use `su` to become root.

5. You can search for **all** deleted files too, but it will take longer...

- If the system does not respond to any of these steps, you have to go through the “SysRq” (“System Request”) sequence. The “SysRq” sequence involves pressing three keys at once, the left **ALT** key, the **SysRq** key (labeled **PrintScreen** on older keyboards) and a letter key.
 - **Left ALT+SysRq+r** puts the keyboard in “raw” mode. Now try the pressing **ALT+CTRL+BACKSPACE** again, to kill the *X*. If that does not work, carry on.
 - **Left ALT+SysRq+s** attempts to write all unsaved data to disk (“sync” the disk).
 - **Left ALT+SysRq+e** sends a termination signal to all processes, except for *init*.
 - **Left ALT+SysRq+i** sends a kill signal to all processes, except for *init*.
 - **Left ALT+SysRq+u** attempts to remount all mounted filesystems read-only. This removes the “dirty flag” and will avoid a file system check upon reboot.
 - **Left ALT+SysRq+b** reboots the system. You might just as well press the “reset” button on your machine.



Remember that this is a sequence, i.e. you have to press one combination after the other in the right order: **R**aw, **S**ync, **tE**rm, **kIll**, **U**mount, **rE**boot⁶. You will find more on this feature in `/usr/src/linux/Documentation/sysrq.txt`.

- If none of the above helps, cross fingers and press the “reset” switch on your machine. If you are lucky, *GNU/Linux* will just run a disk check upon reboot.

By all means, try to find out what causes these lockups because they can do severe damage to the file system. You might also want to consider using ReiserFS, a journaling file system included in **Mandrake Linux** since 7.0, which handles such failures more gracefully. However, replacing ext2fs with ReiserFS requires reformatting your partitions.

15.10. Killing Misbehaved Apps

Well, this one is not so hard after all. Actually, it is not likely that you will need it but just in case you do... You have many ways to do it. You can do it by finding the PID of the program that has gone south, and use the `kill` to terminate it, or you can use the `xkill` tool or other graphical tools like the ones that show the process tree.

15.10.1. From The Console

The first thing to do to terminate a misbehaving program is to find its PID, or process ID. To do so, execute the following from a console: `ps aux | grep netscape`, supposing that *netscape* is the misbehaving program. You will get something like the following:

```
dodo      3505  7.7 23.1 24816 15076 pts/2  Z   21:29   0:02 /usr/lib/netscape
```

This tells us, among other things, that *netscape* was started by user *dodo* and has 3505 as its PID.

Now that we have the PID of the misbehaving program, we can proceed to execute the `kill` command to terminate it. So, we execute the following: `kill -9 3505`, and that's it! *netscape* will get killed. Note that this is **only** to be used when the program does not respond to your input anymore. **Do not** use it as a standard means to exit from applications.

Actually, what we have done was send the `KILL` signal to the process number 3505. The `kill` command accepts other signals besides `KILL`, so you can have greater control over your processes. For more info, see the man page for `kill` and the chapter “Process control”, page 33.

15.10.2. Using XKILL

Mandrake Linux includes a graphical kill tool: `xkill`. It is useful to terminate misbehaved graphical applications with a single click. All you have to do is click on `xkill`'s desktop icon and, when your mouse pointer changes to a skull and cross-bones, move it over the window of the application you want to kill and click on the window. That's it!



You can also access `xkill` with the following keyboard shortcut:
CTRL+ALT+ESC (at least from *KDE*).

15.10.3. Using Other Graphical Tools

You can also use the graphical process' status tools (like *KPM*, *KSySGuard*, and *GTOP* to name a few) which allow you to point to the process name and with one click send that process a signal or just kill that process.

15.11. Mandrake's Specific Troubleshooting Tools

Well, actually each administration tool (*Control Center*, *RpmDrake*, *drakgw*, *draknet*, *mandrakeupdate*, *drakbackup*, etc.) is a potential trouble fixing tool. You can use all these tools to revert configuration changes, to add or remove software, to update your system with the latest fixes from **MandrakeSoft**, etc.

15.12. Final Thoughts

Well, as you have seen there are many more ways to recover from an emergency than re-installing the whole system again.⁷ Sure, you need a little expertise in applying some of the techniques described in this chapter, but with a little practice you will gain such expertise. However, we hope that you will never need to really master this techniques ... although it does not hurt to know them. We hope that the instructions and examples given will be useful when you are in need. Good luck recovering from an emergency!

7. The usual way to fix things in certain other operating systems...

Appendix A. The GNU General Public License

The following text is the GPL license that applies to most programs found in **Mandrake Linux** distributions.

Version 2, June 1991 Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

A.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

A.2. Terms and conditions for copying, distribution and modification

- 0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only

if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

- 1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

- 2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

- 3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally dis-

tributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

- 4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- 5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
- 6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
- 7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

- 8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
- 9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

- 10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free

Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

- 11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- 12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix B. GNU Free Documentation License

B.1. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or proces-

sing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See Copyleft (<http://www.gnu.org/copyleft/>).

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.2. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software

Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossary

APM

Advanced Power Management. A feature used by some *BIOS* es in order to make the machine enter a standby state after a given period of inactivity. On laptops, APM is also responsible for reporting the battery status and, if it is supported, the estimated remaining battery life.

ASCII

American Standard Code for Information Interchange. The standard code used for storing characters, including control characters, on a computer. Many 8-bit codes (such as ISO 8859-1, the Linux default character set) contain ASCII as their lower half.

See Also: ISO 8859.

BSD

Berkeley Software Distribution. A *Unix* variant developed at the Berkeley University computing department. This version has always been considered more advanced technically than the others, and has brought many innovations to the computing world in general and to *Unix* in particular.

CHAP

Challenge-Handshake Authentication Protocol: protocol used by ISP s to authenticate their clients. In this scheme, a value is sent to the client (the machine who connects), the client calculates a *hash* from this value which it sends to the server, and the server compares the *hash* with the one it has calculated. It is different from PAP in that it re-authenticates on a periodic basis after the initial authentication.

See Also: PAP.

CIFS

Common Internet FileSystem The predecessor of the SMB filesystem, used on *DOS* systems.

DHCP

Dynamic Host Configuration Protocol. A protocol designed for machines on a local network to dynamically get an IP address from a DHCP server.

DMA

Direct Memory Access. A facility used on the *PC* architecture which allows for a peripheral to read or write from main memory without the help of the CPU . PCI peripherals use bus mastering and do not need DMA .

DNS

Domain Name System. The distributed name/address mechanism used in the Internet. This mechanism allows you to map a domain name to an IP address, which is what lets you look up a site by domain name without knowing the IP address of the site. DNS also allows reverse lookup, that is you can get a machine's IP address from its name.

DPMS

Display Power Management System. Protocol used by all modern monitors in order to manage power saving features. Monitors supporting these features are commonly called "green monitors".

ELF

Executable and Linking Format. This is the binary format used by most *GNU/Linux* distributions nowadays.

ext2

short for the "Extended 2 filesystem". This is *GNU/Linux* ' native filesystem and has all characteristics of any *Unix* filesystem: support for special files (character devices, symbolic links...), file permissions and ownership, and so on.

FAQ

Frequently Asked Questions. A document containing a series of questions/answers about a specific topic. Historically, FAQ s appeared in newsgroups, but this sort of document now appears on various web sites, and even commercial products have their FAQ . Generally, they are very good sources of information.

FAT

File Allocation Table. Filesystem used by *DOS* and *Windows* .

FDDI

Fiber Distributed Digital Interface. A high-speed network physical layer, which uses optical fiber for communication. Only used on big networks, mainly because of its price.

FHS

Filesystem Hierarchy Standard. A document containing guidelines for a coherent file tree organization on *Unix* systems. **Mandrake Linux** complies with this standard in most aspects.

FIFO

First In, First Out. A data structure or hardware buffer from which items are taken out in the order they were put in. *Unix* pipes are the most common examples of FIFO s.

FTP

File Transfer Protocol. This is the standard Internet protocol used to transfer files from one machine to another.

GFDL

The GNU Free Documentation License. It is the license that applies to all **Mandrake Linux** documentation.

GIF

Graphics Interchange Format. An image file format, widely used on the web . GIF images may be compressed or animated. Due to copyright problems it is a bad idea to use them, replace them as much as possible by the far advanced PNG format instead.

GNU

GNU's Not Unix. The GNU project has been initiated by Richard Stallman at the beginning of the 80s, and aimed at developing a free operating system ("free" as in "free speech"). Currently, all tools are there, except... the kernel. The GNU project kernel, *Hurd* , is not rock solid yet. *Linux* borrows, among others, two things from GNU : its *C* compiler, *gcc* , and its license, the GPL .

See Also: GPL.

GPL

General Public License. The license of the *GNU/Linux* kernel, it goes the opposite way of all proprietary licenses in that it gives no restriction as to copying, modifying and redistributing the software, as long as the source code is made available. The only restriction, if one can call it that, is that the persons to which you redistribute it must also benefit from the same rights.

GUI

Graphical User Interface. Interface to a computer consisting of windows with menus, buttons, icons and so on. The vast majority prefer a GUI over a CLI (*Command Line Interface*) for ease of use, even though the latter is more versatile.

HTML

HyperText Markup Language. The language used to create web documents.

HTTP

HyperText Transfer Protocol. The protocol used to connect to websites and retrieve HTML documents or files.

IDE

Integrated Drive Electronics. The most widely used bus on today's *PC* s for hard disks. An IDE bus can contain up to two devices, and the speed of the bus is limited by the device on the bus which has the slower command queue (and not the slower transfer rate!).

See Also: ATAPI.

IP masquerading

is when you use a firewall to hide your computer's true IP address from the outside. Typically any outside network connections you make beyond the firewall will inherit the firewall's IP address. This is useful in situations where you may have a fast Internet connection with only one IP address but wish to use more than one computer that have internal network IP addresses assigned.

IRC

Internet Relay Chat. One of the few Internet standards for live speech. It allows for channel creation, private talks, and also file exchange. It is also designed to be able to make servers connect to each other, which is why several IRC networks exist today: **Undernet**, **DALnet**, **EFnet** to name a few.

IRC channels

are the "places" inside IRC servers where you can chat with other people. Channels are created in IRC servers and users join those channels so they can communicate with each other. Messages written on an

channel are only visible to those people connected to that channel. Two or more users can also create a “private” channel so they don’t get disturbed by other users. Channel names begin with a #.

ISA

Industry Standard Architecture. The very first bus used on *PC* s, it is slowly being abandoned in favor of the PCI bus. Some hardware manufacturers still use it, though. It is still very common that SCSI cards supplied with scanners, CD writers, ... are ISA . Too bad.

ISDN

Integrated Services Digital Network. A set of communication standards for allowing a single wire or optical fiber to carry voice, digital network services and video. It has been designed in order to eventually replace the current phone system, known as PSTN (*Public Switched Telephone Network*) or POTS (*Plain Ole Telephone Service*). Technically ISDN is a circuit switched data network.

ISO

International Standards Organization. A group of companies, consultants, universities and other sources which enumerates standards in various topics, including computing. The papers describing standards are numbered. The standard number iso9660, for example, describes the filesystem used on CD-ROM s.

ISP

Internet Service Provider. A company which sells Internet access to its customers, whether the access is over telephone lines or dedicated lines.

JPEG

Joint Photographic Experts Group. Another very common image file format. JPEG is mostly suited for compressing real-world scenes, and does not work very well on non-realistic images.

LAN

Local Area Network. Generic name given to a network of machines connected to the same physical wire.

LDP

Linux Documentation Project. A nonprofit organization which maintains *GNU/Linux* documentation. Its mostly known documents are *HOWTOs* , but it also maintains FAQ s, and even a few books.

MBR

Master Boot Record. Name given to the first sector of a bootable hard drive. The MBR contains the code used to load the operating system into memory or a bootloader (such as *LILO*), and the partition table of that hard drive.

MIME

Multipurpose Internet Mail Extensions. A string of the form type/subtype describing the contents of a file attached in an e-mail. This allows MIME -aware mail clients to define actions depending on the type of the file.

MPEG

Moving Pictures Experts Group. An ISO committee which generates standards for video and audio compression. MPEG is also the name of their algorithms. Unfortunately, the license for this format is very restrictive, and as a consequence there are still no *Open Source* MPEG players...

NCP

NetWare Core Protocol. A protocol defined by **Novell** to access Novell NetWare file and print services.

NFS

Network FileSystem. A network filesystem created by **Sun Microsystems** in order to share files across a network in a transparent way.

NIC

Network Interface Controller. An adapter installed in a computer which provides a physical connection to a network, such as an *Ethernet* card.

NIS

Network Information System. NIS was also known as “Yellow Pages”, but **British Telecom** holds a copyright on this name. NIS is a protocol designed by **Sun Microsystems** in order to share common information across a NIS **domain**, which can gather a whole LAN , part of this LAN or several LAN s. It can export password databases, service databases, groups information and more.

PAP

Password Authentication Protocol. A protocol used by many ISP s to authenticate their clients. In this scheme, the client (you) sends an identifier/password pair to the server, which is not encrypted.
See Also: CHAP.

PCI

Peripheral Components Interconnect. A bus created by **Intel** and which is today the standard bus for *PC* architectures, but other architectures use it too. It is the successor of ISA , and it offers numerous services: device identification, configuration information, IRQ sharing, bus mastering and more.

PCMCIA

Personal Computer Memory Card International Association. More and more commonly called "PC Card" for simplicity reasons, this is the standard for external cards attached to a laptop: modems, hard disks, memory cards, *Ethernet* cards, and more. The acronym is sometimes humorously expanded to *People Cannot Memorize Computer Industry Acronyms...*

PNG

Portable Network Graphics. Image file format created mainly for web use, it has been designed as a patent-free replacement for GIF and also has some additional features.

Plug'N'Play

Plug'N'Play. First an add-on for ISA in order to add configuration information for devices, it has become a more widespread term which groups all devices able to report their configuration parameters. As such, all PCI devices are Plug'N'Play.

POP

Post Office Protocol. The common protocol used for retrieving mail from an ISP .

PPP

Point to Point Protocol. This is the protocol used to send data over serial lines. It is commonly used to send IP packets to the Internet, but it can also be used with other protocols such as Novell's IPX protocol.

RAID

Redundant Array of Independent Disks. A project initiated at the computing science department of Berkeley University, in which the storage of data is spread along an array of disks using different schemes. At first, this was implemented using floppy drives, which is why the acronym originally stood for *Redundant Array of Inexpensive Disks*.

RAM

Random Access Memory. Term used to identify a computer's main memory. The "Random" here means that any part of the memory can be directly accessed...

RFC

Request For Comments. RFC s are the official Internet standard documents, published by the IETF (*Internet Engineering Task Force*). They describe all protocols, their usage, their requirements and so on. When you want to learn how a protocol works, pick up the corresponding RFC .

RPM

Redhat Package Manager. A packaging format developed by **Red Hat** in order to create software packages, it is used in many *GNU/Linux* distributions, including **Mandrake Linux** .

SCSI

Small Computers System Interface. A bus with a high throughput designed to allow for several types of peripherals. Unlike IDE , a SCSI bus is not limited by the speed at which the peripherals accept commands. Only high-end machines integrate a SCSI bus directly on the motherboard, *PC* s need add-on cards.

SMB

Server Message Block. Protocol used by *Windows* machines (9x or NT) for file and printer sharing across a network.
See Also: CIFS.

SMTP

Simple Mail Transfer Protocol. This is the common protocol for transferring email. Mail Transfer Agents such as *sendmail* or *postfix* use SMTP . They are sometimes also called SMTP servers.

SVGA

Super Video Graphics Array. The video display standard defined by VESA for the *PC* architecture. The resolution is 800x 600 x 16 colors.

TCP

Transmission Control Protocol. This is the most common reliable protocol that uses IP to transfer network packets. TCP adds the necessary checks on top of IP to make sure that packets are delivered. Unlike UDP, TCP works in connected mode, which means that two machines must establish a connection before exchanging data.

URL

Uniform Resource Locator. A string with a special format used to identify a resource on the Internet in a unique way. The resource can be a file, a server or other item. The syntax for a URL is `protocol://server.name[:port]/path/to/resource`.

When only a machine name is given and the protocol is `http://`, it defaults to retrieving the file `index.html` on the server.

VESA

Video Electronics Standards Association. An industry standards association aimed at the *PC* architecture. It is the author of the SVGA standard, for example.

WAN

Wide Area Network. This network, although similar to a LAN connects computers on a network that is not physically connected to the same wires and are separated by a greater distance.

account

on a *Unix* system, the combination of a name, a personal directory, a password and a *shell* which allows a person to connect to this system.

alias

mechanism used in a *shell* in order to make it substitute one string for another before executing the command. You can see all aliases defined in the current session by typing `alias` at the prompt.

arp

Address Resolution Protocol. The Internet protocol used to dynamically map an Internet address to physical (hardware) addresses on local area networks. This is limited to networks that support hardware broadcasting.

assembly language

is the programming language that is closest to the computer, which is why it's called a "low level" programming language. Assembly has the advantage of speed since assembly programs are written in terms of processor instructions so little or no translation is needed when generating executables. Its main disadvantage is that it is processor (or architecture) dependent. Writing complex programs is very time-consuming as well. So, assembly is the fastest programming language, but it isn't portable between architectures.

ATAPI

("AT Attachment Packet Interface") An extension to the ATA specification ("Advanced Technology Attachment", more commonly known as IDE, *Integrated Drive Electronics*) which provides additional commands to control CDROM drives and magnetic tape drives. IDE controllers equipped with this extension are also referred to as EIDE (*Enhanced IDE*) controllers.

ATM

This is an acronym for **Asynchronous Transfer Mode**. An ATM network packages data into standard size blocks (53 bytes: 48 for the data and 5 for the header) which it can convey efficiently from point to point. ATM is a circuit switched packet network technology oriented towards high speed (multi-megabits) optical networks.

atomic

a set of operations is said to be atomic when it executes all at once, and cannot be preempted.

background

in *shell* context, a process is running in the background if you can type commands while said process is running.

See Also: job, foreground.

backup

is a means of saving your important data to a safe medium and location. Backups should be done regularly, especially with more critical information and configuration files (the prime directories to backup are `/etc`, `/home` and `/usr/local`). Traditionally, many people use `tar` with `gzip` or `bzip2` to backup directories and files. You can use these tools or programs like `dump` and `restore`, along with many other free or commercial backup solutions.

batch

is a processing mode where jobs are submitted to the processor, and then the processor executes them one after the other till it executes the last one and it's ready for another list of processes.

beep

is the little noise your computer's speaker does to warn you of some ambiguous situation when you're using command completion and, for example, there's more than one possible choice for completion. There might be other programs that make beeps to let you know of some particular situation.

beta testing

is the name given to the process of testing the beta version of a program. Programs usually get released in alpha and beta states for testing prior to final release.

bit

stands for *BI*nary *di*giT. A single digit which can take the values 0 or 1, because calculation is done in base two.

block mode files

files whose contents are buffered. All read/write operations for such files go through buffers, which allows for asynchronous writes on the underlying hardware, and for reads, not to read again what is already in a buffer.

See Also: buffer, buffer cache, character mode files.

boot

the procedure taking place when a computer is switched on, where peripherals are recognized one after the other, and where the operating system is loaded into memory.

bootdisk

a bootable floppy disk containing the code necessary to load the operating system from the hard disk (sometimes it is self-sufficient).

bootloader

is a program that starts the operating system. Many bootloaders give you the opportunity to load more than one operating system by letting you choose between them at a boot menu. Bootloaders like *grub* are popular because of this feature and are very useful in dual- or multi-boot systems.

buffer

a small portion of memory with a fixed size, which can be associated with a block mode file, a system table, a process and so on. The coherency of all buffers is maintained by the buffer cache.

See Also: buffer cache.

buffer cache

a crucial part of an operating system kernel, it is in charge of keeping all buffers up-to-date, shrinking the cache when needed, clearing unneeded buffers and more.

See Also: buffer.

bug

illogical or incoherent behavior of a program in a special case, or a behavior which does not follow the documentation or accepted standards issued for the program. Often, new features introduce new bugs in a program. Historically, this term comes from the old days of punch cards: a bug (the insect!) slipped into a hole of a punch card and, as a consequence, the program misbehaved. Ada Lovelace, having discovered this, declared "It's a bug!", and since then the term has remained.

byte

eight consecutive bits, interpreted in base two as a number between 0 and 255.

See Also: bit.

case

when taken in the context of strings, the case is the difference between lowercase letters and uppercase (or capital) letters.

character mode files

files whose content is not buffered. When associated to physical devices, all input/output on these devices is performed immediately. Some special character devices are created by the operating system (`/dev/zero`, `/dev/null` and others). They correspond to data flows.

See Also: block mode files.

client

program or computer that periodically connects to another program or computer to give it orders or ask for information. In the case of **peer to peer** systems such as SLIP or PPP the client is taken to be the end that initiates the connection and the remote end, being called, is taken to be the server. It is one of the components of a **client/server system**.

client/server system

system or protocol consisting of a **server** and one or several **clients**.

command line

what is provided by a shell and allows the user to type commands directly. Also subject of an eternal “flame war” between its supporters and its detractors :-)

command mode

under *Vi* or one of its clones, it is the state of the program in which pressing a key (this above all regards letters) will not insert the character in the file being edited, but instead perform an action specific to the said key (unless the clone has remappable commands and you have customized your configuration). You may get out of it typing one of the “back to insertion mode” commands: **i**, **I**, **a**, **A**, **s**, **S**, **o**, **O**, **c**, **C**, ...

compilation

is the process of translating source code that is human readable (well, with some training) and that is written in some programming language (*C*, for example) into a binary file that is machine readable.

completion

ability of a *shell* to automatically expand a substring to a filename, user name or other, as long as there is a match.

compression

is a way to shrink files or decrease the number of characters sent over a communications connection. Some file compression programs include *compress*, *zip*, *gzip*, and *bzip2*.

console

is the name given to what used to be called terminals. They were the users machines (a screen plus a keyboard) connected to one big central mainframe. On *PC* s, the physical terminal is the keyboard and screen.

See Also: virtual console.

cookies

temporary files written on the local hard disk by a remote web server. It allows for the server to be aware of a user's preferences when this user connects again.

datagram

A datagram is a discrete package of data and headers which contain addresses, which is the basic unit of transmission across an IP network. You might also hear this called a “packet”.

dependencies

are the stages of compilation that need to be satisfied before going on to other compilation stages in order to successfully compile a program.

desktop

If you're using the X Window System, the desktop is the place on the screen inside which you work and upon which your windows and icons are displayed. It is also called the background, and is usually filled with a simple color, a gradient color or even an image.

See Also: virtual desktops.

directory

Part of the filesystem structure. Within a directory, files or other directories are stored. Sometimes there are sub-directories (or branches) within a directory. This is often referred to as a directory tree. If you want to see what's inside another directory, you will either have to list it or change to it. Files inside a directory are referred to as leaves while sub-directories are referred to as branches. Directories follow the same restrictions as files although the permissions mean different things. The special directories `.` and `..` refer to the directory itself and to the parent directory respectively.

discrete values

are values that are non-continuous. That is, there's some kind of "spacing" between two consecutive values.

distribution

is a term used to distinguish one *GNU/Linux* manufacturers product from another. A distribution is made up of the core Linux kernel and utilities, as well as installation programs, third-party programs, and sometimes proprietary software.

DLCI

The DLCI is the Data Link Connection Identifier and is used to identify a unique virtual point to point connection via a Frame Relay network. The DLCI's are normally assigned by the Frame Relay network provider.

echo

is when the characters you type in a username entry field, for example, are shown "as is", instead of showing "*" for each one you type.

editor

is a term typically used for programs that edit text files (aka text editor). The most well-known *GNU/Linux* editors are the GNU Emacs (*Emacs*) editor and the *Unix* editor *Vi*.

email

stands for Electronic Mail. This is a way to send messages electronically between people on the same network. Similar to regular mail (aka snail mail), email needs a destination and sender address to be sent properly. The sender must have an address like "sender@senders.domain" and the recipient must have an address like "recipient@recipients.domain." Email is a very fast method of communication and typically only takes a few minutes to reach anyone, regardless of where in the world they are located. In order to write email, you need an email client like *pine* or *mutt* which are text-mode clients, or GUI clients like *kmail*.

environment

is the execution context of a process. It includes all the information that the operating system needs to manage the process and what the processor needs to execute the process properly.

See Also: process.

environment variables

a part of a process' environment. Environment variables are directly viewable from the *shell*.

See Also: process.

escape

in the shell context, is the action of surrounding some string between quotes to prevent the shell from interpreting that string. For example, when you need to use spaces in some command line and pipe the results to some other command you have to put the first command between quotes ("escape" the command) otherwise the shell will interpret it wrong and won't work as expected.

filesystem

scheme used to store files on a physical media (hard drive, floppy) in a consistent manner. Examples of filesystems are FAT, *GNU/Linux*' ext2fs, iso9660 (used by CD-ROMs) and so on. An example of a virtual filesystem is the `/proc` filesystem.

firewall

a machine or a dedicated piece of hardware which, in the topology of a local network, is the unique connecting point to the outside network, and which filters, or controls the activity on some ports, or makes sure only some specific interfaces may have access to them.

flag

is an indicator (usually a bit) which is used to signal some condition to a program. For example, a filesystem has, among others, a flag indicating if it has to be dumped in a backup, so when the flag is active the filesystem gets backed up, and when it's inactive it doesn't.

focus

the state for a window to receive keyboard events (such as key-presses, key-releases and mouse clicks) unless they are trapped by the window manager.

foreground

in shell context, the process in the foreground is the one which is currently running. You have to wait for such a process to finish in order to be able to type commands again.

See Also: job, background.

Frame Relay

Frame Relay is a network technology ideally suited to carrying traffic that is of bursty or sporadic nature. Network costs are reduced by having many Frame Relay customer sharing the same network capacity and relying on them wanting to make use of the network at slightly different times.

framebuffer

projection of a video card's RAM into the machine's address space. This allows for applications to access the video RAM without the chore of having to talk to the card. All high-end graphical workstations use framebuffers, for example.

full-screen

This term is used to refer to applications that take up the whole visible area of your display.

gateway

link connecting two IP networks.

globbing

in the *shell*, the ability to group a certain set of filenames with a globbing pattern.

See Also: globbing pattern.

globbing pattern

a string made of normal characters and special characters. Special characters are interpreted and expanded by the *shell*.

hardware address

This is a number that uniquely identifies a host in a physical network at the media access layer. Examples of this are **Ethernet Addresses** and **AX.25 Addresses**.

hidden file

is a file which can't be "seen" when doing a `ls` command with no options. Hidden files' filenames begin with a `.` and are used to store the user's personal preferences and configurations for the different programs (s)he uses. For example, *bash*'s command history is saved into `.bash_history`, which is a hidden file.

home directory

often abbreviated by "home", this is the name for the personal directory of a given user.

See Also: account.

host

refers to a computer and is commonly used when talking about computers that are connected on a network.

icon

is a little drawing (normally sized 16x 16, 32x 32, 48x 48 and sometimes 64x 64 pixels) which represents, under a graphical environment, a document, a file or a program.

inode

entry point leading to the contents of a file on a *Unix*-like filesystem. An inode is identified in a unique way by a number, and contains meta-information about the file it refers to, such as its access times, its type, its size, **but not its name!**

insert mode

under *Vi* or one of its clones, it is the state of the program in which pressing a key will insert that character in the file being edited (except pathological cases like the completion of an abbreviation, right justify at the end of the line, ...). One gets out of it pressing the key **Esc** (or **Ctrl-I**).

Internet

is a huge network that connects computers around the world.

IP address

is a numeric address consisting of four parts which identifies your computer on the Internet. IP addresses are structured in a hierarchical manner, with top level and national domains, domains, subdomains and each machine's personal address. An IP address would look something like 192.168.0.1. A machine's personal address can be one of two types: static or dynamic. Static IP addresses are addresses that never change, but rather are permanent. Dynamic IP addresses mean your IP address will change with each new connection to the network. Dial-up and cable modem users typically have dynamic IP addresses while some DSL and other high-speed connections provide static IP addresses.

ISO 8859

The ISO 8859 standard includes several 8-bit extensions to the ASCII character set. Especially important is ISO 8859-1, the "Latin Alphabet No. 1", which has become widely implemented and may already be seen as the de facto standard ASCII replacement.

ISO 8859-1 supports the following languages: Afrikaans, Basque, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Scottish, Spanish, and Swedish.

Note that the ISO 8859-1 characters are also the first 256 characters of ISO 10646 (Unicode). However, it lacks the EURO symbol and does not fully cover Finnish and French. ISO 8859-15 is a modification of ISO 8859-1 that covers these needs.

See Also: ASCII.

job

in *shell* context, a job is a process running in the background. You can have several jobs in the same shell and control these jobs.

See Also: foreground, background.

kernel

is the guts of the operating system. The kernel is responsible for allocating resources and separating processes from each other. It handles all of the low-level operations that allow programs to talk directly to the hardware on your computer, manages the buffer cache and so on.

kill ring

under *Emacs*, it is the set of text areas cut or copied since the beginning of the editor, which may be recalled to be inserted again, and which is organized like a ring.

launch

is the action of invoking, or starting, a program.

library

is a collection of procedures and functions in binary form to be used by programmers in their programs (as long as the library's license allows them to do so). The program in charge of loading shared libraries at run time is called the dynamic linker.

link

reference to an inode in a directory, therefore giving a (file) name to the inode. Examples of inodes which don't have a link (and hence have no name) are: anonymous pipes (as used by the shell), sockets (aka network connections), network devices and so on.

linkage

last stage of the compile process, which consists in linking together all object files in order to produce an executable file, and matches unresolved symbols with dynamic libraries (unless a static linkage has been asked, in which case the code of these symbols will be included in the executable).

Linux

is a *Unix*-like operating system which runs on a variety of different computers, and is free for anyone to use and modify. Linux (the kernel) was written by Linus Torvalds.

login

connection name for a user on a *Unix* system, and the action to connect.

lookup table

is a table that puts in correspondance codes (or tags) and their meaning. It is often a data file used by a program to get further information about a particular item.

For example, *harddrake* uses such a table to know what a manufacturer's product code means. This is one line from the table, giving information about item CTL0001

```
CTL0001 sound    sb      Creative Labs    SB16 \
                HAS_OPL3|HAS_MPU401|HAS_DMA16|HAS_JOYSTICK
```

loopback

virtual network interface of a machine to itself, allowing the running programs not to have to take into account the special case where two network entities are in fact the same machine.

major

number specific to the device class.

manual page

a small document containing the definition of a command and its usage, to be consulted with the `man` command. The first thing one should (learn how to) read when hearing of a command he doesn't know :-)

minor

number identifying the specific device we are talking about.

mount point

is the directory where a partition or another device is attached to the *GNU/Linux* filesystem. For example, your CD-ROM is mounted in the `/mnt/cdrom` directory, from where you can explore the contents of any mounted CD s.

mounted

A device is mounted when it is attached to the *GNU/Linux* filesystem. When you mount a device you can browse its contents. This term is partly obsolete as with the "supermount" feature, users do not need any more to manually mount removable medias.

See Also: mount point.

MSS

The Maximum Segment Size (**MSS**) is the largest quantity of data that can be transmitted at one time. If you want to prevent local fragmentation MSS would equal MTU-IP header.

MTU

The Maximum Transmission Unit (**MTU**) is a parameter that determines the largest datagram than can be transmitted by an IP interface without it needing to be broken down into smaller units. The MTU should be larger than the largest datagram you wish to transmit unfragmented. Note, this only prevents fragmentation locally, some other link in the path may have a smaller MTU and the datagram will be fragmented there. Typical values are 1500 bytes for an ethernet interface, or 576 bytes for a SLIP interface.

multitasking

the ability for an operating system to share CPU time between several processes. At low level, this is also known as multiprogramming. Switching from one process to another requires that all the current process context be saved and restored when this process is elected again. This operation is called context switch, and on Intel, is done 100 times per second; therefore it's fast enough so that a user has the illusion that the operating system runs several applications at the same time. There are two types of multitasking: preemptive multitasking is where the operating system is responsible for taking away the CPU and pass it to another process; cooperative multitasking is where the process itself gives back the CPU. The first variant is, obviously, the better choice because no program can consume the entire CPU time and block other processes. *GNU/Linux* does preemptive multitasking. The policy to select which process should be run, depending on several parameters, is called scheduling.

multiuser

is used to describe an operating system which allows multiple users to log into and use the system at the exact same time, each being able to do their own work independent of other users. A multitasking operating system is required to provide multiuser support. *GNU/Linux* is both a multitasking and multiuser operating system, as any *Unix* system for that matter.

named pipe

a *Unix* pipe which is linked, as opposed to pipes used in shells.

See Also: pipe, link.

naming

a word commonly used in computing for a method to identify objects. You will often hear of “naming conventions” for files, functions in a program and so on.

newsgroups

discussion and news areas that can be accessed by a news or USENET client to read and write messages specific to the topic of the newsgroup. For example, the newsgroup `alt.os.linux.mandrake` is an alternate newsgroup (alt) dealing with the Operating System (os) *GNU/Linux*, and specifically, **Mandrake Linux** (mandrake). Newsgroups are broken down in this fashion to make it easier to search for a particular topic.

null, character

the character or byte number 0, it is used to mark the end of a string.

object code

is the code generated by the compilation process to be linked with other object codes and libraries to form an executable file. Object code is machine readable.

See Also: compilation, linkage.

on the fly

Something is said to be done “on the fly” when it’s done along with something else, without you noticing it or explicitly asking for it.

open source

is the name given to free source code of a program that is made available to development community and public at large. The theory behind this is that allowing source code to be used and modified by a broader group of programmers will ultimately produce a more useful product for everyone. Some popular open source programs include *Apache*, *sendmail* and *GNU/Linux*.

operating system

is the interface between the applications and the underlying hardware. The tasks for any operating system are primarily to manage all of the machine specific resources. On a *GNU/Linux* system, this is done by the kernel and loadable modules. Other well-known operating systems include *AmigaOS*, *MacOS*, *FreeBSD*, *OS/2*, *Unix*, *Windows NT*, and *Windows 9x*.

owner

in the context of users and their files, the owner of a file is the user who created that file.

owner group

in the context of groups and their files, the owner group of a file is the group to which the user who created that file belongs to.

pager

program displaying a text file one screenful at a time, and making it easy to move back and forth and search for strings in this file. We advise you to use `less`.

password

is a secret word or combination of words or letters that is used to secure something. Passwords are used in conjunction with user logins to multi-user operating systems, web sites, FTP sites, and so forth. Passwords should be hard-to-guess phrases or alphanumeric combinations, and should never be based on common dictionary words. Passwords ensure that other people cannot log into a computer or site with your account.

patch, to patch

file holding a list of corrections to issue to a source code in order to add new features, to remove bugs, or to modify it according to one’s wishes and needs. The action consisting of the application of these corrections to the archive of source code (aka “patching”).

path

is an assignment for files and directories to the filesystem. The different layers of a path are separated by the “slash” or `’/’` character. There are two types of paths on *GNU/Linux* systems. The **relative** path is the

position of a file or directory in relation to the current directory. The **absolute** path is the position of a file or directory in relation to the root directory.

pipe

a special *Unix* file type. One program writes data into the pipe, and another program reads the data at the other end. *Unix* pipes are FIFO s, so the data is read at the other end in the order it was sent. Very widely used with the shell. See also **named pipe**.

pixmap

is an acronym for “pixel map”. It’s another way of referring to bitmapped images.

plugin

add-on program used to display or play some multimedia content found on a web document. It can usually be easily downloaded if your browser is not yet able to display or play that kind of information.

porting

a program is translating that program in such a way that it can be used in a system it was not originally intended for, or it can be used in “similar” systems. For example, to be able to run a *Windows* -native program under *GNU/Linux* (natively), it must first be ported to *GNU/Linux* .

precedence

dictates the order of evaluation of operands in an expression. For example: If you have $4 + 3 * 2$ you get 10 as the result, since the product has more precedence than the addition. If you want to evaluate the addition first, then you have to add parenthesis like this $(4 + 3) * 2$, and you get 14 as the result since the parenthesis have more precedence than the addition and the product, so the operations in parenthesis get evaluated first.

preprocessors

are compilation directives that instruct the compiler to replace those directives for code in the programming language used in the source file. Examples of *C* ’s preprocessors are `#include`, `#define`, etc.

process

in the operating system context, a process is an instance of a program being executed along with its environment.

prompt

in a *shell* , this is the string before the cursor. When you see it, you can type your commands.

protocol

Protocols organize the communication between different machines across a network, either using hardware or software. They define the format of transferred data, whether one machine controls another, etc. Many well-known protocols include HTTP, FTP, TCP, and UDP.

proxy

a machine which sits between a network and the Internet , whose role is to speed up data transfers for the most widely used protocols (HTTP and FTP , for example). It maintains a cache of previous demands, which avoids the cost of asking for the file again if another machine asks for the same thing. Proxies are very useful on low bandwidth networks (such as modem connections). Sometimes the proxy is the only machine able to access outside the network.

pulldown menu

it is a menu that is “rolled” with a button in some of its corners. When you press that button, the menu “unrolls” itself showing you the full menu.

quota

is a method for restricting disk usage and limits for users. Administrators can restrict the size of home directories for a user by setting quota limits on specific filesystems.

read-only mode

for a file means that the file cannot be written to. You can read its contents but you can’t modify them.
See Also: read-write mode.

read-write mode

for a file, it means that the file can be written to. You can read its contents and modify them.
See Also: read-only mode.

regular expression

a powerful theoretical tool which is used to search and match text strings. It lets one specify patterns these strings must obey. Many *Unix* utilities use it: *sed* , *awk* , *grep* , *perl* among others.

root

is the superuser of any *Unix* system. Typically root (aka the system administrator) is the person responsible for maintaining and supervising the *Unix* system. This person also has complete access to everything on the system.

root directory

This is the top level directory of a filesystem. This directory has no parent directory, thus *'..'* for root points back to itself. The root directory is written as *'/'*.

root filesystem

This is the top level filesystem. This is the filesystem where *GNU/Linux* mounts its root directory tree. It is necessary for the root filesystem to reside in a partition of its own, as it is the basis for the whole system. It holds the root directory.

route

Is the path that your datagrams take through the network to reach their destination. Is the path between one machine and another in a network.

run level

is a configuration of the system software that only allows certain selected processes to exist. Allowed processes are defined, for each runlevel, in the file */etc/inittab*. There are eight defined runlevels: 0, 1, 2, 3, 4, 5, 6, S and switching among them can only be achieved by a privileged user by means of executing the commands *init* and *telinit*.

script

shell scripts are sequences of commands to be executed as if they were entered in the console one after the other. *shell* scripts are *Unix* 's (somewhat) equivalent of *DOS* batch files.

security levels

Mandrake Linux 's unique feature that allows you to set different levels of restrictions according to how secure you want to make your system. There are 6 predefined levels ranging from 0 to 5, where 5 is the tightest security. You can also define your own security level.

server

program or computer that provides a feature or service and awaits the connections from **clients** to execute their orders or give them the information they ask. In the case of **peer to peer** systems such as **slip** or **ppp** the server is taken to be the end of the link that is called and the end calling is taken to be the client. It is one of the components of a **client/ server system**.

shadow passwords

a password management suite on *Unix* systems in which the file containing the encrypted passwords is not world-readable, whereas it is when using the normal password system. It also offers other features such as password aging.

shell

The *shell* is the basic interface to the operating system kernel and is what provides the command line where users enter commands to run programs and system commands. All shells provide a scripting language which can be used to automate tasks or simplify often-used complex tasks. These *shell* scripts are similar to batch files from the *DOS* operating system, but are much more powerful. Some example shells are *bash* , *sh* , and *tcsh* .

single user

is used to describe a state of an operating system, or even an operating system itself, that only allows a single user to log into and use the system at any time.

site dependent

means that the information used by programs like *imake* and *make* to compile some source file depends on the site, the computer architecture, the computer's installed libraries, and so on.

socket

file type corresponding to any network connection.

soft links

see “symbolic links”.

standard error

the file descriptor number 2, opened by every process, used by convention to print error messages to the terminal screen by default.

See Also: standard input, standard output.

standard input

the file descriptor number 0, opened by every process, used by convention as the file descriptor from which the process receives data.

See Also: standard error, standard output.

standard output

the file descriptor number 1, opened by every process, used by convention as the file descriptor in which the process prints its output.

See Also: standard error, standard input.

streamer

is a device that takes “streams” (not interrupted or divided in shorter chunks) of characters as its input. A typical streamer is a tape drive.

switch

Switches are used to change the behavior of programs, and are also called command-line options or arguments. To determine if a program has optional switches that can be used, read the *man* pages or try to pass the `--help` switch to the program (ie. `program --help`).

symbolic links

special files, containing nothing but a string that makes reference to another file. Any access to them is the same as accessing the file whose name is the referenced string, which may or may not exist, and the path to which can be given in a relative or an absolute way.

target

is the object of compilation, i.e. the binary file to be generated by the compiler.

telnet

creates a connection to a remote host and allows you to log into the machine, provided you have an account. Telnet is the most widely-used method of remote logins, however there are better and more secure alternatives, like `ssh`.

theme-able

a graphical application is theme-able if it is able to change its appearance in real time. Many window managers are theme-able as well.

traverse

for a directory on a *Unix* system, this means that the user is allowed to go through this directory, and possibly to directories under it. This requires that the user has the execute permission on this directory.

username

is a name (or more generally a word) that identifies a user in a system. Each username is attached to a unique and single UID (user ID)

See Also: login.

variables

are strings that are used in *Makefile* files to be replaced by their value each time they appear. Usually they are set at the beginning of the *Makefile*. They are used to simplify *Makefile* and source files tree management.

More generally, variables in programming, are words that refer to other entities (numbers, strings, tables, etc.) that are likely to vary while the program is executing.

verbose

For commands, the verbose mode means that the command reports to standard (or possibly error) output all the actions it performs and the results of those actions. Sometimes, commands have a way to define the “verbosity level”, which means that the amount of information that the command will report can be controlled.

virtual console

is the name given to what used to be called terminals. On *GNU/Linux* systems, you have what are called virtual consoles which enable you to use one screen or monitor for many independently running sessions. By default, you have six virtual consoles which can be reached by pressing **ALT-F1** through **ALT-F6**. There is a seventh virtual console by default, **ALT-F7**, which will permit you to reach a running X Window System. In X, you can reach the text console by pressing **CTRL-ALT-F1** through **CTRL-ALT-F6**.
See Also: console.

virtual desktops

In the X Window System, the window manager may provide you several desktops. This handy feature allows you to organize your windows, avoiding the problem of having dozens of them stacked on top of each other. It works as if you had several screens. You can switch from one virtual desktop to another in a manner that depends on the window manager you're using.

See Also: window manager, desktop.

wildcard

The '*' and '?' characters are used as wildcard characters and can represent anything. The '*' represents any number of characters, including no characters. The '?' represents exactly one character. Wildcards are often used in regular expressions.

window

In networking, the **window** is the largest amount of data that the receiving end can accept at a given point in time.

window manager

the program responsible for the "look and feel" of a graphical environment, dealing with window bars, frames, buttons, root menus, and some keyboard shortcuts. Without it, it would be hard or impossible to have virtual desktops, to resize windows on the fly, to move them around, ...

Index

- .bashrc, 9
- account, 1
- applications
 - ImageMagick, 14
 - terminals, 14
- attribute
 - file, 11
- character
 - globbing, 12
- characters
 - special, 15
- collating order, 13
- command
 - cat, 6
 - cd, 5
 - init, 57
 - less, 7, 13
 - ls, 7
 - mount, 42
 - pwd, 5
 - sed, ??
 - umount, 42
 - wc, 13
- command line
 - completion, 14
- command line
 - introduction, 9
- commands
 - at, 28
 - bzip2, 30, 83
 - chgrp, 11
 - chmod, 11
 - chown, 11
 - configure, 85
 - cp, 10
 - crontab, 27
 - find, 26
 - grep, 25
 - gzip, 30
 - make, 87
 - mkdir, 9
 - msec, 73
 - mv, 10
 - patch, 96
 - rm, 9
 - rmdir, 9
 - tar, 29, 83
 - touch, 9
- console, 1
- contributors page, ii
- conventions
 - typing, iii
- directory
 - copying, 10
 - creating, 9
 - deleting, 9
 - moving, 10
 - renaming, 10
- Docbook, iii
- documentation, ii

- donation, ii
- editor
 - Emacs, 17
 - vi, 19
- environment
 - process, 53
 - variable, 6
- FHS, 39
- file
 - attribute, 11, 51
 - block mode, 47, 50
 - character mode, 47
 - copying, 10
 - creating, 9
 - deleting, 9
 - link, 47, 48
 - moving, 10
 - permission, 78
 - renaming, 10
 - socket, 47
- framebuffer, 100
- Free Software Foundation, i
- GFDL, 121
- GID, 2
- globbing
 - character, 12
- GNU Free Documentation License, i
- GNU/Linux, 1
- GPL, 117
- group, 1
 - change, 11
- grub, 99
- inode, 47, 47
- internationalization, ii
- lilo, 100
- link
 - hard, 51
 - symbolic, 50
- Makefile, 82, 88
- Mandrake
 - Mailing Lists, i
 - Mandrake Forum, i
 - Mandrake Secure, ii
 - MandrakeCampus, i
 - MandrakeExpert, i
 - MandrakeSoft, i
 - MandrakeSoft S.A., i
 - MandrakeStore, ii
- modules, 55
- NFS, 78
- owner, 11
 - change, 11
- packaging, ii
- password, 1
- permissions, 11
- Peter Pingus, iv
- PID, 4
- pipe
 - anonymous, 48
 - file, 47
 - named, 48
- pipes, 14

- printing
 - administration, 61
 - user, 66
- process, 4, 15, 53
- programming, ii
- promiscuous mode, 79
- prompt, 2, 5
- Queen Pingusa, iv
- redirection, 13
- root
 - directory, 39, 54
 - user, 2
- runlevel, 57
- security
 - level, 73
- shell, 5, 9
 - globbing patterns, 12
- standard
 - error, 13
 - input, 13, 25
 - output, 13, 25
- SUID, 78
- supermount, 44
- timestamps
 - atime, 9
 - ctime, 9
 - mtime, 9
- Torvalds, Linus, i
- UID, 2
- umask, 75
- UNIX, 1
- user, 1
- users
 - generic, iv
- utilities
 - file-handling, 9
- values
 - discrete, 12
- virus, 4