Reference Manual

Mandriva Linux 2006

http://www.mandriva.com

Reference Manual: Mandriva Linux 2006
Published September 2005

by NeoDoc (http://www.neodoc.biz)Camille Bégnis, Christian Roy, Fabian Mandelbaum, Roberto Rosselli del Turco, Marco De Vitis, Alice Lafox, John Rye, Wolfgang Bornath, Funda Wang, Patricia Pichardo Bégnis, Debora Rejnharc Mandelbaum, Mickael Scherer, Jean-Michel Dault, Lunas Moon, Céline Harrand, Fred Lepied, Pascal Rigaux, Thierry Vignaud, Giuseppe Ghibò, Stew Benedict, Francine Suzon, Indrek Madedog Triipus, Nicolas Berdugo, Thorsten Kamp, Fabrice Facorat, Xiao Ming, Snature , Guylhem Aznar, Pavel Maryanov, Annie Tétrault, Aurelio Marinho Jargas, Felipe Arruda, Marcia Gawlak Hoshi, Bob Rye, Jean-Luc Borie, and Roberto Patriarca

Legal Notice

About the Making of this Manual

This manual is written and maintained by NeoDoc (http://www.neodoc.biz). Translations are ensured by NeoDoc, Mandriva and other translators.
This document was written in DocBook XML. The set of files involved were managed using the Borges Collaborative Content Creation System (C3S) (http://sourceforge.net/projects/borges-dms). The XML source files were processed by xsltproc, and jadetex (for the electronic version) using a customized version of Norman Walsh's stylesheets. Screen shots were taken using xwd or GIMP and converted with convert (from the ImageMagick package). All these programs are free software and all of them are available in your Mandriva Linux distribution.

Table of Contents

List of Tables

Preface

1. About Mandriva Linux

Mandriva Linux is a GNU/Linux distribution supported by Mandriva S.A. which was born on the Internet in 1998. Its main goal was and still is to provide an easy-to-use and friendly GNU/Linux system. Mandriva's two pillars are open source and collaborative work.

On April 7th 2005 the Mandrakesoft company changed its name to Mandriva to reflect its merger with Brazil-based Conectiva. Its core product, Mandrakelinux, became Mandriva Linux.

1.1. Contacting the Mandriva Linux Community

The following are various Internet links pointing you to the most important Mandriva Linux-related sources. If you wish to know more about the Mandriva company, connect to our web site (`http://www.mandriva.com/`). You can also check out the Mandriva Linux distribution web site (`http://www.mandrivalinux.com/`) and all its derivatives.

Mandriva Expert (`http://www.mandrivaexpert.com/`) is Mandriva's support platform. It offers a new experience based on trust and the pleasure of rewarding others for their contributions.

We also invite you to subscribe to the various mailing lists (`http://www.mandriva.com/community/resources/newsgroups`) where the Mandriva Linux community demonstrates its vivacity and keenness.

Please also remember to connect to our security page (`http://www.mandriva.com/security`). It gathers all security-related material about Mandriva Linux distributions. You will find security and bug advisories, as well as kernel update procedures, the different security-oriented mailing lists which you can join, and Mandriva Online (`https://online.mandriva.com/`). This page is a must for any server administrator or user concerned about security.

1.2. Join the Club!

Mandriva offers a wide range of advantages through its Mandriva Club (`http://club.mandriva.com`):

- download commercial software normally only available in retail packs, such as special hardware drivers, commercial applications, freeware, and demo versions;
- vote for and propose new software through a volunteer-run RPM voting system;
- access more than 50,000 RPM packages for all Mandriva Linux distributions;
- obtain discounts for products and services on Mandriva Store (`http://store.mandriva.com`);
- access a better mirror list, exclusive to Club members;
- read multilingual forums and articles.
- access Mandriva's Knowledge Base (`http://club.mandriva.com/xwiki/bin/view/KB/`), a wiki-based site which holds documentation on many subjects such as administration, connectivity, troubleshooting, and more;
- chat with the Mandriva Linux developers on the Club Chat (`https://www.mandrivaclub.com/user.php?op=clubchat`);
- enhance your GNU/Linux knowledge through Mandriva's e-training lessons (`http://etraining.mandriva.com/`).

By financing Mandriva through the Mandriva Club you will directly enhance the Mandriva Linux distribution and help us provide the best possible GNU/Linux desktop to our users.

1.3. Subscribing to Mandriva Online

Mandriva offers a very convenient way to keep your system automatically up-to-date, keeping away bugs and fixing security holes. Visit the Mandriva Online Web site (`https://online.mandriva.com//`) to learn more about this service.

1.4. Purchasing Mandriva Products

Mandriva Linux users may purchase products on-line through the Mandriva Store (`http://store.mandriva.com/`). You will not only find Mandriva Linux software, operating systems and "live" boot CDs (such as Move), but also special subscription offers, support, third-party software and licenses, documentation, GNU/Linux-related books, as well as other Mandriva goodies.

1.5. Contributing to Mandriva Linux

The skills of the many talented folks who use Mandriva Linux can be very useful in the making of the Mandriva Linux system:

- **Packaging.** A GNU/Linux system is mainly made of programs picked up on the Internet. They have to be packaged in order to work together.
- **Programming.** There are many, many projects directly supported by Mandriva: find the one which most appeals to you and offer your help to the main developer(s).
- **Internationalization.** You can help us translate web pages, programs and their respective documentation.

Consult the development projects (`http://qa.mandriva.com/`) page to learn more about how you can contribute to the evolution of Mandriva Linux.

2. About this Reference Guide

This *Reference Manual* is aimed at people who wish to better understand their Mandriva Linux system, and who want to take advantage of its huge capabilities. After reading this manual, we hope that you'll be at ease with the daily administration of a GNU/Linux box. Here's an overview of its two components, along with a brief description of each chapter it contains:

- In the first part (*Introduction to the Linux System*), we introduce you to the GNU/Linux system. We discuss its architecture, the main files systems available and some more peculiar aspects like the `/proc` file system.

  In the first chapter ("*Basic UNIX® System Concepts*", page 7) we introduce the UNIX® paradigm while speaking more specifically of the GNU/Linux world. We discuss the standard file-manipulation utilities as well as some useful features provided by the shell. Then comes a complementary chapter ("*Disks and Partitions*", page 15) in which we explain how hard disks are managed under GNU/Linux. We also deal with hard disk partitioning.

  We explore the organization of the file tree in "*File-Tree Organization*", page 19. 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.

  The next chapter deals with file systems ("*The Linux File System*", page 23). After presenting the available file systems, we discuss file types and some additional concepts and utilities such as inodes and pipes. The following chapter ("*The /proc File System*", page 31) introduces a special (and virtual) GNU/Linux file system called `/proc`.
- The second part (*Hands On*) deals with more practical topics. We speak about the relationship between file systems and mount points, how to use the command line in your daily tasks, how to edit configuration files with light and powerful editors, and more.

  We cover the topics of ***file systems*** and ***mount points*** ("*File Systems and Mount Points*", page 37) by defining both terms, as well as explaining them with real life examples.

Then we tackle the command-line interface ("*Introduction to the Command Line*", page 43). We discuss file-handling utilities such as the `mkdir` and `touch` commands, and how to move, delete and copy files and directories in the file system. We also speak about file attributes and how to handle them with commands such as `chown` and `chgrp`. We then tackle shell globbing patterns, redirections and pipes, command-line completions, as well as basic job(s) control.

The next chapter covers text editing ("*Text Editing: Emacs and VI*", page 51). As most UNIX® configuration files are text files, you will eventually want or need 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, written by Richard M. Stallman, and the good-old Vi, written in 1976 by Bill Joy.

You should then be able to perform basic maintenance on your system. The following two chapters present practical uses of the command line ("*Command-Line Utilities*", page 59), and process control ("*Process Control*", page 71) in general.

The next chapter ("*The Start-Up Files: init sysv*", page 75) presents the Mandriva Linux boot-up procedure, and how to use it efficiently. We speak about init (the process which allows your system to boot) and different run levels you may wish to use (especially for maintenance tasks). We also briefly explain how to use drakxservices to manage services.

In the following chapter ("*Secure Remote Access*", page 79) we explain how to securely access a remote system (through `ssh`) to perform maintenance tasks, to run programs on it, etc. We give you a quick overview of the connection scheme and we then describe a basic server/client `ssh` setup. The usage of `scp` is also discussed.

We close out this book with a chapter dedicated to package management through the command line ("*Package Management From The Command Line*", page 81). In it you will learn how to use the urpmi utility along with its counterpart, urpme. We also explain how to manage media sources.

3. Note from the Editor

In the open-source philosophy, contributors are always welcomed! Updating the Mandriva Linux documentation pool is quite a task. You could provide help in many different ways. In fact, the documentation team is constantly looking for talented volunteers to help us out accomplish the following tasks:

- writing or updating;
- translating;
- copy editing;
- XML/XSLT programming.

If you have a lot of time, you can write or update a whole chapter; if you speak a foreign language, you can help us translate our manuals; if you have ideas on how to improve the content, let us know; if you have programming skills and would like to help us enhance the Borges Collaborative Content Creation System (C3S) (`http://sourceforge.net/projects/borges-dms`), join in. And don't hesitate to contact us if you find any mistakes in the documentation so we can correct them!

For any information about the Mandriva Linux documentation project, please contact the documentation administrator (`mailto:documentation@mandriva.com`) or visit the Mandriva Linux Documentation Project Pages (`http://qa.mandriva.com/twiki/bin/view/Main/DocumentationTask/`).

Please note that since June 2004 the Mandriva Linux documentation and the development of Borges is handled by NeoDoc (`http://www.neodoc.biz`).

4. Conventions Used in this Book

4.1. Typing Conventions

In order to clearly differentiate special words from the text flow, we use different renderings. The following table shows examples of each special word or group of words with its actual rendering, as well as its significance.

Formatted Example	Meaning
inode	Used to emphasize a technical term explained in the *Glossary*, page 85.
`ls -lta`	Used for commands and their arguments. (see *Commands Synopsis*, page 4).
`a_file`	Used for file names. It might also be used for RPM package names.
ls(1)	Reference to a `man` page. To read the page, simply type `man 1 ls`, in a command line.
`$ ls *.pid`	Formatting used for text snapshots of what you may see on your screen including computer interactions, program listings, etc.
`localhost`	Literal data which does not generally fit in any of the previously defined categories. For example, a key word taken from a configuration file.
OpenOffice.org	Defines application names. Depending on context, the application and command name may be the same but formatted differently. For example, most commands are written in lowercase, while applications names usually begin with an uppercase character.
Files	Indicates menu entries or graphical interface labels. The underlined letter, if present, informs you of a keyboard shortcut, accessible by pressing the **Alt** key plus the letter in question.
Le petit chaperon rouge	Identifies foreign language words.
Warning!	Reserved for special warnings in order to emphasize the importance of words. Read out loud.

Highlights a note. Generally, it gives additional information about a specific area.

Represents a tip. It can be general advice on how to perform a particular action, or hints at nice features, such as shortcuts, which could 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.

4.2. General Conventions

4.2.1. Commands Synopsis

The example below shows the symbols you will see 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 will find them elsewhere such as in the `man` pages.

The "<" (lesser than) and ">" (greater than) symbols denote a **mandatory** argument not to be copied as is, which should 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`, not `<foo.txt>` or `<filename>`.

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

The ellipsis (“...”) means an arbitrary number of arguments may be included.

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

4.2.2. Special Notations

From time to time, you will be asked to press, for example, the keys **Ctrl**-**R**, which means you need to press and hold the **Ctrl** key and tap the **R** character right after as well. The same applies for the **Alt** and **Shift** keys.

We use capital letters to represent the letter keys; this doesn't mean that you have to type them capitalized. However, there might be programs where typing **R** is not the same than typing **r**. You will be informed when dealing with such programs.

Regarding menus, going to menu item File→Reload user config (**Ctrl-R**) means: click on the File text displayed on the menu (generally located in the upper-left of the window). Then in the pull-down menu, click on the Reload user config item. Furthermore you are informed that you can use the **Ctrl**-**R** key combination (as described above) to get the same result.

4.2.3. System-Generic Users

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

Queen Pingusa	queen	This is our default user, used through most examples in this book.
Peter Pingus	peter	This user can be created afterward by the system administrator and is sometimes used to vary the text.

Chapter 1. Basic UNIX® System Concepts

The name "UNIX®" may be familiar to you. You may even use a UNIX® system at work, in which case this chapter may be of less interest.

For those of you who have never used a UNIX® system, reading this chapter is absolutely necessary. Understanding the concepts which will be introduced here will answer a surprisingly large number of questions commonly asked by beginners in the **GNU/Linux** world. Similarly some of these concepts will likely help you solve many of the problems you may encounter in the future.

1.1. Users and Groups

Since they have a direct influence on all other concepts, this section will introduce the concepts of users and groups which are extremely important.

Linux is a true ***multiuser*** system, and in order to use your GNU/Linux machine, you must have an ***account*** on the machine. When you created a user during installation, you actually created an account. In case you don't remember, you were prompted for the following items:

- the user's "real name" (which could actually be whatever you want);
- a ***login*** name;
- and a ***password***.

The two most important parameters here are the login name (commonly abbreviated to login) and the password. You must have both of these in order to access the system.

When you create a user, a default group is also created. Later on, we will see that groups are useful when you want to share files with other 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, at a university, you could have one group per department, another group for teachers, and so on. The opposite is also true: a user may be a member of one or more groups. A math teacher, for example, could be a member of the teachers' group and also of his math students' group.

Now that we've covered the background information, let us look at how to actually log in.

If the graphical interface is automatically started on boot up, your start-up screen will look like figure 1-1.

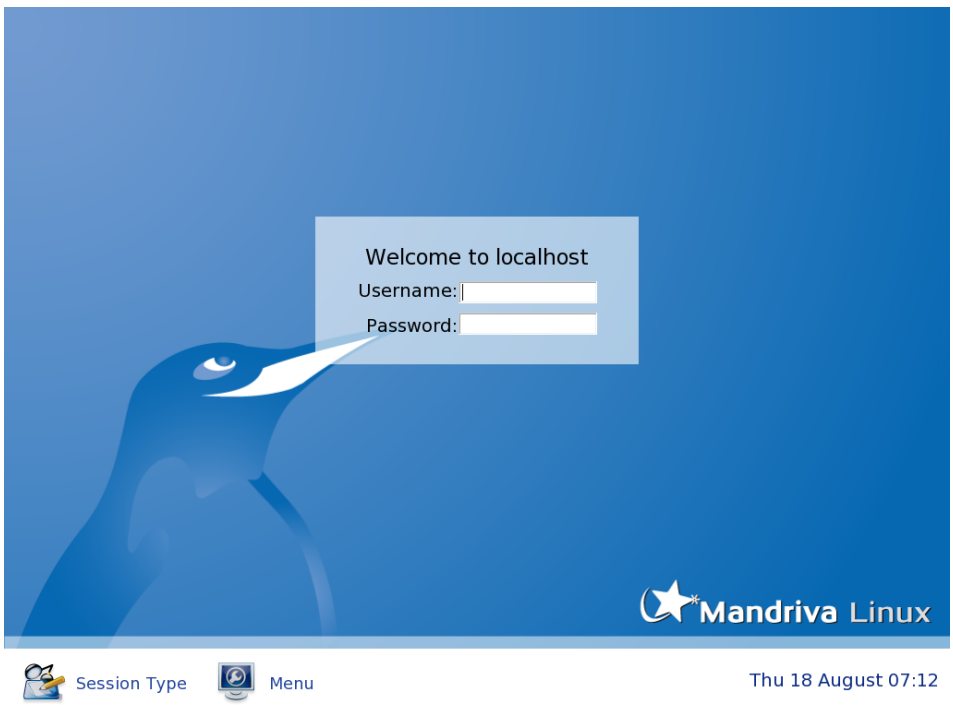


Figure 1-1. Graphical Mode Login Session

In order to log in, you must first select your account from the list. A new dialog will be displayed, prompting you for your password . Note that you will have to type in your password blindly, because the characters will be echoed on screen as stars (*) instead of the characters you type in the password field. You may also choose your session type (window manager). Once you're ready, press the Login button.

If you are in console or "text" mode, you will be presented with something similar to the following:

```
Mandriva Linux release 2006.0 (CodeName) for i586
Kernel 2.6.12-6mdk on an i686 / tty1
[machine_name] login:
```

To log in, type your login name at the login: prompt and press **Enter**. Next, the login program (called login) will display a Password: prompt and wait for you to enter your password. Like the graphic mode login, the console login will not echo the characters you are typing on the screen, but there will be no stars either.

Note that you can log in several times with the same account on additional ***consoles*** and under X. Each session you open is independent of the others, and it is even possible to open several X sessions at the same time (although this is not recommended since it consumes a lot of resources). By default, Mandriva Linux has six ***virtual consoles*** in addition to the one reserved for the graphical interface. You can switch to any of them by pressing the **Ctrl-Alt-F<n>** key sequence, where <n> is the number of the console that you want to switch to. By default, the graphical interface is on console number 7. Therefore, to switch to the second console, you would press the **Ctrl**, **Alt** and **F2** keys.

During the installation, DrakX also prompted you for the password of a very special user: root. This is the system administrator who will most likely be yourself. For your system's security, it is very important for the root account to be always protected by a good and hard-to-guess password!

If you regularly log in as ***root***, it can be very easy to make a mistake which could render your system unusable: one single mistake can do it. In particular, if you did not set a password for the root account, then **any** user can alter **any** part of your system (and even other operating systems on your machine!). Obviously this is not a good idea.

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

1.2. File Basics

Compared to Windows® and most other ***operating systems***, files are handled very differently under GNU/Linux. In this section we will cover the most obvious differences. For more information, please read "*The Linux File System*", page 23.

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. One thing we didn't mention about users is that every one of them possesses a personal directory (called the ***home directory***). The user is the owner of this directory and all files created in it. Also note that these have an associated group as well and it is the primary group that the user belongs to. As it was mentioned before (see *Users and Groups*, page 7), a user may be in more than one group at the same time.

However, this would not be very useful if that were the only notion of file ownership. As the file owner, a user may set **permissions** on files. These permissions distinguish between three user categories: 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 a member of the owner group.

There are three different permissions:

1. **Read** permission (r): enables a user to read the contents of a file. For a directory, the user can list its contents (i.e. the files in this directory).
2. **Write** permission (w): allows modification of a file's content. For a directory, the write permission allows a user to add or remove files from this directory, even if he is not the owner of these files.

3. **eXecute** permission (`x`): enables a file to be executed (normally only executable files have this permission set). For a directory, it allows a user to ***traverse*** it, which means going into or through that directory. Note that this is different to the read access: you may be able to traverse a directory but still be unable to read its content!

Every permission combination is possible. For example, you can allow only yourself to read the file and forbid access to all other users. As the file owner, you can also change the owner group (if and only if you're a member of the new group).

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

```
$ 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 with it. The first character is the file's type: if it's a regular file, you will see a dash (`-`). If it's a directory, the leftmost character will be a `d`. There are other file types, which we'll discuss later on. The next nine characters represent permissions associated with that file. The nine characters are actually three groups of three permissions. The first group represents the rights associated with the file owner; the next three apply to all users belonging to the owner group; and the last three apply to others. A dash (`-`) means that the permission is not set.
- Next comes the number of links for the file. Later on we'll see that the unique identifier of a file is not its name, but a number (the ***inode number***), and that it's possible for one file on disk to have several names. For a directory, the number of links has a special meaning, which will also be discussed a bit further.
- The next piece of information is the name of the file owner followed by the name of the owner group.
- Finally, the size of the file (in ***bytes***) and its last modification time are displayed, with the name of the file or directory itself as the last item on the line.

Lets take a closer look at the permissions associated with 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-----`. Here's a breakdown of the permissions.

- The first three characters (`rw-`) are the owner's rights, which in this case is queen. Therefore, queen has the right to read the file (`r`), to modify its content (`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. They will be able to read the file (`r`), but will not be able to 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. Those users don't have any rights on the file at all, for them the file will be "invisible".

For the `a_directory` directory, the rights are `rwxr-xr--`, so:

- peter, as the directory owner, can list files contained inside (`r`), add to or remove files from that directory (`w`), and may traverse it (`x`).
- Each user who isn't peter, but is a member of the `users` group, will be able to list files in this directory (`r`), but not remove or 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`). Because they don't have `wx` permissions, they won't be able to write files or 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, and could therefore grant ownership of the file to himself! `root` can read files on which he has no read permissions, traverse directories which he would normally have no access to, and so on. And if `root` lacks a permission, he only has to add it. `root` has complete control over the system, which involves a certain amount of trust in the person wielding the `root` password.

Lastly, it's 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 fewer limitations.

- A file name may contain any character, including non-printable ones, except for the ASCII character 0, which denotes the end of a string, and /, which is the directory separator. Moreover, because UNIX® is case sensitive, the files `readme` and `Readme` are different, because `r` and `R` are considered two **different** characters on UNIX®-based systems.
- As you may have noticed, a file name does not have to include an extension, unless that's the way you prefer to name your files. File extensions don't identify the content of files under GNU/Linux, nor almost any other operating system. So-called "file extensions" are quite convenient though. The period (`.`) under UNIX® is just one character among others, but it also has one special meaning. Under UNIX®, file names beginning with a period are "hidden files"[1], which also includes directories whose names start with a `.`

However it's worth noting that many graphical applications (file managers, office applications, etc.) actually use file extensions to recognize their files. It is therefore a good idea to use file-name extensions for those applications which support them.

1.3. Processes

A ***process*** defines an instance of a program being executed and its ***environment***. We will only mention the most important differences between GNU/Linux and Windows® here (please refer to "*Process Control*", page 71 for more information).

The most important difference is directly related to the **user** concept: each process is executed with the rights of the user who launched it. Internally, the system identifies processes with a unique number, called *process ID*, or PID. From this PID, the system knows who (that is, which user) has launched the process and a number of other pieces of information, and the system only needs to verify the process' validity. Lets take our `a_file` example. peter will be able to open this file in ***read-only mode***, but not in ***read-write mode*** because the permissions associated with the file forbid it. Once again the exception to this rule is `root`.

Because of this, GNU/Linux is virtually immune to viruses. In order to operate, viruses must infect executable files. As a user, you don't have write access to vulnerable system files, so the risk is greatly reduced. Generally speaking, viruses are very rare in the UNIX® world. There are only a few known viruses for Linux, and they are harmless when executed by a normal user. Only one user can damage a system by activating these viruses: `root`.

Interestingly enough, anti-virus software does exist for GNU/Linux, but mostly for DOS/Windows® files! Why are there anti-virus programs running on GNU/Linux which focus on DOS/Windows®? More and more often, you will see GNU/Linux systems acting as file servers for Windows® machines with the help of the Samba software package (see the Sharing Files and Printers chapter of the *Server Administration Guide*).

Linux makes it easy to control processes. One way is with "signals", which allow you to suspend or kill a process by sending it the corresponding signal. However, you are limited to sending signals to your own processes. With the exception of `root`, Linux and UNIX®-based systems do not allow you to send signals to a process launched by any other user. In "*Process Control*", page 71, you will learn how to obtain the PID of a process and how to send it signals.

1.4. A Short 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 encountered previously. This power is available because you have access, not only to all X applications, but also to thousands of other utilities in console mode (as opposed to graphical mode) which don't have graphical equivalents, with their many options and possible combinations, which would be hard to access in the form of buttons or menus.

Admittedly most people require a little help to get started. If you're not already working in console mode and are using the graphical interface, the first thing to do is to launch a terminal emulator. Access the main menu and you will find emulators in the System+Terminals menu. Choose the one you want, for example Konsole

1. By default, hidden files won't be displayed in a file manager, unless you tell it to. In a terminal, you must type the `ls -a` command to see all hidden files besides normal files. Essentially, they hold configuration information. From your `home/` directory, take a look at `.mozilla` or `.openoffice` to see an example.

or RXvt. Depending on your user interface, there may also be an icon which clearly identifies it on the panel (figure 1-2).

Figure 1-2. The Terminal Icon on the KDE Panel

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

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

This assumes that your user name is queen and that your machine's name is `localhost` (which is the case if your machine is not part of an existing network). Following the prompt there is space for you to type your commands. Note that when you're `root`, the prompt's `$` character becomes a `#` (this is true only in the default configuration, since you may customize all such details in GNU/Linux). In order to become `root`, type `su` after launching a shell.

```
[queen@localhost queen]$ su
# Enter the root password; (it will not be echoed to the screen)
Password:
# exit (or Ctrl-D) will take you back to your normal user account
[root@localhost queen]# exit
[queen@localhost queen]$
```

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

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

Next we will look at a few basic commands which are very useful.

1.4.1. cd: Change Directory

The `cd` command is just like the DOS one, with extras. It does just what its acronym states, changes the working directory. You can use `.` and `..`, which respectively stand 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 peter's home directory by typing `cd ~peter` (`~` on its own means your own `home/` directory). Note that as a normal user, you cannot usually get into another user's `home/` directory (unless they explicitly authorized it or if this is the default configuration on the system), unless you are `root`, so lets become `root` and practice:

```
$ su -
Password:
# 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 by typing `exit` and pressing the **Enter** key (or simply by pressing **Ctrl-D**).

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 the path to your home directory.
2. `PATH`: it contains the list of all directories in which the shell should look for executable files 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 contains your user ID.
5. `PS1`: determines what your prompt will display, and is often a combination of special sequences. You may read the bash(1) ***manual page*** for more information by entering the `man bash` command in a terminal.

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

```
$ 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 $HOME` example would not have worked. In fact, the shell first replaced `$HOME` with its value (`/home/queen`) so the line became `cd /home/queen`, which is what we wanted. The same thing happened with the `echo $USERNAME` example.

If one of your environment variables doesn't exist, you can create them temporarily by typing `export ENV_VAR_NAME=value`. Once this is done, you can verify it has been created:

```
$ export USERNAME=queen $ echo $USERNAME queen
```

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

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

```
$ cat /etc/fstab
# This file is edited by fstab-sync - see 'man fstab-sync' for details
/dev/hda2 / ext3 defaults 1 1
/dev/hdc /mnt/cdrom auto umask=0022,user,iocharset=utf8,noauto,ro,exec,users 0 0
none /mnt/floppy supermount dev=/dev/fd0,fs=ext2:vfat,--,umask=0022,iocharset=utf8,sync 0 0
none /proc proc defaults 0 0
/dev/hda3 swap swap defaults 0 0
$ cd /etc
$ cat modprobe.preload
# /etc/modprobe.preload: kernel modules to load at boot time.
#
# This file should contain the names of kernel modules that are
# to be loaded at boot time, one per line.  Comments begin with
# a '#', and everything on the line after them are ignored.
# this file is for module-init-tools (kernel 2.5 and above) ONLY
# for old kernel use /etc/modules
```

```
nvidia-agp
$ cat shells
/bin/bash
/bin/csh
/bin/sh
/bin/tcsh
```

1.4.4. less: a Pager

The name is a play on words related to the first pager ever used under UNIX® called `more`. A ***pager*** is a program which allows a user to view long files page by page (more accurately, screen by screen). The reason that we discuss `less` rather than `more` is that `less` is more intuitive. You should use `less` to view large files which will not fit on a single screen. For example:

```
less /etc/termcap
```

To browse through this file, use the up and down arrow keys. Press **Q** to quit. `less` can do far more than just that: press **H** for help to display the various options available.

1.4.5. ls: Listing Files

The `ls` (*LiSt*) command is equivalent to the `dir` command in DOS, but it can do much much more. In fact, this is largely because 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 numerous, so we will only describe 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 for "." and ".."
- `-R`: lists recursively, i.e. all files and sub-directories of directories entered on the command line.
- `-h`: if the file size is shown, it is in a human readable format, next to each file. It means that you'll see file sizes using suffixes like "K", "M" and "G", for example "234K" and "132M". Please also note that sizes are referred in a power of 2, not a power of 10. It means that 1K is really 1024 bytes instead of 1000 bytes.
- `-l`: prints additional information about the files such as the permissions associated to it, the owner and owner group, the file's size and the last-modification time.
- `-i`: prints the inode number (the file's unique number in the file system, see "*The Linux File System*", page 23) next to each file.
- `-d`: treats directories on the command line as if they were normal files rather than listing their contents.

Here are some examples:

- `ls -R`: recursively lists the contents of the current directory.
- `ls -ih images/ ..`: lists the inode number and the size, in human-readable format, for each file in the `images/` directory as well as in the parent directory of the current one.
- `ls -l images/*.png`: lists all files in the `images/` directory whose names end with `.png`, including the file `.png`, if it exists.

1.4.6. Useful Keyboard Shortcuts

There are a number of shortcuts available, their primary advantage being that they may save you a lot of typing time. This section assumes you're using the default shell provided with Mandriva Linux, bash, but these keystrokes might 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. In addition, the history is persistent from one session to another, so you won't lose all the commands you typed in previous sessions.

The left and right arrow keys move the cursor left and right on the current line, allowing you to edit your commands. But there's more to editing than just moving one character at a time: **Ctrl-A** and **Ctrl-E**, for example, will take you to the beginning and the end of the current line. The **Backspace** and **Del** keys work as expected. **Ctrl-K** will delete from the position of the cursor to the end of line, and **Ctrl-W** will delete the word before the cursor (so will **Alt-Backspace**).

Typing **Ctrl-D** on a blank line will let 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 put you back to the prompt. **Ctrl-L** clears the screen. **Ctrl-Z** temporarily stops a task, it suspends it. This shortcut is very useful when you forget to type the "&" character after typing a command. For instance:

```
$ xpdf MyDocument.pdf
```

Hence you cannot use your shell anymore since the foreground task is allocated to the xpdf process. To put that task in the background and restore your shell, simply type **Ctrl-Z** and then enter the `bg` command.

Finally, there are **Ctrl-S** and **Ctrl-Q**, which are used to suspend and restore output to the screen. They are not used often, but you might type **Ctrl-S** by mistake (after all, **S** and **D** are close to each other on the keyboard). So, if you get into the situation where you're typing but you can't see any characters appearing on the Terminal, try **Ctrl-Q**. Note that all the characters you typed between the unwanted **Ctrl-S** and **Ctrl-Q** will be printed to the screen all at once.

Chapter 2. Disks and Partitions

This chapter contains information for those who simply wish to know more about the technical details underlying their system. It will give a complete description of the PC partitioning scheme. Therefore it will be most useful if you are planning to manually configure your hard drive partitions.

2.1. Structure of a Hard Disk

A disk is physically divided into sectors. A sequence of sectors can form a partition. Roughly speaking, you can create as many partitions as you wish, up to 67 (3 primary partitions and a secondary partition containing up to 64 logical partitions inside): each partition is regarded as a single hard drive.

2.1.1. Sectors

To simplify, a hard disk is merely a sequence of sectors, which are the smallest data unit on a hard disk. The typical size of a sector is 512 bytes. The sectors on a hard disk of "n" sectors are numbered from "0" to "n-1".

2.1.2. Partitions

The use of multiple partitions enables you to create many virtual hard drives on your real physical drive. This has many advantages:

- Different operating systems use different disk structures (called ***file systems***): this is the case with Windows® and GNU/Linux. Having multiple partitions on a hard drive also allows you to install various operating systems on the same physical drive.
- For performance reasons, an operating system may prefer different drives with various file systems on them because they may be used for completely different things. One example is GNU/Linux which requires a second partition called Swap. The latter is used by the virtual memory manager as virtual memory.
- Even if all of your partitions use the same file system, it may prove useful to separate the different parts of your OS into different partitions. A simple configuration example would be to split your files into two partitions: one for your personal data, and another one for your programs. This allows you to update your OS, completely erasing the partition containing the programs while keeping the data partition safe.
- Because physical errors on a hard disk are generally located on adjacent sectors, not scattered across the disk, distributing your files across different partitions could limit data loss if your hard disk is physically damaged.

Normally, the partition type specifies the file system which the partition is supposed to contain. Each operating system might recognize some partition types, but not others. Please see "*File Systems and Mount Points*", page 37, and "*The Linux File System*", page 23, for more information.

2.1.3. Defining the Structure of Your Disk

2.1.3.1. The Simplest Way

This scenario would imply only two partitions: one for the Swap space, the other one for the files[1], called `root` and labelled as `/`.

1. The default file system used by Mandriva Linux is called `ext3`

A rule of thumb is to set the swap partition size to twice the size of your RAM memory (i.e.: if you have 128 MB of RAM memory the swap size should be of 256 MB). However for large memory configurations (512 MB or more), this rule isn't critical, and smaller sizes are acceptable. Please bear in mind that the swap partition's size may be limited depending on which platform you are using. For example it is limited to 2GB in x86, PowerPC and MC680x0; to 512MB on MIPS; to 128GB on Alpha and to 3TB on Ultrasparc. Bear in mind also that the larger the swap partition, the greater the amount of OS resources (notably RAM memory) needed to manage it.

2.1.3.2. Another Common Scheme

Separate data from programs. To be even more efficient, one usually defines more partitions to separate the system and programs from the data. The system partition will contain the programs required to start your system and to perform basic maintenance.

Therefore we could define four partitions:

Swap

A Swap partition whose size is roughly twice the size of physical RAM.

Root: `/`

The most important partition. Not only does it contain critical data and programs for the system, it also acts as a mount point for other partitions (see "*File Systems and Mount Points*", page 37).

The needs of the root partition in terms of size are not great, 400MB is generally enough. However, if you plan to install commercial applications, which are often located in the `/opt` directory, you will need to increase the size of the root partition accordingly. Alternatively, you could create a separate partition for `/opt`.

Static data: `/usr`

Most packages install the majority of their executables and data files under the `/usr` directory. The advantage of creating a separate partition is that it allows you to easily share it with other machines over a network.

The recommended size depends on the packages you wish to install, and can vary from 100MB for a very lightweight installation, to several GB for a full installation. A compromise of two or three GB (depending on your disk size) is usually sufficient.

Home directories: `/home`

This directory contains the personal directories for all of the users hosted on your machine. The partition size depends on the number of users hosted and their needs.

Another solution is to **not** create a separate partition for the `/usr` files: `/usr` could simply be a directory inside the root (`/`) partition, however you would need to increase the size of your root (`/`) partition accordingly.

Finally, you could also only create the Swap and `root` (`/`) partitions, in case you're not sure what you want to do with your computer. In this case, your `/home` directory would be located on the `root` partition, as would the `/usr` and the other directories.

2.1.3.3. Exotic Configurations

When setting up your machine for specific uses — such as a web server or a firewall — the needs are radically different to that of a standard desktop machine. For example, a FTP server will probably need a large separate partition for `/var/ftp`, while the `/usr` directory could be relatively small. In these situations, you're encouraged to carefully think about your needs before even beginning the installation process.

It is possible to resize most partitions; if you do need to resize them or to use a different partition scheme, without the need to reinstall your system and without losing your data. Please consult *Managing Your Partitions* of the *Starter Guide* .

With some practice, you'll even be able to move a crowded partition to a brand new hard drive.

2.2. Conventions for Naming Disks and Partitions

GNU/Linux uses a logical method to name partitions. First, when numbering the partitions, it ignores the file-system type of any partition you may have. Second, it names the partitions according to the disk on which they are located. This is how the disks are named:

- The primary master and primary slave IDE devices (whether they be hard disks, CD-ROM drives or anything else) are called `/dev/hda` and `/dev/hdb` respectively.
- On the secondary interface, the master is called `/dev/hdc` and the slave is `/dev/hdd`.
- If your computer contains other IDE interfaces (for example, the IDE interface present on some Soundblaster cards), the disks will be called `/dev/hde`, `/dev/hdf`, etc. You may also have additional IDE interfaces if you have RAID controllers.
- SCSI disks are called `/dev/sda`, `/dev/sdb`, etc., in the order of their appearance on the SCSI chain (depending on the increasing IDs). The SCSI CD-ROM drives are called `/dev/scd0`, `/dev/scd1`, always in the order of their appearance on the SCSI chain.

If you have SATA IDE disks, the SCSI naming scheme applies.

The partitions are named after the disk on which they're found, in the following way (in our example, we've used partitions on a primary master IDE disk but the same applies to all other types of disks):

- The primary (or extended) partitions are named `/dev/hda1` through `/dev/hda4`, when present.
- Logical partitions, if any, are named `/dev/hda5`, `/dev/hda6`, etc. in the order of their appearance in the table of logical partitions.

So GNU/Linux will name the partitions as follows:

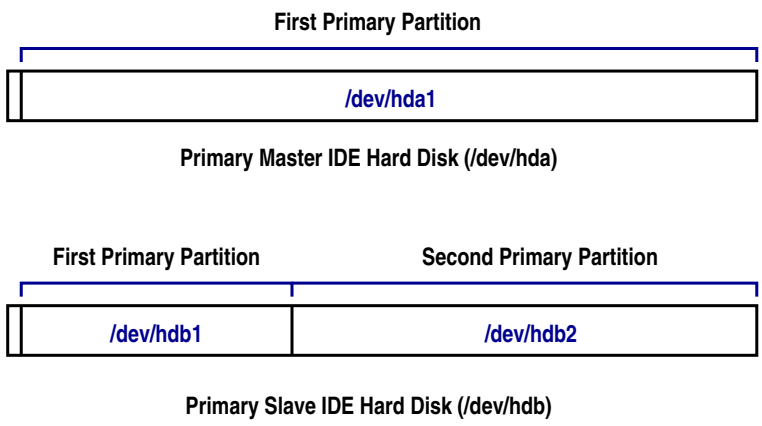


Figure 2-1. First Example of Partition Naming under GNU/Linux

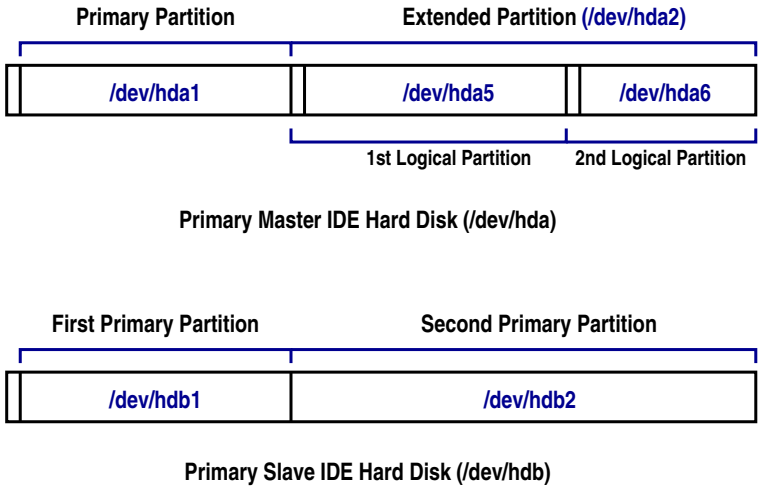


Figure 2-2. Second Example of Partition Naming under GNU/Linux

With this knowledge in hand, you should be able to name your various partitions and hard disks when you need to manipulate them. You'll also see that GNU/Linux names the partitions even if it doesn't know how to manage them initially (it ignores the fact that they're not native GNU/Linux partitions).

Mandriva Linux now uses udev (refer to the udev FAQ (`http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ`) for more information). It ensures full compatibility with the scheme described above and with standards like the Linux Standards Base Project (`http://www.linuxbase.org/`). Each device is dynamically added to the system as soon as it becomes available or needed.

Chapter 3. File-Tree Organization

Nowadays, a UNIX® system is big, very big. This is especially true of GNU/Linux: the amount of software available would make for an unmanageable system if no guidelines for the location of files in the tree existed.

The acknowledged standard is the FHS (Filesystem Hierarchy Standard) for which version 2.3 was released in January 2004. 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 will only provide a brief summary, but it should be enough to show you which directory is likely to contain a given file, or where a given file should be placed.

3.1. Shareable/unshareable, Static/Variable Data

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

These classifications are only a recommendation. It's not mandatory to follow them, but adopting these guidelines will greatly help you manage your system. Also, bear in mind that the static/variable distinction only applies to general system usage, not its configuration. If you install a program, you will obviously have to modify "normally" static directories, such as `/usr`.

3.2. The root Directory: /

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

- `/bin`: essential binary files. It contains the basic commands which will be used by all users and which are necessary for 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, yaboot for PPC, etc). It may or may not contain the kernel, but if the kernel isn't located in this directory then it must be in the root directory. Static, unshareable.
- `/dev`: system device files (`dev` for *DEVices*). Some files contained by `/dev` are mandatory, such as `/dev/null`, `/dev/zero`, and `/dev/tty`. Static, unshareable.
- `/etc`: contains all configuration files specific to the computer. This directory cannot contain binary files. Static, unshareable.
- `/home`: where all the personal directories of the system's users are located. This directory may or may not be shared (some large networks make it shareable via NFS). Your favorite application's (like e-mail readers or browsers) configuration files are located in this directory and start with a period ("."). For instance, the Mozilla configuration files lie in the `.mozilla` directory. Variable, shareable.
- `/lib`: it contains libraries which are essential to the system; it also stores kernel modules in the `/lib/modules/KERNEL_VERSION` sub-directory. It contains all libraries required by the binaries in the `/bin` and `/sbin` directories. The optional `ld*` execution time linker/loader as well as the dynamically-linked C library `libc.so` must also reside in this directory. Static, unshareable.
- `/mnt`: directory containing the mount points for temporarily-mounted file systems such as `/mnt/cdrom`, `/mnt/floppy`, etc. The `/mnt` directory is also used to mount temporary directories (a USB card will be mounted in `/mnt/removable`, for instance). Variable, unshareable.
- `/opt`: contains packages not essential for system operation. It is reserved for add-on packages; packages such as Adobe Acrobat Reader are often installed into /opt. The FHS recommends that static files (binaries, libraries, manual pages, etc.) installed in the `/opt` structure be placed in `/opt/package_name` and the specific configuration files in `/etc/opt`.

- /root: home directory for root. Variable, unshareable.
- /sbin: contains system binaries essential for system start-up. Most of these files can only be executed by root. A normal user may run them, but they might not do anything. Static, unshareable.
- /tmp: directory intended to contain temporary files which certain programs may create. Variable, unshareable.
- /usr: explained in more detail in */usr: The Big One, page 20*. Static, shareable.
- /var: location for data which may be modified in real time by programs (such as mail servers, audit programs, print servers, etc.). Variable. Its various sub-directories may be shareable or unshareable.

3.3. /usr: The Big One

The /usr directory is the main application-storage directory. The binary files in this directory are not required for system start-up or maintenance, so the /usr hierarchy may be, and often is, located on a separate file system. Because of its (usually) large size, /usr has its own hierarchy of sub-directories. We will mention just a few:

- /usr/X11R6: the entire X Window System hierarchy. All binaries and libraries required for the operation of X (including the X servers) 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: contains the majority of the system's binaries. **Any** binary program which isn't necessary for the maintenance of the system and isn't a system administration program must be located in this directory. The only exceptions are programs you compile and install yourself, which must be located in /usr/local.
- /usr/lib: 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 /usr/X11R6/lib/X11, the directory which contains the X Window System libraries (but only if X is installed)[1].
- /usr/local: this is where you must install any applications you compile from source. The installation program should create the necessary hierarchy.
- /usr/share: this directory contains all read-only, 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.

3.4. /var: Data Modifiable During Use

The /var directory contains all operative 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 sub-directories, and some are very useful:

- /var/log: contains the system's log files which you may read to troubleshoot your system (/var/log/messages and /var/log/kernel/errors to only name those two).
- /var/run: used to keep track of all processes utilized by the system since it was booted, enabling you to act on them in the event of a system ***runlevel*** change (see *"The Start-Up Files: init sysv", page 75*).
- /var/spool: contains the system's working files waiting for some kind of action or processing. For example, /var/spool/cups contains the print server's working files, while /var/spool/mail contains the mail server's working files (for example, all mail arriving and leaving your system).

1. Please note that Mandriva Linux now uses Xorg instead of X Window System as the default X Window system.

3.5. /etc: Configuration Files

/etc is one of UNIX® systems' most essential directories because it contains all the host-specific 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 and must be on the root partition at boot time.

Here are some important files:

- passwd and shadow: these are text files which contain all system users and their encrypted passwords. You will only see shadow if shadow passwords are used, which happens to be the default installation option for security reasons.
- inittab: this is the configuration file for init which plays a fundamental role in starting up the system.
- services: this file contains a list of existing network services.
- profile: this is the shell system-wide configuration file. Its settings can be overridden by shell-specific configuration files. For example, .bashrc for the bash shell.
- crontab: the configuration file for cron, the program responsible for periodic execution of commands.

Certain sub-directories exist for programs which require a large number of configuration files. This applies to the X Window System, for example, which stores all of its configuration files in the /etc/X11 directory.

Chapter 4. The Linux File System

Your GNU/Linux system is contained on your hard disk within a file system. In this chapter we will discuss the various aspects of file systems available on GNU/Linux, as well as the possibilities they offer.

4.1. Comparing a Few File Systems

During installation, you can choose different file systems for your partitions, so they will be formatted using different algorithms.

Unless you are a specialist, choosing a file system is not obvious. We will take a quick look at a few current file systems, all of which are available with Mandriva Linux.

4.1.1. Different Usable File Systems

4.1.1.1. Ext2

The Second Extended File System (its abbreviated form is ext2FS or simply ext2) has been GNU/Linux's default file system for many years. It replaced the Extended File System (that's where the "Second" comes from). ext2 corrects certain problems and limitations of its predecessor.

ext2 respects the usual standards for UNIX®-type file systems. Since its inception, it was destined to evolve while still offering great robustness and good performance.

Caution: It needs to be unmounted to be resized.

4.1.1.2. Ext3

As its name suggests, the Third Extended File System is ext2's successor. It is compatible with the latter but enhanced by incorporating ***journaling***.

One of the major flaws of "traditional" file systems like ext2 is their low tolerance to abrupt system breakdowns (power failure or crashing software). Generally speaking, once the system is restarted, these types of events involve a very long examination of the file system's structure and attempts to correct errors, which sometimes results in an extended corruption. This corruption could cause partial or total loss of saved data.

Journaling answers this problem. To simplify, let's say that what we are doing is storing the actions (such as the saving of a file) **before** really performing them. We could compare this functionality to that of a boat captain who uses a log book to note daily events. The result: an always coherent file system. And if problems occur, the verification is very rapid and the eventual repairs, very limited. Therefore the time spent in verifying a file system is proportional to its actual use and not related to its size.

So ext3 offers journaling file system technology while keeping ext2's structure, ensuring excellent compatibility. This makes it very easy to switch from ext2 to ext3 and back again.

As with ext2, it needs to be unmounted to be resized.

4.1.1.3. ReiserFS

Unlike ext3, `reiserfs` was written from scratch. It is a journalized file system like ext3, but its internal structure is radically different because it uses binary-tree concepts inspired by database software and also has a variable block size, making it optimal for use with several (thousands or hundreds of thousands of) small files. It also performs well with big files, making it suitable for various uses.

It can be resized "on the fly", without unmounting the file system.

4.1.1.4. JFS

JFS is the journalized file system designed and used by IBM. Proprietary and closed at first, IBM decided to open it to access to the free software movement. Its internal structure is similar to that of `reiserfs`.

It can not be resized on GNU/Linux.

4.1.1.5. XFS

XFS is the journalized file system designed by SGI and also used with the Irix operating system. Proprietary and closed at first, SGI decided to open it to access by the free software movement. Its internal structure has lots of different features, such as support for real-time bandwidth, extents, and clustered file systems (but not in the free version).

With GNU/Linux it can be resized for a bigger size only. You can't reduce it. Resizing can only be made with a mounted filesystem.

4.1.2. Differences Between File Systems

	Ext2	**Ext3**	**ReiserFS**	**JFS**	**XFS**
Stability	Excellent	Very Good	Good	Medium	Good
Tools to restore erased files	Yes (complex)	Yes (complex)	No	No	No
Reboot time after crash	Long, even very long	Fast	Very fast	Very fast	Very fast
Status of the data in case of a crash	Generally speaking, good, but high risk of partial or total data loss	Very good	Medium[a]	Very good	Very good
ACL support	Yes	Yes	No	No	Yes
Notes: a. It is possible to improve results on crash recovery by journaling the **data** and not just the **metadata**, adding the option *`data=journal`* to `/etc/fstab`.					

Table 4-1. File System Characteristics

The maximum size of a file depends on many parameters (i.e. the block size for ext2/ext3), and is likely to evolve depending on the kernel version and architecture.

In kernel 2.6.X the block device limit can be extended using a kernel compiled with Large Block Device support enabled (`CONFIG_LBD=y`). For more information, consult Adding Support for Arbitrary File Sizes to the Single UNIX Specification (`http://www.unix.org/version2/whatsnew/lfs.html`), Large File Support in Linux (`http://www.suse.com/~aj/linux_lfs.html`), and Large Block Devices (`http://www.gelato.unsw.edu.au/IA64wiki/LargeBlockDevices`). With this and a file system which supports it you can reach up to many TB without special file-system tricks as is done by JFS for the file-system size.

4.1.3. And Performance Wise?

It is always very difficult to compare performance between file systems. All tests have their limitations and the results must be interpreted with caution, comparisons done a couple of months or weeks ago are already too old. Let's not forget that today's hardware (specially concerning hard drive capacities and hard disk controller performance) has greatly leveraged the differences between the different file systems.

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 ext2. For a server, a journalized file system such as ext3 is preferred. `reiserfs`, perhaps because of its genesis, is more suited to a database server. JFS is preferred in cases were file system throughput is the main issue. XFS is interesting if you need any of its advanced features. For "normal" use, these four file systems give approximately the same results and all of them have different options to tune the file system for a particular use. Please refer to the file system's documentation for more information.

4.2. Everything is a File

The *Starter Guide* introduced the file ownership and permissions access concepts, but really understanding the UNIX® ***file system*** (and this also applies to Linux's file systems) requires that we redefine the concept of "What is a file".

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. 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 **content** 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.

4.2.1. The Different File Types

When you issue `ls -l`, the character before the access rights identifies the file type. We have already seen two types of files: regular files (`-`) and directories (`d`). You can also find other types if you wander through the file tree and list the contents of directories:

1. ***Character mode files***: they are either special system files (such as `/dev/null`, which we have already discussed), or peripherals (serial or parallel ports), which share the trait 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 unlike character files, their contents **are** buffered. For example, some files in this category are: hard disks, partitions on a hard disk, floppy drives, CD-ROM drives and other storage devices. Files like `/dev/hda`, `/dev/sda5` are examples of block-mode files. Such files are identified by the letter `b`.
3. ***Symbolic links***: these files are very common and heavily used in the Mandriva Linux system start-up procedure (see "*The Start-Up Files: init sysv*", page 75). As their name implies, their purpose is to link files in a symbolic way, which means that they are files whose content is the path to a different file. They may not point to an existing file. They are very frequently called ***soft links***, and such files are identified by the letter `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 actually have names. However they are very rare and it's not likely that you will see one during your journey into the file tree. Such files are identified by the letter `p`. See *"Anonymous" Pipes and Named Pipes*, page 27.

5. ***Sockets***: this is the file type for all network connections, but only a few of them have names. 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=
$
```

4.2.2. Inodes

Inodes are, along with the "Everything Is a File" paradigm, a 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 file system, including directories, named pipes, character-mode files and so on. Which leads to this other famous sentence: "The inode is the file". Inodes are how UNIX® identifies a file in a unique way.

No, you didn't misread that: in UNIX®, you **do not identify a file by its name**, but by its inode number[1]. The reason for this is that the same file may have several names, or even no name. In UNIX®, a file name is just an entry in a directory inode. Such an entry is called a link. Let us look at links in more detail.

4.3. Links

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

```
$ pwd
/home/queen/example
$ ls
$ touch a
$ ls -il a
32555 -rw-r--r--  1 queen queen 0 Aug  6 19:26 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;
- 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's currently the only one: the link counter shows `1`.

But now, if we type:

```
$ ln a b
$ ls -il a b
32555 -rw-r--r--  2 queen queen 0 Aug  6 19:26 a
32555 -rw-r--r--  2 queen queen 0 Aug  6 19:26 b
```

1. **Important**: note that inode numbers are unique **per file system**, which means that an inode with the same number can exist on another file system. This leads to the difference between on-disk inodes and in-memory inodes. While two on-disk inodes may have the same number if they are on two different file systems, in-memory inodes have a unique number right across the system. One solution to obtain uniqueness, for example, is to hash the on-disk inode number against the block device identifier.

```
$
```

We create another link to the same inode. As you can see, we didn't create a file named `b`. 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 rather than 1.

Now, if we do:

```
$ rm a
$ ls -il b
32555 -rw-r--r--  1 queen queen 0 Aug  6 19:26 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 in UNIX® has no name; instead, it has one or more ***link***(s) in one or more directories.

Directories themselves are also stored in inodes. Their link count coincides with the number of sub-directories within them. This is due to the fact that there are at least two links per directory: the directory itself (represented by the entry `.`) and its parent directory (represented by `..`). So a directory with two sub-directories will have at least four links: `.`, `..` and links for each sub-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 Mandriva Linux web site (`http://www.mandrivalinux.com`) in your file tree, no matter which directory you look in. Similarly, when you use a ***pipe*** in the shell, the inode corresponding to the pipe exists, but it is not linked. Temporary files are another example of inodes without names. You create a temporary file, open it, and then remove it. The file exists while it's open, but nobody else can open it (as there is no name to open it). This way, if the application crashes, the temporary file is removed automatically.

4.4. "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, while 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've already seen examples of how to use pipes in the shell, but let's take another look 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. This 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 unlike the pipes used by shells, have names (i.e.: they are linked, whereas shell pipes are not)[2]. The command to create a named pipe is `mkfifo`:

```
$ mkfifo a_pipe
$ ls -il
total 0
169 prw-rw-r--  1 queen queen 0 Aug  6 19:37 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
```

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

```
169 prw-rw-r--  2 queen queen 0 Aug  6 19:37 a_pipe|
169 prw-rw-r--  2 queen queen 0 Aug  6 19:37 the_same_pipe|
$ ls -d /proc/[0-9] >a_pipe
  #
  # The process is blocked, as there is no reader at the other end.
  # Type Control Z to suspend the process...
  #
[1]+  Stopped                 ls -F --show-control-chars --color=auto -d /proc/[0-9] >a_pipe
  #
  # ...Then put in into the background:
  #
$ bg
[1]+ ls -F --show-control-chars --color=auto -d /proc/[0-9] >a_pipe &
  #
  # now read from the pipe...
  #
$ head -5 <the_same_pipe
  #
  # ...the writing process terminates
  #
/proc/1/
/proc/2/
/proc/3/
/proc/4/
/proc/5/
[1]+  Done                    ls -F --show-control-chars --color=auto -d /proc/[0-9] >a_pipe
$
```

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

```
$ head -5 <a_pipe
#
# Program blocks, suspend it: C-z
#
[1]+  Stopped                 head -5 <a_pipe
#
# Put it into the background...
#
$ bg
[1]+ 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]+  Done                    head -5 <a_pipe
$
```

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 were immediately returned to the prompt, but `head` executed later and you only saw its output after returning.

4.5. Special Files: Character Mode and Block Mode Files

As already stated, such files are either created by the system or peripherals on your machine. We also mentioned that the contents of block mode character files were buffered, while 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 should have observed the following: the first time the command was launched, the entire content of the floppy was read. The second time you executed the command, there was no access to the floppy drive at all. This is because the content of the floppy was buffered the first time you launched the command — and you did not change anything on the floppy between the two instances.

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 has a nice side effect: not only are reads buffered, but writes are buffered too. This allows for writes to the disks to be asynchronous: when you write a file on disk, the write operation itself is not immediate. It will only occur when the Linux kernel decides to execute the write to the hardware. Of course, if you need it can be overridden for a certain filesystem; take a look at the `sync` and `async` options at the mount(8) man page and also at *File Attributes*, page 30 for more details.

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/hdc /dev/lp0
brw-rw----  1 queen cdrom 22, 0 Feb 23 19:18 /dev/hdc
crw-rw----  1 root root 6, 0 Feb 23 19:17 /dev/lp0
```

Here, the major and minor of `/dev/hdc` are 22 and 0, whereas for `/dev/lp0`, they are 6 and 0. Note that these numbers are unique per file category, which means that there can be a character mode file with major 22 and minor 0, and similarly, there can be a block mode file with major 6 and minor 0. These numbers exist for a simple reason: it allows the kernel to associate the correct operations to these files (that is, to the peripherals these files refer to): you don't handle a floppy drive the same way as, say, a SCSI hard drive.

4.6. 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 — since 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 content 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 `rwxrwxrwx`. 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 file system to the said inode. The reason is simple: the link counter is stored in the inode itself, and inodes cannot be shared between file systems. Symlinks allow do this;
- You cannot link directories to avoid creating loops in the file system. 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.

4.7. File Attributes

The same way that FAT has file attributes (archive, system file, invisible, read-only), a GNU/Linux file system has its own, but they are different. We will briefly go over 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` and `chattr`. You probably guessed it, `lsattr` "LiSts" attributes, whereas `chattr` "CHanges" them. These attributes can only be set on directories and regular files. The following are some of the attributes possible, for a complete list please refer to chattr(1):

1. `A` ("no **A**ccess time"): if a file or directory has this attribute set, whenever it is accessed, either for reading or for writing, its last access time won't be updated. This can be useful, for example, on files or directories which are often accessed for reading, especially since this parameter is the only one which changes on an inode when it is open read-only.
2. `a` ("**a**ppend 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 **d**ump"): `dump` is the standard UNIX® utility for backups. It dumps any file system for which the dump counter is 1 in `/etc/fstab` (see chapter "*File Systems and Mount Points*", page 37). 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 sub-directories and files under it.
4. `i` ("**i**mmutable"): a file or directory with this attribute set can not be modified at all: it cannot be renamed, no further link can be created to it[3] and it cannot be removed. Only `root` can set or clear this attribute. Note that this also prevents changes to access time, therefore you don't need to set the `A` attribute when `i` is set.
5. `s` ("**s**ecure deletion"): when a file or directory with this attribute is deleted, the blocks it was occupying on disk are overwritten with zeroes.
6. `S` ("**S**ynchronous mode"): when a file or directory has this attribute set, all modifications on it are synchronous and written to the disk immediately.

For example, you may want to set the `i` attribute on essential system files in order to avoid bad surprises. Also, consider the `A` attribute on man pages: this prevents a lot of disk operations and, in particular, can save some battery life on laptops.

3. Be sure to understand what "adding a link" means, both for a file and a directory!

Chapter 5. The /proc File System

The /proc file system is specific to GNU/Linux. It is a virtual file system, so the files that you will find in this directory do not actually take up any space on your hard drive. It is a very convenient way to obtain information about the system, especially since 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 the results. There are a few programs which display information about processes (top, ps and friends) which do exactly that. /proc is also a good source of information about your hardware, and just like the programs which display processes, quite a few programs are just interfaces to the information contained in /proc.

There is also a special subdirectory, /proc/sys. It allows you to display kernel parameters and to change them, with the changes taking effect immediately.

5.1. Information About Processes

If you list the contents of the /proc directory, you will see many directories where the name of the directory is a number. These are the directories containing 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 those of other users. So, login as root and see what information is available from process 1, which is the init process and is the one responsible for starting up all other processes:

```
$ su
Password:
# cd /proc/1
# ls -l
total 0
-r--------  1 root root 0 Aug 15 18:14 auxv
-r--r--r--  1 root root 0 Aug 15 18:14 cmdline
lrwxrwxrwx  1 root root 0 Aug 15 18:14 cwd -> //
-r--------  1 root root 0 Aug 15 18:14 environ
lrwxrwxrwx  1 root root 0 Aug 15 18:14 exe -> /sbin/init*
dr-x------  2 root root 0 Aug 15 18:14 fd/
-rw-r--r--  1 root root 0 Aug 15 18:14 loginuid
-r--r--r--  1 root root 0 Aug 15 18:14 maps
-rw-------  1 root root 0 Aug 15 18:14 mem
-r--r--r--  1 root root 0 Aug 15 18:14 mounts
-rw-r--r--  1 root root 0 Aug 15 18:14 oom_adj
-r--r--r--  1 root root 0 Aug 15 18:14 oom_score
lrwxrwxrwx  1 root root 0 Aug 15 18:14 root -> //
-rw-------  1 root root 0 Aug 15 18:14 seccomp
-r--r--r--  1 root root 0 Aug 15 18:14 stat
-r--r--r--  1 root root 0 Aug 15 18:14 statm
-r--r--r--  1 root root 0 Aug 15 18:14 status
dr-xr-xr-x  3 root root 0 Aug 15 18:14 task/
-r--r--r--  1 root root 0 Aug 15 18:14 wchan
#
```

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

1. cmdline: this (pseudo-)file contains the entire command line used to invoke the process. It is not formatted: there are no spaces between the program and its arguments, and there is no newline at the end of the line. To view it, you could 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 `VARIABLE=value` form. Similar to `cmdline`, the output is not formatted at all: no newlines separate the different variables, and there is no newline at the end. One way to view it: `perl -ple '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 content 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. From left to right, the fields 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 or filename, this is an anonymous mapping. 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. Note that the `stat` and `statm` files are obsolete. The information they contained is now stored in `status`.

If we list the contents of the `fd` directory for a randomly chosen process we obtain this:

```
# ls -l /proc/8141/fd/
total 4
lrwx------  1 peter peter 64 Aug  4 09:05 0 -> /dev/tty1
lrwx------  1 peter peter 64 Aug  4 09:05 1 -> /dev/tty1
lrwx------  1 peter peter 64 Aug  4 09:05 2 -> /dev/tty1
l-wx------  1 peter peter 64 Aug  4 09:05 3 -> /home/peter/seti32/lock.sah
#
```

In fact, this is the list of file descriptors opened by the process. Each opened descriptor is shown by a symbolic link, where the name is the descriptor number, and which points to the file opened by this descriptor[1]. Note 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.

5.2. Information on the Hardware

Apart from the directories associated with 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 shows the following:

```
$ ls -d [a-z]*
acpi/      diskstats    iomem      locks       pci            sysvipc/
asound/    dma          ioports    mdstat      scsi/          tty/
buddyinfo  driver/      irq/       meminfo     self@          uptime
bus/       execdomains  kallsyms   misc        slabinfo       version
cmdline    fb           kcore      modules     splash         vmstat
config.gz  filesystems  keys       mounts@     stat
cpuinfo    fs/          key-users  mtrr        swaps
crypto     ide/         kmsg       net/        sys/
devices    interrupts   loadavg    partitions  sysrq-trigger
$
```

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 uses 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 track down a conflict, look at the contents of these three files:

```
$ cat interrupts
           CPU0
  0:     543488          XT-PIC  timer
  2:          0          XT-PIC  cascade
  5:        109          XT-PIC  ohci_hcd:usb2, eth1
  7:          1          XT-PIC  parport0
```

1. If you remember what was described in *Redirections and Pipes*, page 47, you know what descriptors `0`, `1` and `2` stand for. Descriptor `0` is the standard input, descriptor `1` is the standard output and descriptor `2` is the standard error.

```
  8:          0          XT-PIC  rtc
  9:       3432          XT-PIC  acpi, NVidia CK8
 10:      52855          XT-PIC  ehci_hcd:usb3, eth0
 11:       7538          XT-PIC  libata, ohci_hcd:usb1
 12:       1386          XT-PIC  i8042
 14:         20          XT-PIC  ide0
 15:       5908          XT-PIC  ide1
NMI:          0
LOC:          0
ERR:          0
MIS:          0

$ cat ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
0378-037a : parport0
037b-037f : parport0
03c0-03df : vesafb
03f6-03f6 : ide0
03f8-03ff : serial
0778-077a : parport0
0970-0977 : 0000:00:0b.0
0970-0977 : sata_nv
09f0-09f7 : 0000:00:0b.0
09f0-09f7 : sata_nv
0b70-0b73 : 0000:00:0b.0
0b70-0b73 : sata_nv
0bf0-0bf3 : 0000:00:0b.0
0bf0-0bf3 : sata_nv
0cf8-0cff : PCI conf1
4000-407f : motherboard
4000-4003 : PM1a_EVT_BLK
4004-4005 : PM1a_CNT_BLK
4008-400b : PM_TMR
4020-4027 : GPE0_BLK
4080-40ff : motherboard
4080-40ff : pnp 00:00
4200-427f : motherboard
4200-427f : pnp 00:00
4280-42ff : motherboard
4280-42ff : pnp 00:00
4400-447f : motherboard
4400-447f : pnp 00:00
4480-44ff : motherboard
44a0-44af : GPE1_BLK
5000-503f : motherboard
5000-503f : pnp 00:01
5100-513f : motherboard
5100-513f : pnp 00:01
9000-9fff : PCI Bus #02
9000-907f : 0000:02:07.0
9000-907f : 0000:02:07.0
ac00-ac0f : 0000:00:0b.0
ac00-ac0f : sata_nv
b000-b07f : 0000:00:0b.0
b000-b07f : sata_nv
b800-b8ff : 0000:00:06.0
b800-b8ff : NVidia CK8
bc00-bc7f : 0000:00:06.0
bc00-bc7f : NVidia CK8
c000-c007 : 0000:00:04.0
c000-c007 : forcedeth
c400-c41f : 0000:00:01.1
f000-f00f : 0000:00:09.0
```

```
f000-f007 : ide0
f008-f00f : ide1

$cat dma
 3: parport0
 4: cascade
$
```

Or, more simply, use the `lsdev` command, which gathers information from these files and sorts them by peripheral, which is undoubtedly more convenient.[2]:

```
$ lsdev
Device            DMA   IRQ  I/O Ports
------------------------------------------------
0000:00:01.1                 c400-c41f
0000:00:04.0                 c000-c007
0000:00:06.0                 b800-b8ff bc00-bc7f
0000:00:09.0                 f000-f00f
0000:00:0b.0                 0970-0977 09f0-09f7 0b70-0b73 0bf0-0bf3 ac00-ac0f b000-b07f
0000:02:07.0                 9000-907f     9000-907f
cascade             4     2
CK8                          9
dma                          0080-008f
dma1                         0000-001f
dma2                         00c0-00df
eth0                        10
eth1                         5
forcedeth                    c000-c007
fpu                          00f0-00ff
GPE0_BLK                     4020-4027
GPE1_BLK                     44a0-44af
i8042                       12
ide0                        14  01f0-01f7 03f6-03f6   f000-f007
ide1                        15  0170-0177 0376-0376   f008-f00f
keyboard                     0060-006f
motherboard                  4000-407f 4080-40ff 4200-427f 4280-42ff 4400-447f 4480-44ff 5000-503f 5100-513f
NVidia                       b800-b8ff   bc00-bc7f
ohci_hcd:usb1               11
parport0            3        7  0378-037a 037b-037f 0778-077a
PCI                          0cf8-0cff 9000-9fff
pic1                         0020-0021
pic2                         00a0-00a1
PM1a_CNT_BLK                 4004-4005
PM1a_EVT_BLK                 4000-4003
PM_TMR                       4008-400b
pnp                          4080-40ff   4200-427f   4280-42ff   4400-447f   5000-503f   5100-513f
rtc                          8  0070-0077
sata_nv                      0970-0977   09f0-09f7   0b70-0b73   0bf0-0bf3   ac00-ac0f   b000-b07f
serial                       03f8-03ff
timer                        0
timer0                       0040-0043
timer1                       0050-0053
vesafb                       03c0-03df
$
```

An exhaustive listing of files would take 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 information used by the `lsmod` command which displays it in a more readable form.
- `meminfo`: this file contains information on memory usage at the time you print its contents. The `free` command will display the same information in a easier-to-read format.
- `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 charge level of your battery, and if the APM BIOS of your laptop supports it (unfortunately this is not the case for all laptops), the remaining battery life in minutes, etc. The

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

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.

Note that modern computers now provide ACPI support instead of APM. See below.

- `bus`: this subdirectory contains information on all peripherals found on different buses in your machine. The information is usually not readable, and for the most part it is reformatted with external utilities: `lspcidrake`, `lspnp`, etc.
- `acpi`: several of the files and directories provided in this directory are interesting especially for laptops, where you can select several power-saving options. Note that it is easier to modify those options using a higher level application, such as the ones included in the `acpid` `kapacity` packages.

  The most interesting entries are:

  `battery`

  Shows how many batteries are in the laptop, and related information as current remaining life, maximum capacity, etc.

  `button`

  Allows you to control actions associated to "special" buttons such as power, sleep, lid, etc.

  `fan`

  Displays the state of the fans on your computer, whether they are running or not, and enables you to start/stop them according to certain criteria. The amount of control of the fans in your machine depends on the motherboard.

  `processor`

  There is one subdirectory for each of the CPUs in your machine. Control options vary from one processor to another. Mobile processors have more features enabled, including:

  - possibility to use several power states, balancing between performance and power consumption;
  - possibility to use clock rate change in order to reduce the amount of CPU power consumption;

  Note that there are several processors that do not offer these possibilities.

  `thermal_zone`

  Information about how hot your system/processor is running.

5.3. Display and change kernel parameters

The role of the `/proc/sys` subdirectory is to report different kernel parameters, and to allow you to interactively change some of them. As opposed to all other files in `/proc`, some files in this directory can be written to, but only by `root`.

A list of directories and files would take too long to describe, mostly because the content of the directories are system-dependent and that most files will only be useful for very specialized applications. However, here are two common uses of this subdirectory:

1. Allow routing: even if the default kernel from Mandriva 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 with a 0 if you want to forbid routing.

2. Prevent IP spoofing: IP spoofing consists of 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]. You can make the kernel prevent this kind of intrusion. Type:

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

3. But not ***hackers***!

and this kind of attack becomes impossible.

These changes will only remain in effect while the system is running. If the system is rebooted, then the values will go back to their defaults. To reset the values to something other than the default at boot time, you can take the commands that you typed at the shell prompt and add them to `/etc/rc.d/rc.local` so that you avoid typing them each time. Another solution is to modify `/etc/sysctl.conf`, refer to sysctl.conf(5) and to sysctl(8) for more information.

Chapter 6. File Systems and Mount Points

As we have seen in "*Disks and Partitions*", page 15, all the files of the system are organized in a single tree. And actually each time we want to access a removable device such as a CD-ROM or a remote location on a file server, its content will be literally "grafted" on some branch of the tree.

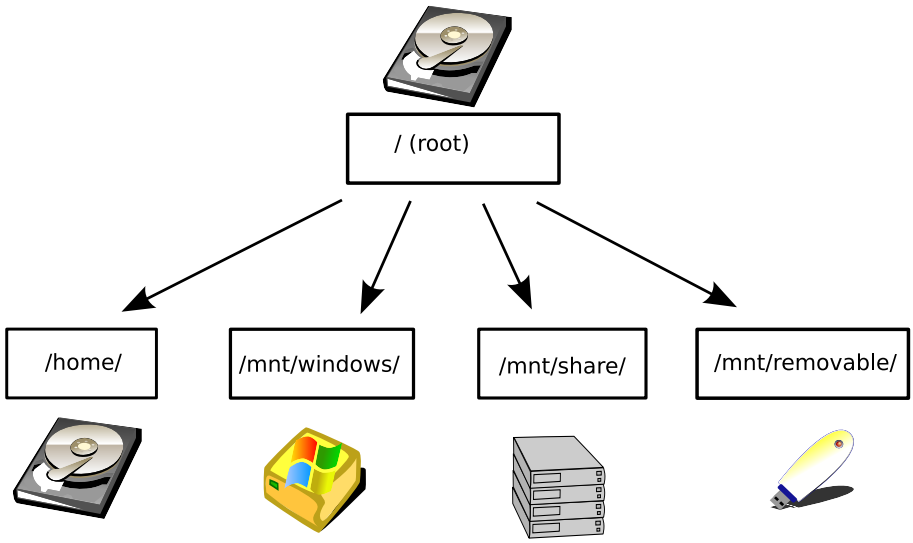


Figure 6-1. Mount Points Illustrated

figure 6-1 shows this: the root, made of a GNU/Linux partition contains another linux partition for `/home/` but also a Windows® one, a remote share from a file server (either Windows® or UNIX®), and a USB key. Nowadays many devices can be mounted on a GNU/Linux filesystem, including almost all existing filesystem types, WebDAV and even exotic things such as Google™ mail...

In order to better grasp the concepts around mount points, we base this chapter on a practical case. Suppose you have just purchased a brand new hard disk with no partitions on it. Your Mandriva Linux partition is completely full, and as you need more space, you decide to move a whole section of the tree structure[1] to your new hard disk. Because your new disk has a large capacity, you decide to move your biggest directory to it: `/usr`.

We will use this example along this chapter starting *Partitioning a Hard Disk, Formatting a Partition*, page 38, but first a bit of theory.

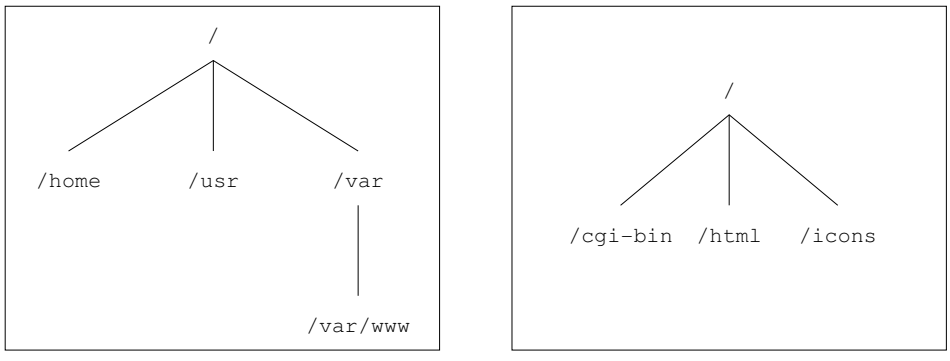
6.1. Principles

Every hard disk may be divided into partitions, and each one contains a file system. While Windows® assigns a letter to each of these file systems (well, actually only to those it recognizes), GNU/Linux has a unique tree structure of files, and each file system is ***mounted*** at one location in that tree structure.

Just as Windows® needs a `C:` drive, GNU/Linux must be able to mount the root of its file tree (`/`) on a partition which contains the ***root file system***. Once the root is mounted you can mount other file systems in the tree structure at various ***mount points*** within the tree. Any directory below the root structure can act as a mount point, and you can mount the same file system several times on different mount points.

This allows for great configuration flexibility. For example if you were to configure a web server, it's fairly common to dedicate an entire partition to the directory which contains the web server's data. The directory which usually contains the data is `/var/www` and acts as the mounting point for the partition. Also, a big `/home` partition should be considered if you plan on downloading a large amount of software, store many work or personal documents and photos, music files, etc.. You can see in figure 6-2 and figure 6-3 how the system looks before and after mounting the file system.

1. Our example assumes that the whole tree is contained in a single partition.



Root file system
(already mounted)

File system containing files
of directory "/var/www"
(not yet mounted)

Figure 6-2. A Not Yet Mounted File System

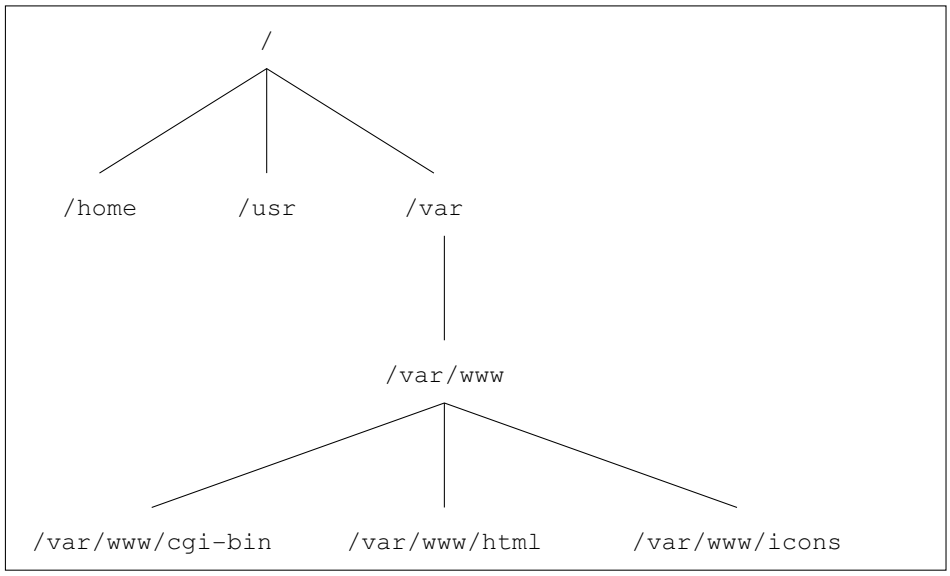


Figure 6-3. File System Is Now Mounted

As you can imagine, this offers a number of advantages: the tree structure will always be the same, whether it's on a single file system or extended over several dozen. This flexibility allows you to 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 sub-directories, these will simply be "hidden" by the newly mounted file system. The files will not be deleted, but they will not be accessible until you free the mount point.

It's actually possible to access the data "hidden" by the newly mounted file system. You simply need to mount the hidden directory with the `--bind` option. For example, if you just mounted a directory in `/hidden/directory/` and want to access its original content in `/new/directory`, you would have to run:

```
mount --bind /hidden/directory/ /new/directory
```

6.2. Partitioning a Hard Disk, Formatting a Partition

There are two things to keep in mind as you read through this section: a hard disk is divided into partitions, and each of these partitions hosts a file system. Your brand new hard disk has neither, so we begin with partitioning. In order to proceed, you must be `root`.

First you have to know the hard disk's "name" (i.e.: what file it is designated as). Suppose the new drive is set up as the slave on your primary IDE interface. In that case, it will be known as `/dev/hdb`.[2]. Please refer to the *Starter Guide*'s *Managing Your Partitions* section, which will explain how to partition a disk. DiskDrake will also create the file systems for you, so once the partitioning and file system creation steps are complete, we can proceed.

6.3. The mount and umount Commands

Now that the file system has been created, you can mount the partition. Initially, it will be empty, since the system hasn't had access to the file system to add files to it. The command to mount file systems is `mount`, 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/new` (or any other mount point you have chosen: remember that the mount point must exist). The command for mounting our newly created partition is:

```
$ mount -t ext3 /dev/hdb1 /mnt/new
```

The `-t` option is used to specify what type of file system the partition is supposed to host. The file systems you will most frequently encounter are ext2FS (the GNU/Linux file system) or ext3FS (an improved version of ext2FS with journaling capabilities), VFAT (for almost all DOS/Windows® partitions: FAT 12, 16 or 32), NTFS (for newer versions of Windows®) and ISO9660 (CD-ROM file system). If you don't specify any type, `mount` will try guessing which file system is hosted by the partition by reading the superblock.

The `-o` option is used to specify one or more mounting options. The options appropriate for a particular file system will depend on the file system being used. Refer to the mount(8) man page for more details.

Now that you've mounted your new partition, it's time to copy the entire `/usr` directory onto it:

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

or:

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

Sometimes it may happen that a device (usually the CD-ROM) is busy. If this happens most users would solve this problem by rebooting their computer. For example if `umount /dev/hdc` fails, then you could try the "lazy" `umount`. The syntax is fairly simple:

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

This command disconnects the device and closes all open handles to the device when possible. Usually you can eject the disc using the `eject <mount point|device>` command. So if the `eject` command does nothing and you don't want to reboot, use lazy unmounting.

2. Determining the name of a disk is explained in *Conventions for Naming Disks and Partitions*, page 17.

Since this partition is going to "become" our /usr directory, we need to tell the system. To do this, we edit the /etc/fstab file. It makes it possible to automate the mounting of certain file systems, especially at system start-up. It contains a series of lines describing the file systems, their mount points and other options. Here's an example of such a file:

```
/dev/hda2 / ext3 defaults 1 1
/dev/hdd /mnt/cdrom auto umask=0022,user,iocharset=utf8,noauto,ro,exec,users 0 0
/dev/fd0 /mnt/floppy supermount dev=/dev/fd0,fs=ext2:vfat,--,umask=0022,iocharset=utf8,sync 0 0
/dev/hda1 /mnt/windows ntfs umask=0,nls=utf8,ro 0 0
none /proc proc defaults 0 0
/dev/hda3 swap swap defaults 0 0
```

Each line consists of:

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

There is **always** an entry for the root file system. The Swap partitions are special since they're not visible in the tree structure, and the mount point field for those partitions contains the swap keyword. As for the /proc file system, it will be described in more detail in "*The /proc File System*", page 31. Another special file system is /dev/pts.

Also note that your system might have entries added and removed automatically from this file. This is done by fstab-sync, a command which receives special events from the Hardware Abstraction Layer (HAL) system, and manipulates the /etc/fstab file. Take a look at the fstab-sync(8) man page for more details.

Coming back to our file-system change, at this point we have moved the entire /usr hierarchy to /dev/hdb1 and we want this partition to be mounted as /usr at boot time. To accomplish this, add the following entry anywhere in the /etc/fstab file:

```
/dev/hdb1 /usr ext3 defaults 1 2
```

Now the partition will be mounted every time your system boots, and will be checked for errors if necessary.

If your partition isn't of type ext3FS you will have to change it to the correct type. Common options are ext2 and reiserfs. Also note that the last field has a value of 2. This means it will be checked after all other entries with a value of 1, and after other file systems with the same priority on the same hard disk which appear before it in /etc/fstab. Only the root (/) partition should have a value of 1.

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

One advantage of using /etc/fstab is that it simplifies the mount command syntax. To mount a file system described in the file, 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 our partition moving example, let's review what we have already done. We copied the /usr hierarchy and modified /etc/fstab so that the new partition will be mounted at start-up. But for the moment, the old

`/usr` files are still in their original place on the drive, so we need to delete them to free up space (which was, after all, our initial goal).

- To do so, you first need to switch to single user mode by issuing the `telinit 1` command on the command line. It will stop all services and prevent users from connecting to the machine.
- Next, we delete all files in the `/usr` directory. Remember that we are still referring to the "old" directory, since the newer, larger one, is not yet mounted. `rm -Rf /usr/*`.
- Finally, we mount the new `/usr` directory: `mount /usr`.

And that's it. Now, go back to multi-user mode (`telinit 3` for standard text mode or `telinit 5` for the graphical mode), and if there's no further administrative work left, you should now log off from the `root` account.

Chapter 7. Introduction to the Command Line

In "*Basic UNIX® System Concepts*", page 7 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 utility is devoted to a particular task. We will only look at a (very) small number of these utilities here. One of UNIX®'s greatest assets is the ability to combine these utilities, as we shall see later.

7.1. File-Handling Utilities

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

7.1.1. 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 they did not exist previously. Without this option, `mkdir` would just fail, complaining that these directories do not exist;
2. it will return silently if the directory you wanted 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 the 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 was not intended for creating files but for updating file access and modification times[1]. However, `touch` will create the files listed as empty files if they do not exist. 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`, if the files did not previously exist.

7.1.2. rm: Deleting Files or Directories

The `rm` command (*ReMove*) is equivalent to the DOS commands `del` and `deltree`, but has 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.

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

- `-i`: request confirmation before each deletion. Note that by default in Mandriva 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 create an empty `~/.alias` file which will prevent setting system wide aliases. Alternatively you can edit your `~/.bashrc` file to disable some of the system wide aliases by adding this line: `unalias rm cp mv`
- `-f`, the opposite of `-i`, forces deletion of the files or directories, even if the user has no write access on the files[2].

Some examples:

- `rm -i images/*.jpg file1`: deletes all files with names ending in `.jpg` in the `images` directory and deletes `file1` 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! (Well, actually there are several ways to do this but it is no trivial task and usually a preparation of the system prior to the deletion is involved.) Do not hesitate to use the `-i` option to ensure that you do not delete something by mistake.

7.1.3. mv: Moving or Renaming Files

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

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

Note that when you move multiple files the destination has to be a directory. To rename a file you simply move it to the new name.

Some options:

- `-f`: forces the operation. No warnings are given if an existing file is to be 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 already there.
- `mv foo bar`: rename file `foo` to `bar`. If a `bar` directory already existed, the effect of this command would be to move file `foo` or the whole 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. 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.

7.1.4. cp: Copying Files and Directories

`cp` (*CoPy*) is equivalent to the DOS commands `copy` and `xcopy` but has more options. Its syntax is as follows:

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

Here are the most common options `cp` has:

- `-R`: recursive copy; **mandatory** for copying a directory, even an empty directory.
- `-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:

- `cp -i /timages/* images/`: copies all files in the `/timages/` 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 with the name `bar` in the current directory.

7.2. 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 "*Basic UNIX® System Concepts*", page 7.

7.2.1. chown, chgrp: Change the Owner or 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 sub-directories in a given directory.
- `-v`: verbose mode. Displays all actions performed by `chown`; reports which files have changed ownership 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 sub-directories 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`.

7.2.2. 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, `rwx`) +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 to be exactly what was passed on the command line. 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` and `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/`: removes write permission for `group` and `others` for the whole `private/` directory, and removes the execution permission for `others`, recursively.
- `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.

7.3. 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 **Enter**, 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 too:

- `?`: 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-zBE5-7]` will match all characters between `a` and `z`, a `B`, an `E`, a `5`, a `6` or a `7`;
- `[!...]`: matches any character **not** found in the brackets. `[!a-z]`, for example, will match any character which is not a lowercase letter[3];
- `{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 meanings:

- `/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 also 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/mandriva/README` match, for example, but not `/usr/share/doc/myprog/doc/README`.
- `*[!a-z]`: all files with names which do **not** end with a lowercase letter in the current directory.

7.4. Redirections and Pipes

7.4.1. 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. Most UNIX® processes (this also includes graphical applications but excludes most daemons) use 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.

7.4.2. 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
```

3. Beware! While this is true for most languages, this may not be true for your own language setting (`locale`). This depends on the **collating order**. On some language configurations, `[a-z]` will match a, A, b, B, (...), z. And remember the fact that some languages have accentuated characters too...
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!

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 does exist, 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. Then you could 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:

```
$ 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. the standard output is not displayed) and the standard error channel (2) is redirected (>) to the errors file.

7.4.3. Pipes

Pipes are in some ways 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 which can be read by human beings. For example, the following command sent from a Terminal:

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

will send a screen shot of your desktop to the my_desktop.png [5] file in your personal directory.

5. Yes, it will indeed be a PNG image (however, the ImageMagick package needs to be installed).

7.5. Command-Line Completion

Completion is a very handy function, 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.

7.5.1. 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:

```
$ 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.

Use the **q** key to exit the file viewer.

7.5.2. 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 (`nsl<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 respectively[6]. There is also a magic character for completing a file name (`/`) and a command to recall a command from the history (`!`).

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 sub-string 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 nsl` then **Esc-!**, and the shell will automatically complete the command to `man nslookup`.

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

7.6. Starting and Handling Background Processes: Job Control

You have 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 have 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, press **Ctrl-C**: this will terminate the process and take you back to the prompt. But wait, don't do it yet! Read on.

Suppose you want the command to run while you do something else. The solution is then to put the process into the ***background***. To do this, press **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 *BackGround*) to get the desired result:

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

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 background ***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.

7.7. 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 have only mentioned a few of the available commands: Mandriva 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 (Ogg Vorbis 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 over 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 or the screen(1) man page).

Chapter 8. Text Editing: Emacs and VI

As stated in the introduction, text editing[1] is a fundamental feature when using 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, each one can prove to be a powerful tool. This is particularly because of the numerous edit modes available which provide specific editing features for a great variety of file types (Perl, C++, XML, etc.).

8.1. Emacs

Emacs is probably the most powerful text editor in existence. It can do absolutely everything and is infinitely extensible through 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. This is not to say that you will learn how to do all of that in this chapter, but you will get a good start with opening Emacs, editing one or more files, saving them and quitting Emacs.

If, after reading this, you wish to learn more about Emacs, you can have a look at this Tutorial Introduction to GNU Emacs (`http://www.lib.uchicago.edu/keith/tcl-course/emacs-tutorial.html`).

8.1.1. Short Presentation

Invoking Emacs is done as follows on the command line:

```
emacs [file1] [file2...]
```

Emacs will open every file entered as an argument into a separate buffer. If more than two files are specified at the command line, the window will be automatically split in two and there will be one part of it with the last file specified while the other part shows a list of available buffers. If you start Emacs without specifying any files on the command line you will be placed into a buffer called `*scratch*`. If you are in X, menus will be available and usable with the mouse, if you're on text mode, you can still access the menus with the **F10** key, but in this chapter we will concentrate on working strictly with the keyboard and without any menus.

8.1.2. Getting Started

It's now time to get some hands-on experience. For our example, let us start by opening two files, `file1` and `file2`. If these files do not exist, they will be created as soon as you write something in them:

```
$ emacs file1 file2
```

By typing that command, the following window will be displayed:

1. "To edit text" means to modify the content of a file containing only letters, digits, and punctuation symbols. It contains no layout information such as fonts, quadding, etc. Such files may be e-mail messages, source code, documents, or even configuration files.

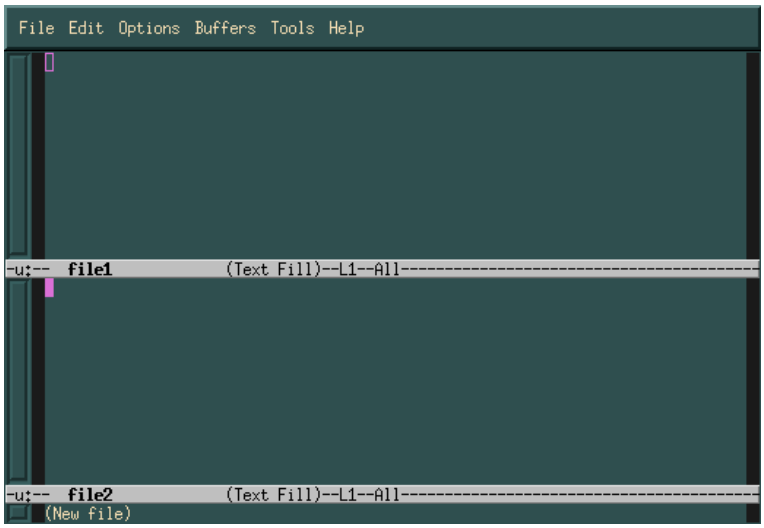


Figure 8-1. Editing Two Files at Once

As you can see, two buffers have been created. A third one is also present at the bottom of the screen (where you see `(New file)`). That is the mini-buffer. You cannot access this buffer directly. You must be invited by Emacs during interactive entries. To change the current buffer, type **Ctrl-X-O**. You type text just as in a "normal" editor, deleting characters with the **Del** or **Backspace** key.

To move around, you can use the arrow keys, or you could use the following key combinations: **Ctrl-A** to go to the beginning of the line, **Ctrl-E** to go to the end of the line, **Alt-<** or **Ctrl-Home** to go to the beginning of the buffer and **Alt->** or **Ctrl-End** to go to the end of the buffer. There are many other combinations, even ones for each of the arrow keys [2].

Once you are ready to save your changes to disk, type **Ctrl-X Ctrl-S**, or if you want to save the contents of the buffer to another file, type **Ctrl-X Ctrl-W**. Emacs will ask you for the name of the file that the contents of the buffer should be written to. You can use ***completion*** to do this by pressing the **Tab** key just like with bash.

8.1.3. Handling buffers

If you want, you can switch to displaying a single buffer on the screen. There are two ways of doing this:

- If you are in the buffer that you want to hide: type **Ctrl-X 0**.
- If you are in the buffer which you want to keep on the screen: type **Ctrl-X 1**.

There are two ways of restoring a buffer back to the screen:

- type **Ctrl-X B** and enter the name of the buffer you want, or
- type **Ctrl-X Ctrl-B**. This will open a new buffer 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 you want to get rid of the associated buffer, type **Ctrl-X K**. Emacs will then ask you which buffer it should close. By default, this will be 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 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"). To move between buffers, use the commands that were previously mentioned.

You can open other files at any time, using **Ctrl-X Ctrl-F**. Emacs will prompt you for the file name and you can again use completion if you find it more convenient.

2. Emacs has been designed to work on a great variety of machines, some of which do not have arrow keys on their keyboards. This is even more true of Vi.

8.1.4. Copy, Cut, Paste, Search

Suppose you find yourself in the following situation: figure 8-2.

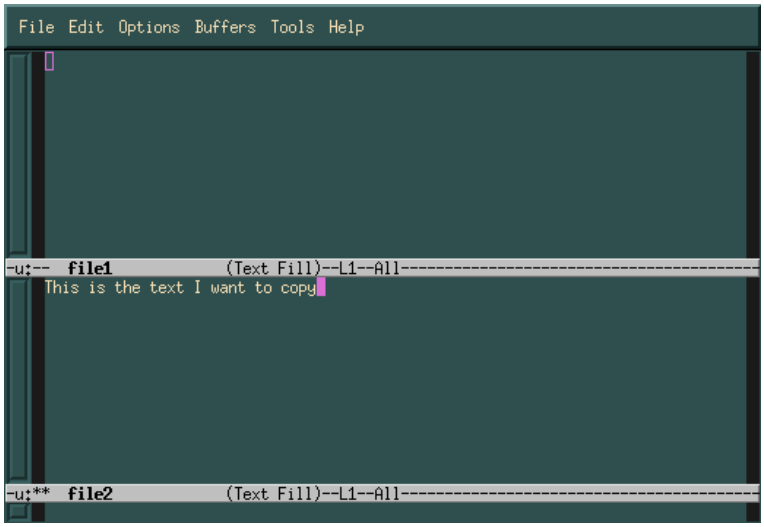


Figure 8-2. Emacs, before copying the text block

First off, you will need to select the text you want to copy. In this example we want to copy the entire sentence. The first step is to place a mark at beginning of the area. Assuming the cursor is in the position where it is in figure 8-2, the command sequence would be **Ctrl-Space** (**Control** + space bar). Emacs will display the message `Mark set` in the mini-buffer. Next, move to the beginning of the line with **Ctrl-A**. The area selected for copying or cutting is the entire area located between the mark and the cursor's current position, so in this case it will be the entire line of text. There are two command sequences available: **Alt-W** (to copy) or **Ctrl-W** (to cut). If you copy, Emacs will briefly return to the mark position so that you can view the selected area.

Finally, go to the buffer where you want the text to end up and type **Ctrl-Y**. This will give you the following result:

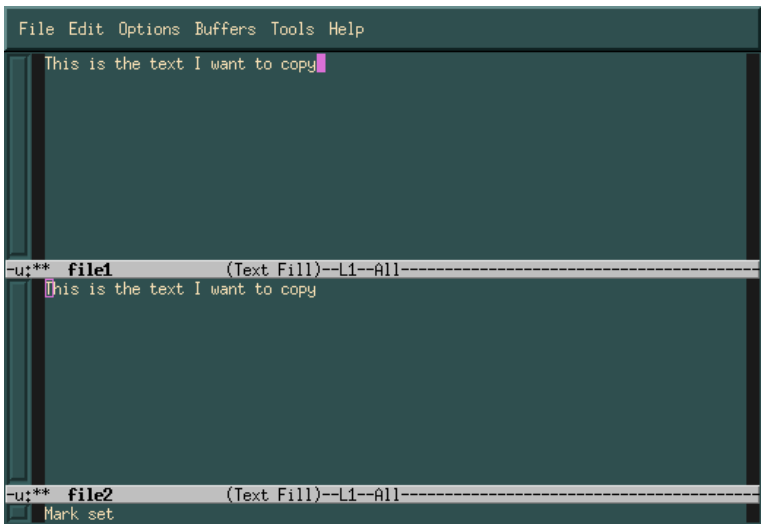


Figure 8-3. Copying Text with emacs

In fact, what you've done is copy text to Emacs's ***kill ring***. This kill ring contains all of 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 **Ctrl-Y** sequence only "pastes" the area at the top. If you want to access any of the other areas, press **Ctrl-Y** then **Alt-Y** until you get to the desired text.

To search for text, go to the desired buffer and type **Ctrl-S**. Emacs will ask you what string it should search for. To continue a search with the same string in the current buffer, just 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.

To Undo, type **Ctrl-X U** or **Ctrl-Shift--** which will undo the previous operation. You can undo as many operations as you want.

8.1.5. Quit emacs

The shortcut to quit Emacs is **Ctrl-X Ctrl-C**. If you have not saved your changes, Emacs will ask you whether you want to save your buffers or not.

8.2. Vi: the ancestor

Vi was the first full-screen editor in existence. It is one of the main programs UNIX® detractors point to, but also one of the better arguments of its defenders: while it is complicated to learn, it is also an extremely powerful tool once you get into the habit of using it. With a few keystrokes, a Vi user can move mountains, and other than Emacs, few text editors can make the same claims.

The version supplied with Mandriva Linux is in fact Vim, for *VI iMproved*, but we will refer to it as Vi throughout this chapter.

If you wish to learn more about Vi, you can have a look at this Hands-On Introduction to the Vi Editor (`http://www.library.yale.edu/wsg/docs/vi_hands_on/`) or at the Vim home page (`http://www.vim.org/`).

8.2.1. Insert Mode, Command Mode, ex Mode...

To begin using Vi we use the same sort of command line as we did with Emacs. So let us go back to our two files and type:

```
$ vi file1 file2
```

At this point, you will find yourself looking at a window resembling the following one:

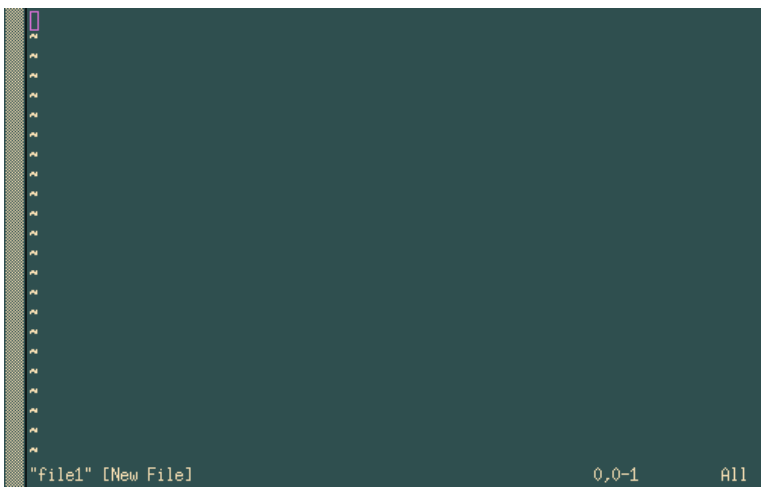


Figure 8-4. Starting position in VIM

You are now in ***command mode*** in front of the first opened file. In this mode you cannot insert text into a file. To do so you must switch to ***insert mode***.

Here are some shortcuts to inserting text:

Please note that keyboard shortcut combinations must be typed exactly as shown, Vi distinguishes between capital and lowercase letters in commands, so the **a** command is not the same as the **A** one.

- **a** and **i**: to insert text 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 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 which mode you are in). This is the only mode which will allow you to insert 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. The arrow keys will allow you to move around within the text in Command mode and Insert mode. 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. A `:` will appear at the bottom left of the screen with the cursor positioned after it. Vi will consider everything you type up to the **Enter** key as an `ex` command. If you delete the command and the **:** you typed in, you will be returned to command mode and the cursor will go back to its original position in the text.

You have command completion available while on `ex` mode, type the first letters of the command and press the **Tab** key to complete it.

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

8.2.2. Handling Buffers

To move, in the same buffer, between the files whose names were passed on the command line, 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 to either change to the desired file if it is already open, or to open another file. You may also use completion.

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

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 and the **`:q`** command closes it.

You should be aware that if you try to hide or close a buffer without saving the changes, the command will not be carried out and Vi will display this message:

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

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

8.2.3. 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. All the commands shown hereafter 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 requested text 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.

- **h**, **j**, **k**, **l**: one character left, down, up, right[3]
- **e**, **b**, **w**: to the end, beginning of the current word and the beginning of the next word.
- **^**, **0**, **$**: to the first non-blank character of the current line, the beginning of the current line, and the end of current line.
- **f<x>**: go 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 and previous occurrence of string or regexp `<string>`. For example, **/foobar** moves the cursor to 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**, ("Go") this references the line number in the file. Based on this information, you can make all sorts of combinations.

Here are some examples:

- **6b**: moves 6 words backwards;
- **c8fk**: delete all text until the eighth occurrence of the character `k` then go 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.

While many of these commands are not very intuitive, the best method to remember the commands is to practice them. But you can see that the expression "move mountains with a few keys" is not much of an exaggeration.

8.2.4. Cut, Copy, Paste

Vi contains a command which we have already seen for copying text: the **y** command. To cut text, simply use the **d** command. There are 27 memories or buffers for storing text: an anonymous memory and 26 memories named after the 26 lowercase letters of the English alphabet.

To use the anonymous memory you enter the command "as is". So the command **y12w** will copy the 12 words after the cursor into anonymous memory[4]. 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. Therefore, to copy the same 12 words into the memory `k`, you would write **"ky12w**, or **"kd12w** to cut them.

To paste the contents of the anonymous memory, use the commands **p** or **P** (for *Paste*), to insert text after or before the cursor. 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 an example:

3. A shortcut for **dl** (delete one character forward) is **x**; a shortcut for **dh** (delete one character back) is **X**; **dd** deletes the current line.
4. But only if the cursor is positioned at the beginning of the first word!

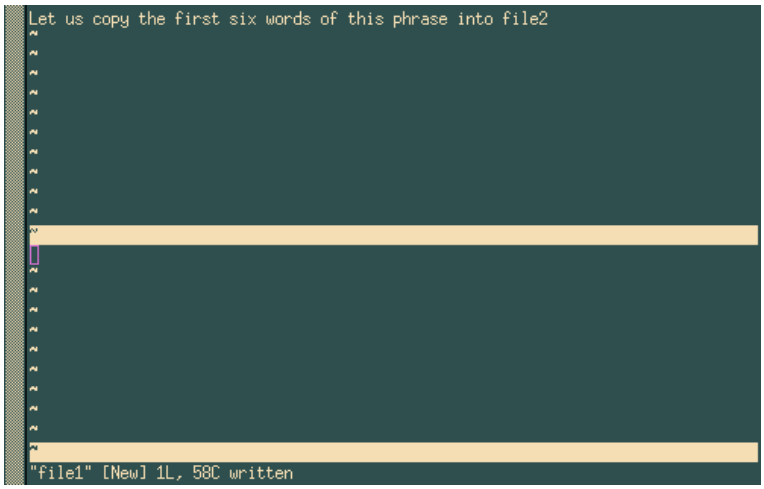


Figure 8-5. VIM, before copying the text block

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 8-6.

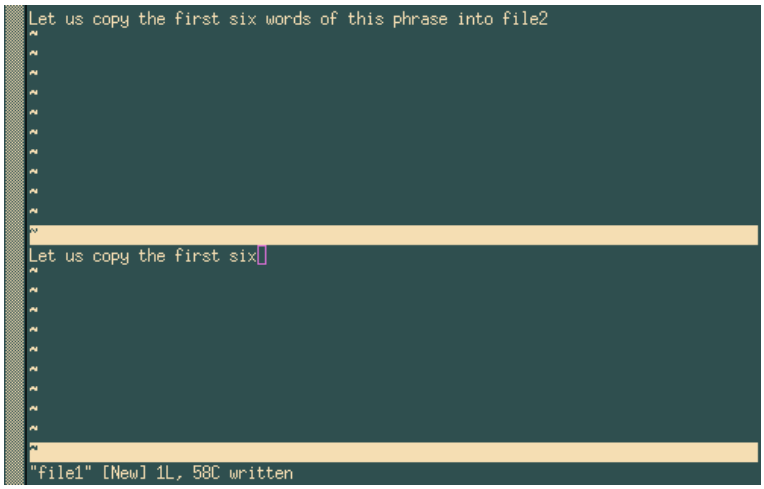


Figure 8-6. VIM, after having copied the text block

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 **/**.

8.2.5. Quit Vi

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

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

5. **`y6w`** literally means: “Yank 6 words”.

You should note that if you have several buffers, **:wq** will only write the active buffer and then close it.

8.3. 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 hopefully we have been able 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 each of these editors.

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

Chapter 9. Command-Line Utilities

The purpose of this chapter is to introduce a number of command-line tools which may prove useful for everyday use.

One of the primary strengths of GNU/Linux is the use of simple tools to achieve complex tasks. We have shown you how to tie commands together and how to clean the output to make it more visible (see *Redirections and Pipes*, page 47). Now it is time to learn about some useful tools which will give you a lot more of control and productivity.

This chapter is meant as an exercise in order for you to fully grasp each command's function and use. Therefore each command will be illustrated by an example. Don't be afraid to stop and consult the manual page for any of these commands. At the bottom of each section you will see "SEE ALSO" sections which cross-reference you to other interesting commands. You have a new place to explore on your GNU/Linux system!

9.1. File Operations and Filtering

Most command-line work is done on files. In this section we will show you how to watch and filter file content, how to take required information from files using a single command, and how to easily sort a file's content.

9.1.1. cat, tail, head, tee: File-Printing Commands

These commands have almost the same syntax: `command_name [option(s)] [file(s)]`, and may be used in a pipe. All of them are used to print part of a file according to certain criteria.

The `cat` utility concatenates files and prints the results to the standard output, which is usually the screen of your computer. This is one of the most widely used commands. For example you can use:

```
# cat /var/log/mail/info
```

to print the content of a mailer daemon log file to the standard output[1]. The `cat` command has a very useful option (`-n`) which allows you to print the line numbers.

Some files such as daemon log files (if they are running) are usually huge in size[2] and printing them completely on the screen is not very useful. Generally speaking you only need to see some lines of the file. You can use the `tail` command to do so. The following command will print, by default, the last 10 lines of the `/var/log/mail/info` file:

```
# tail /var/log/mail/info
```

Files such as logs usually vary dynamically because the daemon associated to that log constantly adds actions and events to the log file. To interactively watch these changes you can take advantage of the `-f` option:

```
# tail -f /var/log/mail/info
```

In this case all changes in the `/var/log/mail/info` file will be printed on screen immediately. Using the `tail` command with option `-f` is very helpful when you want to know how your system works. For example, looking through the `/var/log/messages` log file, you can keep up with system messages and various daemons.

If you use `tail` with more than one file it will print the name of the file on a line by itself before printing its contents. It also works with the `-f` option and is a valuable addition to see how different parts of the system interact.

You can use the `-n` option to display the last n lines of a file. For example, to display the last 2 lines, you would issue:

```
# tail -n2 /var/log/mail/info
```

1. Some examples in this section are based on real work and server log files (services, daemons, etc.). Make sure syslogd (which allows the logging of a daemon activity), and the corresponding daemon (in our case Postfix) are running, and that you work as `root`. Of course, you can always apply our examples to other files.
2. For example, the `/var/log/mail/info` file contains information about all sent mails, messages about fetching mail by users with the POP protocol, etc.

Just as for other commands, you can use different options at the same time. For example, using both `-n2` and `-f` at the same time, you start with the two last lines of the file and keep on seeing new lines as they are written to the log file.

The `head` command is similar to `tail`, but it prints the first lines of a file. The following command will print, by default, the first 10 lines of the `/var/log/mail/info` file:

```
# head /var/log/mail/info
```

As with `tail` you can use the `-n` option to specify the number of lines to be printed. For example, to print the first two, issue:

```
# head -n2 /var/log/mail/info
```

You can also use these commands together. For example, if you wish to display only lines 9 and 10, you can use a command where the `head` command will select the first 10 lines from a file and pass them through a pipe to the `tail` command.

```
# head /var/log/mail/info | tail -n2
```

The last part will then select the last 2 lines and will print them to the screen. In the same way you can select line number 20, starting from the end of the file:

```
# tail -n20 /var/log/mail/info |head -n1
```

In this example we tell `tail` to select the file's last 20 lines and pass them through a pipe to `head`. Then the `head` command prints to the screen the first line of the data obtained.

Lets suppose we want to print the result of the last example to the screen and save it to the `results.txt` file. The `tee` utility can help us. Its syntax is:

```
tee [option(s)] [file]
```

Now we can change the previous command this way:

```
# tail -n20 /var/log/mail/info |head -n1|tee results.txt
```

Lets take yet another example. We want to select the last 20 lines, save them to `results.txt`, but print on screen only the first of the 20 selected lines. Then we should type:

```
# tail -n20 /var/log/mail/info |tee results.txt |head -n1
```

The `tee` command has a useful option (`-a`) which enables you to append data to an existing file.

In the next section we will see how we can use the grep command as a filter to separate Postfix messages from other messages coming from different services.

9.1.2. grep: Locating Strings in Files

Neither the name nor the acronym ("General Regular Expression Parser") is very intuitive, but what it does and its use are simple: grep looks 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 names will precede each matching line displayed in the result. You can use the `-h` option to prevent displaying these names or you can use the `-l` option to get nothing but the matching file names. 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.

So let's go back to analyze the mailer daemon's log file. We want to find all lines in the `/var/log/mail/info` file which contain the `postfix` pattern. Then we type this command:

```
# grep postfix /var/log/mail/info
```

If we want to find all lines which do NOT contain the `postfix` pattern, we would use the `-v` option:

```
# grep -v postfix /var/log/mail/info
```

The grep command can be used in a pipe.

Let's suppose we want to find all messages about successfully sent mails. In this case we have to filter all lines which were added to the log file by the mailer daemon (contains the `postfix` pattern) and they must contain a message about successful sending (`status=sent`)[3]:

```
# grep postfix /var/log/mail/info |grep status=sent
```

In this case grep is used twice. It is allowed, but not very elegant. The same result can be achieved by using the `fgrep` utility. `fgrep` is actually a simpler method to call `grep -F`. First we need to create a file containing patterns written out one to a line. Such a file can be created this way (we use `patterns.txt` as the file name):

```
# echo -e 'status=sent\npostfix' >./patterns.txt
```

Check the results with the `cat` command. `\n` is a special pattern which means "new line".

Then we call the next command where we use the `patterns.txt` file and the `fgrep` utility instead of "double calling" of grep:

```
# fgrep -f ./patterns.txt /var/log/mail/info
```

The file `./patterns.txt` may contain as many patterns as you wish. For example, to select messages about successfully sent mails to `peter@mandriva.com`, it would be enough to add this e-mail address into our `./patterns.txt` file by running this command:

```
# echo 'peter@mandriva.com' >>./patterns.txt
```

It is clear that you can combine grep with `tail` and `head`. If we want to find messages about the second-to-last e-mail sent to `peter@mandriva.com` we would type:

```
# fgrep -f ./patterns.txt /var/log/mail/info | tail -n2 | head -n1
```

Here we apply the filter described above and place the result in a pipe for the `tail` and `head` commands. They select the last but one value from the data.

9.1.3. Regular Expressions and Filtering egrep

With grep we are stuck with patterns and fixed data. How would we find all e-mails sent to each and every employee of "ABC Company"? Listing all their e-mails would not be an easy task since we might end up missing someone or having to dig in the log file by hand.

As with `fgrep`, grep has a shortcut to the command `grep -E`: `egrep`. It takes regular expressions instead of patterns, providing us with a more powerful interface to "grep" text.

Besides what we mentioned in *Shell Globbing Patterns*, page 46 while talking about globbing patterns, here are some additional regular expressions:

- `[:alnum:]` (all letters plus all digits), `[:alpha:]` (all uppercase and lowercase letters) and `[:digit:]` (all digits) can be used instead of defining the classes of characters yourself. They have an additional bonus: they include internationalized characters and respect the localization of the system.
- `[:print:]` represents all characters which can be printed on screen.
- `[:lower:]` and `[:upper:]` represent all lowercase and uppercase letters.

3. Although it is possible to filter just by the status pattern please stay with us as we want to show you a new command with this example.

There are more classes available and you can see all of them in egrep(1). The above are the most commonly used ones.

A regular expression may be followed by one of several repetition operators:

?
: The preceding item is optional, that is: it is matched zero times or once, but not more than once.

*
: The preceding item will be matched zero or more times.

+
: The preceding item will be matched one or more times.

{n}
: The preceding item is matched exactly `n` times.

{n,}
: The preceding item is matched `n` or more times.

{n,m}
: The preceding item is matched at least `n` times, but not more than `m` times.

If you put a regular expression inside parenthesis you can recover it later. Lets say that you specified the `[:alpha:]+` expression. It might represent a word. If you want to detect words which occur twice you can put this inside parenthesis and reuse it with `\1` if it is the first group. You can have up to 9 of these "memories".

```
$ echo -e "abc def\nabc abc def\nabc1 abc1\nabcdef\nabcdabcd\nabcdef abcef" > testfile
$ egrep "([[:alpha:]]+) \1" testfile
abc abc def
$
```

The `[` and `]` characters are part of the group name so we have to include them to use that class of characters. The first `[` says that we will be using a group of characters, the second is part of the name of the group, and then there are the corresponding closing `]` characters.

The only line returned is the one which exclusively matched two groups of letters separated by a space. No other group matched the regular expression.

You can also use the `|` character to match the expression to the left of the `|` or one to the right of it. It is an operator which joins those expressions. Using the same `testfile` created above, you can try looking for expressions which contain only double words or contains double words with numbers:

```
$ egrep "([[:alpha:]]+) \1|([[:alpha:][:digit:]]+) \2" testfile
abc abc def
abc1 abc1
$
```

Note that for the second group using parenthesis we had to use `\2`, otherwise it would not match with what we wanted. A more efficient expression would be, in this particular case:

```
$ egrep "([[:alnum:]]+) \1" testfile
abc abc def
abc1 abc1
$
```

Finally to match certain characters you have to "escape" them, preceding them with a backslash. Those characters are: `?`, `+`, `{`, `|`, `(`, `)` and of course `\`. To match those you have to write: `\?`, `\+`, `\{`, `\|`, `\(`, `\)`, and `\\`.

This simple trick might help to prevent you typing repeated words in "your your" text.

Regular expressions on all tools should follow these, or very similar, rules. Taking some time to understand theses rules will help a lot with other tools such as `sed`. `sed` allows you to manipulate text, changing it using regular expressions as rules amongst other things.

9.1.4. wc: Counting Elements in Files

The `wc` command (*Word Count*) is used to count the number of lines, words and characters in files. It can also compute the length of the longest line. Its syntax is:

```
wc [option(s)] [file(s)]
```

The following options are useful:

- `-l`: print the number of lines;
- `-w`: print the number of words;
- `-m`: print the total number of characters;
- `-c`: print the number of bytes;
- `-L`: print the length of the longest line in the text.

The `wc` command prints the number of lines, words and characters by default. Here are some usage examples:

If we want to find the number of users in our system, we can type:

```
$ wc -l /etc/passwd
```

If we want to know the number of CPU's in our system, we write:

```
$ grep "model name" /proc/cpuinfo |wc -l
```

In the previous section we obtained a list of messages about successfully sent mails to e-mail addresses listed in our `./patterns.txt` file. If we want to know how many messages it contains, we can redirect our filter's results in a pipe to the `wc` command:

```
# fgrep -f ./patterns.txt /var/log/mail/info | wc -l
```

9.1.5. sort: Sorting File Content

Here is the syntax of this powerful sorting utility[4]:

```
sort [option(s)] [file(s)]
```

Let's consider sorting on part of the `/etc/passwd` file. As you can see this file is not sorted:

```
$ cat /etc/passwd
```

If we want to sort it by `login` field. Then we type:

```
$ sort /etc/passwd
```

The `sort` command sorts data in ascending order starting by the first field (in our case, the `login` field) by default. To sort data in descending order, use the `-r` option:

```
$ sort -r /etc/passwd
```

Every user has his or her own `UID` written in the `/etc/passwd` file. The following command sorts a file in ascending order using the `UID` field:

```
$ sort /etc/passwd -t":" -k3 -n
```

Here we use the following `sort` options:

4. We will only discuss `sort` briefly here. Whole books can be written about its features.

- `-t":"`: tells `sort` that the field separator is the `":"` symbol;
- `-k3`: means that sorting must be done on the third column;
- `-n`: says that the sort is to occur on numerical data, not alphabetical.

The same can be done in reverse:

```
$ sort /etc/passwd -t":" -k3 -n -r
```

Note that `sort` has two other important options:

- `-u`: perform a strict ordering: duplicate sort fields are discarded;
- `-f`: ignore case (treat lowercase characters the same way as uppercase ones).

Finally, if we want to find the user with the highest `UID` we can use this command:

```
$ sort /etc/passwd -t":" -k3 -n |tail -n1
```

where we sort the `/etc/passwd` file in ascending order according to the `UID` column, and redirect the result through a pipe to the `tail` command. The latter will print out the first value of the sorted list.

9.2. find: Finding 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 those directories. Even though it is very useful, the syntax is truly obscure, and using it requires a little practice. The general syntax is:

```
find [options] [directories] [criterion1] ... [criterionN] [action]
```

If you do not specify any directory, `find` will search the current directory. If you do not specify criteria, 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. Here are some options:

- `-xdev`: do not search on directories located on other file systems.
- `-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 links.
- `-daystart`: when using tests related to time (see below), take the beginning of current day as a time stamp instead of the default (24 hours before current time).

A criteria may be one or more of several ***atomic*** tests. Some useful tests are:

- `-type <file_type>`: search for a given type of file. `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, the pattern is treated as a ***shell globbing*** pattern (see *Shell Globbing Patterns*, page 46).
- `-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 at most or at least `n` days/minutes ago.
- `-anewer <a_file>`: find files which have been accessed more recently than file `a_file`.
- `-ctime <n>`, `-cmin <n>`, `-cnewer <file>`: same as for `-atime`, `-amin` and `-anewer`, but applies to the last time that the contents of the file were modified.
- `-regex <pattern>`: same as `-name`, but `pattern` is treated as a ***regular expression***.
- `-iregex <pattern>`: same as `-regex`, but ignoring 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 `c1`, `c2` and `c3` tests 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 to match files which match criteria `c1` or `c2` and also 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 the standard output. This is the default action.
- `-ls`: prints on the standard output the equivalent of `ls -ilds` for each file found.
- `-exec <command_line>`: executes command `command_line` on each file found. The command line `command_line` 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.
- `-ok <command>`: same as `-exec` but asks for confirmation for each command.

The best way to consolidate all of the options and parameters is with some examples. We want to find all directories in the `/usr/share` directory. We would 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 whose contents have not been modified for a month. Because you have pages from several writers, some files have the `html` extension and some have the `htm` extension. You want to link these files in the `/var/www/obsolete` directory. You would type[5]:

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

This is a fairly complex example, 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 whose 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 {} /var/www/obsolete \;
```

Here too you have to escape the `;` character from the shell. Otherwise the shell would interpret it as a command separator. If you happen to forget, `find` will complain that `-exec` is missing an argument.

A last example: you have a huge directory (`/shared/images`) containing all kinds of images. You regularly 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 which are newer than the `stamp` file, but because you got the 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 user name is **`peter`**:

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

5. Note that this example requires that `/var/www` and `/var/www/obsolete` be on the same file system!

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. A simple way to have the command run periodically is to use the cron daemon as shown in the next section.

9.3. Scheduling of Commands Startup

9.3.1. crontab: Reporting or Editing your crontab File

`crontab` allows you to execute commands at regular time intervals with the added bonus that you don't have to be logged in. `crontab` will have the output of your command 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;
- `-u <user>`: apply one of the above options for user `<user>`. Only `root` can do this.

Let's start by editing a `crontab` file. 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 Vi 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 as comments and will be ignored by `crond` (the program which is responsible for executing `crontab` files). This format is a little different for the system `crontab`, in `/etc/crontab`. There, the sixth field is the user name that should be used to start the program on the seventh field. It should be used for administrative tasks only, and for running jobs of users which exist only to enhance the system safety (such as an anti-virus user or a user that was created to run a database server). 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, it means that line is 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="your_email_address"
#
# 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. 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!

9.3.2. at: Scheduling 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, the `X11R6-contrib` package is installed, and you would 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 CTRL-d to exit
at> <EOT>
job 1 at 2005-02-23 17:30
$
```

You can specify the time in different ways:

- `now + <interval>`: means, well, now, plus an interval (Optional. 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 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 this in different ways 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` and `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.

9.4. Archiving and Data Compression

9.4.1. tar: Tape ARchiver

Just like `find`, `tar` is a long standing UNIX® utility, so its syntax is a bit special. The syntax is:

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

Here is a list of some options. Note that all of them have an equivalent long option, but you will have to refer to tar(1), as we will not list them here.

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

- `c`: used in order to create new archives.
- `x`: used to extract files from an existing archive.
- `t`: lists files from an existing archive.
- `v`: enhances verbosity. Lists files which are added to an archive or extracted from an archive. If in conjunction with the `t` option (see above), it outputs a long listing of files instead of a short one.
- `f <file_name>`: creates an archive with name `file_name`, extracts from archive `file_name` or lists files from archive `file_name`. If this parameter is omitted, the default file will be `/dev/rmt0`, which is generally

the special file associated with 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 to compress is `bzip2`.
- `p`: when extracting files from an archive, preserve all file attributes, including ownership, last access time and so on. Very useful for file system dumps.
- `r`: appends the list of files given on the command line to an existing archive. Note that the archive to which you want to append files must **not** be compressed!
- `A`: appends archives given on the command line to the one submitted with the `f` option. Similar to `r`, the archives must not be compressed in order for this to work.

There are many, many, many other options, so please refer to tar(1) for the entire list. See, for example, the `d` option.

Let's proceed with an example. Say you want to create an archive of all images in the `/shared/images` directory, compressed with `bzip2`, named `images.tar.bz2` and located in your `/home` directory. You would 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 we want its name to be `images.tar.bz2`. We may want to check if the archive is valid now. We can do this 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 the `images.tar.bz2` archive (`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 `images/cars` directory from the archive, and nothing else. Then you can type this:

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

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. 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`.

9.4.2. bzip2 and gzip: Data Compression Programs

We have already discussed these utilities when dealing with `tar`. Unlike WinZip® on 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 a different compression tool, programs such as `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 requires more resources while working. However `gzip` is still used for compatibility with older systems.

Both commands have a similar syntax:

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

If no file name 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.
- `-d`: decompress 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 doing this in `bzip2` by using the `-k` option. `gzip` has no equivalent option.

Now some examples. Let's say you want to compress all files ending with `.txt` in the current directory using `bzip2` with maximum compression. You would type:

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

Now you want to share your image archive with someone, but he does not have `bzip2`, only `gzip`. You don't need to decompress the archive and re-compress it, you can just decompress to the standard output, use a pipe, compress from standard input and redirect the output to the new archive. Like this:

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

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` for `bzip2` files and `zless` for `gzip` if you want to view compressed files directly instead of having to decompress them first. As an exercise, try and find the command you would have to type in order to view compressed files without decompressing them, and without using `bzless` or `zless`.

9.5. 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(1L) (yes, you are right: 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 10. Process Control

We have already seen in *Processes*, page 10 what a process is. We will now learn how to list processes and their characteristics, and how to manipulate them.

10.1. More About Processes

It is possible to monitor processes and to tell them to terminate, pause, continue, etc. To understand the examples we are going to examine, it is helpful to know a bit more about these processes.

10.1.1. The Process Tree

As with files, all processes which run on a GNU/Linux system are organized in a tree form. The root of this tree is `init`, a system-level process which is started at boot time. The system assigns a number (PID, *Process ID*) to each process in order to uniquely identify processes. They also inherit the PID of their parent process (PPID, *Parent Process ID*). `init` is its own father: the PID and PPID of `init` is 1.

10.1.2. 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 (`SIGx`, where `x` is the signal's name). The 32 "higher" signals (`33` to `64`) are real-time signals and are beyond 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 number 19 (`STOP`).

Signal `9` terminates 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 `kill -l` command.

10.2. Information on Processes: ps and pstree

These two commands display a list of processes currently running on the system, according to the criteria you set. `pstree` has a cleaner output when compared to `ps -f`.

10.2.1. ps

Running `ps` without arguments will show only processes initiated by you and attached to the terminal you are using:

```
$ ps
   PID TTY          TIME CMD
 18614 pts/3    00:00:00 bash
 20173 pts/3    00:00:00 ps
```

As with many UNIX® utilities, `ps` has a handful of options, the most common of which are:

- `a`: displays processes started by all users;
- `x`: 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 at which it was started.

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

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

```
$ ps ax | less
```

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

10.2.2. pstree

The `pstree` command displays processes in the form of a tree structure. One advantage is that you can immediately see the processes' parents: 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 will want to use the `-p` option to display the PID of each process, and the `-u` option to show the name of the user who started it. Because the tree structure is generally quite long, you should invoke `pstree` in the following way:

```
$ pstree -up | less
```

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

10.3. Sending Signals to Processes: kill, killall and top

10.3.1. 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 process name.

Both of these commands can optionally receive the signal number of the signal to be sent 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, enter the command:

```
$ kill 785
```

If you want to send it signal 19 (`STOP`), enter:

```
$ kill -19 785
```

Suppose instead that you want to kill a process where you know the command name. Instead of finding the process number using `ps`, you can kill the process by its name. If the process name is "mozilla" you could issue the command:

```
$ killall -9 mozilla
```

Whatever happens, you will only kill your own processes (unless you are `root`) so you do not need to worry about other users' processes if you are running on a multi-user system, since they will not be affected.

10.3.2. Mixing ps and kill: top

`top` is a program which simultaneously fulfills the functions of `ps` and `kill`, and is also used to monitor processes in real-time giving information about CPU and memory usage, running time, etc., as shown in figure 10-1.

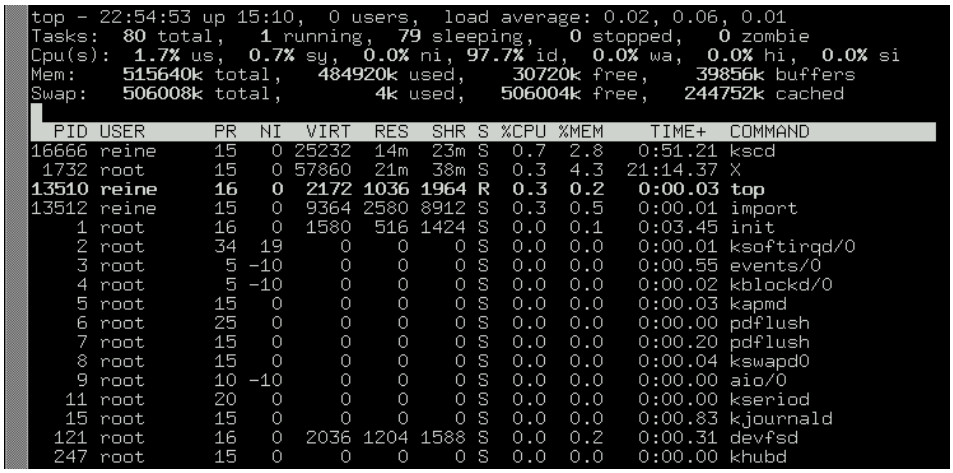


Figure 10-1. Monitoring Processes with top

The `top` utility is entirely keyboard controlled. You can access help by pressing **h**. Its most useful commands are the following:

- **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 or the name of the signal to be sent (`TERM` or `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 sorting method;
- **u**: this one 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**: by default, all processes, even sleeping ones, are displayed. This command ensures that only processes currently running are displayed (processes whose `STAT` field shows `R`, *Running*) and not the others. Using this command again takes you back to showing all processes.
- **r**: this command is used to change the priority of a selected process.

10.4. Setting Priority to Processes: nice, renice

Every process in the system is running with defined priorities, also called "nice value", which may vary from -20 (highest priority) to 19 (lowest priority). If not defined, every process will run with a default priority of 0 (the "base" scheduling priority). Processes with greater priority (lower nice value, down to -20) will be scheduled to run more often than others which have less priority (up to 19), thus granting them more processor cycles. Regular users may only lower the priority of processes they own within a range of 0 to 19. The super-user (`root`) may set the priority of any process to any value.

10.4.1. renice

If one or more processes use too many system resources, you can change their priorities instead of killing them. To do so, use the `renice` command. Its syntax is as follows:

```
renice priority [[-p] pid ...] [[-g] pgrp ...] [[-u] user ...]
```

where `priority` is the value of the priority, `pid` (use option `-p` for multiple processes) is the process ID, `pgrp` (use the `-g` option if there is more than one) is the process group ID, and `user` (`-u` for more than one) is the user name of the process' owner.

Let's suppose you have a process running with PID 785, which is performing a long and complex scientific calculation, and while it is working you want to play a game for which you need to free system resources. Then you would type:

```
$ renice +15 785
```

In this case your process could potentially take longer to complete but will not take CPU time from other processes.

If you are the system administrator and you see that some user is running too many processes and they are using too many system resources, you can change that user's processes priority with a single command:

```
# renice +20 -u peter
```

After this, all of peter's processes will have the lowest priority and will not obstruct any other processes launched by other users.

10.4.2. nice

Now that you know that you can change the priority of processes, you may wish to run a command with a defined priority. For this, use the `nice` command.

In this case you need to specify your command as an option to `nice`. Option `-n` is used to set priority value. By default `nice` sets a priority of 10.

For example, you want to create an ISO image of a Mandriva Linux installation CD-ROM:

```
$ dd if=/dev/cdrom of=~/mandriva1.iso
```

On some systems with a standard IDE CD-ROM, the process of copying large volumes of information can use too many system resources. To prevent the copying from blocking other processes, you can start the process with a lower priority by using this command:

```
$ nice -n 19 dd if=/dev/cdrom of=~/mandriva1.iso
```

Chapter 11. The Start-Up Files: init sysv

The System V start-up scheme is inherited from AT&T UNIX® and is one of the traditional UNIX® schemes to start up the system. It is responsible for starting or stopping services to bring the system into one of the default system states. Services range from basic user authentication to local graphical server or Internet services.

11.1. In the Beginning Was init

When the system starts, and after the kernel has configured everything and mounted the root file system, it executes `/sbin/init`[1]. `init` is the father of all of the system's processes and is responsible for taking the system to the desired ***run level***. We will look at runlevels later on (see *Runlevels*, page 75).

The `init` configuration file is called `/etc/inittab` and has its own manual page (inittab(5)), so we will only document a few of the possible configuration values.

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

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

This line tells `init` that `/etc/rc.d/rc.sysinit` is to be run once the system has been initialized (`si` stands for *System Init*). To determine the default runlevel, `init` will then look 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 cannot be started by any other process. For example, each of the login programs which run on the six virtual consoles are started by `init`[2]. The second virtual console is identified this way:

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

11.2. Runlevels

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

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

As already stated, `rc.sysinit` is the first file run by the system. It is responsible for setting up the basic machine configuration: keyboard type, configuration of certain devices, file-system 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 Mandriva Linux installation, you might therefore see that there are six runlevels:

- 0: complete machine stop.
- 1: ***single-user*** mode. To be used in the event of major problems or system recovery.
- 2: ***multi-user*** mode, with no network.
- 3: Multi-user mode, with networking

1. Which is why putting `/sbin` on a file system other than the root one would be a very bad idea. The kernel has not mounted any other partitions at this point, and therefore would not be able to find `/sbin/init`.
2. If you do not want six virtual consoles, you may add or remove some by modifying this file. If you wish to increase the number of consoles, you can have up to a maximum of 64. But don't forget that X also runs on a virtual console, so leave at least one console free for X.

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

Let's take a look at the content of the `rc3.d` directory:

```
$ ls /etc/rc.d/rc3.d/
K09dm@          S12syslog@    S24messagebus@      S40atd@          S91dictd-server@
S01udev@        S13partmon@   S25haldaemon@       S55sshd@         S92lisa@
S03iptables@    S15cups@      S25netfs@           S56ntpd@         S95kheader@
S05harddrake@   S17alsa@      S29numlock@         S56rawdevices@   S99local@
S10network@     S18sound@     S33nifd@            S75keytable@
S11shorewall@   S20xfs@       S34mDNSResponder@   S90crond@
$
```

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) it. The scripts are run in ascending numerical order, and if two scripts have the same number, the ascending alphabetical order will apply. We can also see that each symbolic link points to a given script located in the `/etc/init.d` directory (other than the `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: the `rc` command looks at where the link is pointing, then calls up the corresponding script with a single argument: `stop`. It then runs the `S` scripts using the same method, except that the scripts are called with a `start` parameter.

So, without discussing all the scripts, we can see that when the system goes into runlevel 3, it first runs the `K09dm` command, (i.e. `/etc/init.d/dm stop`). Next, it runs all the `S` scripts: first `S01udev`, which in turn calls `/etc/init.d/udev start`, then `S03iptables`, and so on.

Armed with this information, you can create your own complete runlevel in a few minutes (using runlevel 4, for instance), or prevent a service from starting or stopping by deleting the corresponding symbolic link.

11.2.1. Configuring Services on Run Levels

You can also use the `chkconfig` command to add, remove, activate or deactivate services from given run levels. Use `chkconfig --add service_name` to add (activate) the service `service_name` to all supported[3] run levels and `chkconfig --del service_name` to remove (deactivate) the named service from all run levels.

Issue `chkconfig --list` to see which services are available, their names, and their status in all defined run levels.

Issuing `chkconfig --levels 35 sshd on` will activate the SSH server (`sshd`) on run levels 3 and 5, while issuing `chkconfig --levels 3 sound off` will remove sound support from run level 3. If you omit the `--levels levels_list` parameter, the named service will be activated or deactivated on run levels 2, 3, 4 and 5. Note however, that you could end up enabling services on run levels without proper support for those services, so it's better to specify the run levels to be affected.

3. "Supported" run levels mean that, for example, network-related services will not be added to run level 2, which doesn't support networking.

11.2.2. Controlling Services On a Running System

Services can be controlled on a running system with the `service` command whether or not they are configured to be run on a particular run level. Its syntax is as follows:

```
service service_name action
```

Where `service_name` is the name of the service to control as listed by `chkconfig --list`, and `action` is one of:

start

Starts the named service. Please note that most services will warn you when they are already started and you pretend to start them again: use `restart` instead, see below.

stop

Stops the named service. Please note that all users connected to this service will be automatically disconnected when you stop the service.

restart

Stops and then starts the named service. It is the equivalent of running `service service_name stop && service service_name start`. Please note that all users connected to this service will be automatically disconnected when you restart the service.

other actions that are service-dependent

Different services support different actions (the previous ones are supported by all services). For example `reload` to reload the service's configuration file without restarting the service; `force-stop` to force shutdown of the service; `status` to be informed of the service's status; etc. Run `service service_name` to be informed of all actions supported by the named service.

Chapter 12. Secure Remote Access

System administrators need to connect to physically distant machines in order to edit configuration files, control services, run programs, etc. `telnet` was used to access remote systems, however it is an insecure solution. Since all communications take place over the public (and inherently insecure) Internet, system administrators need a secure remote access solution. `ssh` (which stands for **S**ecure **SH**ell) lets you access remote machines in a secure way, encrypting all communications.

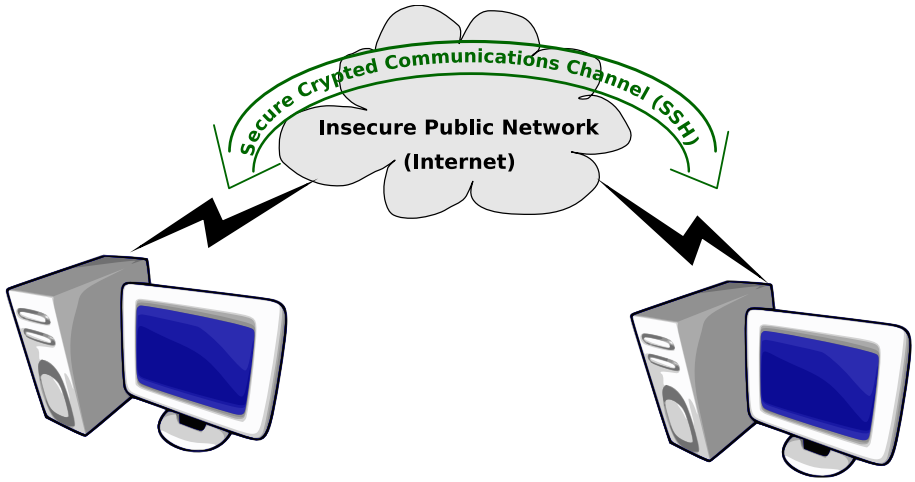


Figure 12-1. SSH Connection Schema

12.1. SSH Server Setup

By "server" we mean the machine you will be connecting to. Simply make sure that the `openssh-server` package is installed, and that the `sshd` service is running[1].

The basic SSH server setup allows users to access (or "ssh into") a machine, provided they have an account on it. If you want to restrict SSH access to a given list of users, edit the `/etc/ssh/sshd_config` file and add or modify a line to look like the following:

```
AllowUsers queen peter@192.168.0.*
```

The above example will only allow users queen and peter to connect through SSH into the machine; peter will only be allowed access from a machine in the `192.168.0.` (local) network.

Users will have to connect with their normal accounts and then use the `su` command to become `root`. To allow users to connect as `root` directly through SSH, change the line `PermitRootLogin no` to `PermitRootLogin yes`. Please bear in mind that this setting, though convenient, is not very secure.

Please refer to sshd(8) and sshd_config(5) for more information on SSH server options and setup.

12.2. SSH Client Setup

By "client" we mean the machine you will be connecting from. Simply make sure that the `openssh-clients` package is installed.

Type `ssh username@remote_machine` to connect to the `remote_machine` system with the `username` account. You will be asked for your password on the remote system. Enter it and you'll be granted access as if you were sitting at that remote system's console.

Whether you connect to one or many machines (common scenario for system administrators), the password prompt step can be bypassed by using SSH keys. Use the `ssh-keygen` command to generate your SSH key, and then the `ssh-copy-id username@remote_machine` command to copy your key over to the remote machines. When you issue the `ssh-copy-id` command you will be asked to enter your password on the

1. Issue the `service sshd start` command to start it right away. The SSH service is configured to be launched at boot time.

remote system, but only once per system. Now you can SSH directly into the remote machines without being asked for your password.

For this mechanism to work, you have to run the `ssh-add` command and enter your passphrase — created when you generated your SSH keys — each time you begin your session on the client machine.

If you get a message stating that the connection to your authentication agent could not be opened, run the `eval 'ssh-agent'` command (note the backquotes) before running `ssh-add`.

12.3. Copying Files to or From The Remote System

To transfer files to a remote system running an SSH server use the `scp` command, which stands for **S**ecure **C**o**P**y. Its syntax is as follows:

```
scp [options] local_path [user@]remote_host:[full_path_on_the_remote_host]
```

If you don't specify the `user@` part, then your login on the client machine will be used. If you omit the path on the remote machine, the file will be copied on the `user`'s home directory on the remote system. Note that the colon (`:`) separates the user name and the machine specification from the path on the remote machine.

To transfer files from the remote system to the local machine the syntax is as follows:

```
scp [options] [user@]remote_host:full_path_on_the_remote_host local_path
```

If the source path specifies a directory, then the `-r` (recursive) option is mandatory. Please refer to scp(1) for more information on `scp`'s options.

Chapter 13. Package Management From The Command Line

Rpmdrake applications are merely graphical interfaces to the powerful urpmi command line tools. For those wishing to manage their packages via the command line (useful if you are working remotely, for example) we present the most useful commands.

13.1. Installing and Removing Packages

This is done with two simple commands:

`urpmi <package_name>`

With this command the `package_name` will be installed if it exists. If the package contains the `package_name` string it will also work. If more than one package matches, you will see a numbered list of potential matches: just type the number of the one you are interested in and tap **Enter**.

If the package you are trying to install has dependencies (other packages it needs to function correctly) its list will be displayed. Review it and press the **Y** key to install all packages.

`urpme <package_name>`

This command will remove the package `package_name`. If other installed packages depend on the one you are trying to remove, a list will be presented along with the reason why they will have to be removed. Review the list and press the **Y** key to remove all packages.

Both `urpmi` and `urpme` support the `--auto` option to install or remove packages by handling the dependencies automatically.

Consult the urpmi(8) and urpme(8) man pages for more information about these commands' options.

13.2. Media Management

Software media are the different "sources" where you can install packages from. There must be at least one medium defined for `urpmi` to work. Predefined media include the ones you used to install your system (network, CD, DVD, etc.). You should define other media to install bugfixes and security updates. Adding and removing media is easy, but the syntax must be strictly respected.

13.2.1. Adding New Media

`urpmi.addmedia <name> <URL>`

This command allows you to add a new medium either from a local drive, a removable device (CD/DVD), or from the network through the HTTP, FTP, NFS, `ssh` or `rsync` protocols. The URL syntax varies for each of these media so you are encouraged to consult the urpmi.addmedia(8) man page before using it.

If you are adding a new update medium, use the `--update` option on your `urpmi.addmedia` command line.

You can use online resources such as the Easy Urpmi Page (`http://easyurpmi.zarb.org/`) if you don't know where to find media containing useful applications specially packaged for your Mandriva Linux system. The Mandriva Club (`http://club.mandriva.com/`) site also provides the Urpmi media (`http://club.mandriva.com/modules.php?name=Mirrors-list`) module for test and contribution packages.

The Mandriva Club media list is only available to its members.

13.2.2. Removing Media

```
urpmi.removemedia <name>
```

This command will simply remove the medium `name`. If you cannot remember the medium's name, issuing `urpmi.removemedia` alone on the command line will list all defined media.

13.2.3. Updating Media

```
urpmi.update <name>
```

This command scans the named medium and updates the package list associated with it. This is useful for media which change often, such as the security and bugfix updates ones. Use the `-a` option to rescan all the defined media.

13.2.4. Media Order

The order in which media are defined in the `/etc/urpmi/urpmi.cfg` file is important because it dictates the medium from which packages will be installed when there is more than one media providing a given package: packages will be installed from the first listed medium which contains them.

When adding network media, they'll be added before removable and local media. This is mainly because networked media is expected to have more up-to-date packages than removable or local ones.

13.3. Tricks and Recipes

13.3.1. Synthesized vs. Complete Lists

When adding media, there are two options for the list of packages: synthesized or complete. Use the `--probe-synthesis` option to try to find and use a synthesized list of packages, or the `--probe-hdlist` to try to find and use a complete one. Synthesized lists are smaller in size, making them more suited for users with slower network connections. However they are more limited when looking for information on packages.

13.3.2. Finding the Package which Contains a Specific File

You know you need a specific file on your system but you don't know which package provides it. Issue `urpmf <file_name>` and any packages, installed or not, which contain the `file_name` file will be displayed.

If you use the synthesized lists, `urpmf` can only search for files on already installed packages.

You can even provide only a partial name. For example `urpmf salsa` will return a list of all packages which contain a file whose name contains the word `salsa`.

```
[root@test queen]# urpmf salsa
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa-1.1.2.so
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa.la
```

```
kaffe:/usr/lib/kaffe/lib/i386/libtritonusalsa.so
```

13.3.3. Updating Packages

This command will update the named package:

```
urpmi.update -a && urpmi --update <package_name>
```

This command will automatically update all the packages which need it as Mandriva Update would do it:

```
urpmi.update -a && urpmi --update --auto-select --auto
```

If you don't have any medium specifically configured as an update medium, you have to omit the `--update` option in the urpmi commands above.

Appendix A. Glossary

account
On a UNIX® system, the combination of a name, a personal directory, a password and a shell which allows a user to connect to the system.

alias
A mechanism used in a shell in order to substitute one string for another before executing a command. You can see all aliases defined in the current session by typing `alias` at the prompt.

ACPI
Advanced Configuration and Power Interface. A feature used to recognize and configure hardware and for power management. Unlike APM, which relies on the BIOS only, ACPI also relies on the operating system, making its control more simple for the user. ACPI also brings power management capabilities to servers and workstations.

APM
Advanced Power Management. A feature used by some BIOSes 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 supported) the estimated remaining battery life. However, newer laptops are based on ACPI rather than APM.
See Also: ACPI.

ARP
Address Resolution Protocol. The Internet protocol used to dynamically map an Internet address to a physical (hardware) address on a local area network. This is limited to networks which support hardware broadcasting.

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, usually the Linux default character set, unless you have chosen to use something like UTF-8) contain ASCII as their lower half.
See Also: ISO 8859, UTF-8.

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 CD-ROM drives and magnetic tape drives. IDE controllers equipped with this extension are also referred to as EIDE (*Enhanced IDE*) controllers.
See Also: IDE.

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 can be conveyed efficiently from point to point. ATM is a circuit switched packet network technology oriented towards high speed (multi-megabit) networks.

atomic
A set of operations is said to be atomic when they execute all at once and cannot be preempted. It is commonly used for an "all or nothing" set: either all of the operations perform successfully or none of them are taken into account. It might also be used for essential or very simple operations, like the sum of two integral numbers.

background
In shell context, a process is running in the background if you can type commands that are captured by the process while it is running. It is the opposite of a foreground process.
See Also: job, foreground.

backup
A means of saving important data to a safe medium and location. Backups should be made regularly, especially with more critical information and configuration files (the most important 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
A processing mode where jobs or instructions which are submitted to the CPU are executed sequentially until all have been processed.

beep
The little noise your computer's speaker emits 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
The name given to the process of testing the beta version of a program. Programs usually get released in "alpha", "beta" and "release candidate" states for testing prior to final release.

binary
In the context of programming, binaries are the compiled, executable code.

bit
Stands for *BInary digiT*. A single digit which can take the values 0 or 1, because calculation is done in base two. It is the most basic unit of digital information.

block mode files
Files whose contents are buffered. All read/write operations for such files go through buffers, which allow for asynchronous reads and writes to the underlying hardware, which prevents the system from making disk accesses if the data is already in a buffer.
See Also: buffer, buffer cache, character mode files.

boot
The procedure taking place when a computer is ifirst switched on, where peripherals are recognized sequentially and where the operating system is loaded into memory.

boot disk
A bootable disk (floppy, CD, DVD, or any other device) containing the code necessary to load the operating system from the hard disk (sometimes it is self-sufficient).

bootloader
This is a program which starts the operating system. Many bootloaders give you the opportunity to load more than one operating system by allowing you choose between them from a menu. Bootloaders such as GRUB and LILO are popular because of this feature and are very useful in dual- or multi-boot systems.

BSD
Berkeley Software Distribution. A UNIX® variant developed at the Berkeley University computing department. This version has always been considered more technically advanced than the others, and has brought many innovations to the computing world in general and to UNIX® in particular.

buffer
A small portion of memory of fixed size, which can be associated with a block mode file, a system table, a process and so on. The buffer cache maintains coherency of all buffers.
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 that 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. Admiral Grace Hopper, having discovered this, declared "It's a bug!", and since then the term has remained. Note that this is only one of the many stories which attempt to explain the term *bug*.

byte
A sequence of, usually, eight consecutive bits, which when converted to base ten result in an integer number between 0 and 255. A byte is always "atomic" on the system, meaning that it is the smallest addressable unit.
See Also: bit.

case
When taken in the context of strings, the case is the difference between lowercase letters and uppercase (or capital) letters.

CHAP
Challenge-Handshake Authentication Protocol: A protocol used by ISPs to authenticate their clients. In this scheme, a value is sent to the client (the machine making the connection), which it uses to calculate a hash based on the value. The client sends the hash back to the server for comparison to the hash calculated by the server. This authentication method is different to PAP in that it re-authenticates on a periodic basis after the initial authentication.
See Also: PAP.

character mode files
Files whose content is not buffered. When associated with 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.

CIFS
Common Internet File System. The successor to the SMB file system, used on DOS systems.
See Also: SMB.

client
A program or computer which sporadically connects, for a given period of time, 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 which initiates the connection, the remote end receiving the call is designated as the server. It is one of the components of a **client/server system**.
See Also: server.

client/server system
System or protocol consisting of a **server** and one or more **clients**.

command line
Provided by a shell and which 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 its clones, it is the state of the program in which pressing a key does not insert the character into the file being edited, but instead performs an action specific to the 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 which is human readable (well, with some training) and which is written in some programming language (C, for example) into a binary file which is machine readable.

completion
The ability of a shell to automatically expand a substring to a filename, user name or other item, as long as there is a match.

compression
A way to shrink files or decrease the number of characters sent over a communications link. File compression programs include compress, `zip`, `gzip`, and `bzip2`.

console
This is the name given to what used to be called terminals. They were the machines (a screen plus a keyboard) connected to one big central mainframe. On PCs, 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. They allows 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. It is the basic unit of transmission across an IP network. You might also hear this called a "packet".

dependencies
The stages of compilation which need to be satisfied before going on to other compilation stages in order to successfully compile a program. This term is also used where one set of programs you wish to install are dependent on other programs which may or may not be installed on your system, in which case you may get a message telling you that the system needs to "satisfy dependencies" in order to continue the installation.

desktop
If you're using the X Window System, the desktop is the place on the screen where 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.

DHCP
Dynamic Host Configuration Protocol. A protocol designed for machines on a local network to dynamically get an IP address and other network settings from a server.

directory
Part of the file system structure. Files or other directories can be stored within a directory. Sometimes there are subdirectories (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 subdirectories 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. In graphical environments it is also known as a folder.

discrete values
Are values which are non-continuous. That is, there's some kind of "spacing" between consecutive values.

distribution
Is a term used to distinguish one GNU/Linux manufacturer's 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 DLCIs are normally assigned by the Frame Relay network provider.

DMA
Direct Memory Access. A facility used in the PC architecture which allows 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. Bus mastering allows a controller to talk to other devices without going through the CPU.

DNS
Domain Name System. The distributed name and address mechanism used in the Internet. This mechanism allows you to map a domain name to an IP address, allowing you to look up a site by domain name without knowing the IP address of the site. DNS also allows reverse lookup, allowing you to obtain a machine's IP address from its name.

DPMS
Display Power Management System. Protocol used by all modern monitors to manage power saving features. Monitors supporting these features are commonly called "green" monitors.

echo
Occurs when the characters you type are shown on the screen, such as in a user name entry field, for example. Some programs may also mask what is typed for security reasons. The example is a password prompt showing an `*`, or even nothing at all, for each typed char instead of the character itself.

editor
Is a term typically used for programs which edit text files (aka text editor). The most well-known GNU/Linux editors are the GNU Emacs (Emacs) editor and the UNIX® editor Vi.

ELF
Executable and Linking Format. This is the binary format used by most GNU/Linux distributions.

email
Stands for Electronic Mail. This is a way to send messages electronically. 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 such as pine or `mutt` which are text-mode clients, or GUI clients such as 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 a string with quotes to prevent the shell from interpreting that string. For example, when you need to use spaces in a command line and then pipe the results to some other command you have to put the first command between quotes or precede the spaces with a `\` ("escape" the command) otherwise the shell will interpret it incorrectly and your command won't work as expected.

ext2
Short for the "Extended 2 file system". This is GNU/Linux's native file system and has the characteristics of any UNIX® file system: support for special files (character devices, symbolic links, etc), file permissions and ownership, and other features.

FAQ
Frequently Asked Questions. A document containing a series of questions and answers about a specific topic. Historically, FAQs appeared in newsgroups, but this sort of document now appears on various web sites, and even commercial products have FAQs. Generally, they are very good sources of information.

FAT
File Allocation Table. File system used by DOS and Windows®.

FDDI
Fiber Distributed Digital Interface. A high-speed network physical layer, which uses optical fiber for communication instead of wire. Mostly used on large networks, mainly because of its price. It is rarely seen as a means of connection between a PC and a network switch.

FHS
File system Hierarchy Standard. A document containing guidelines for a coherent file tree organization on UNIX® systems. Mandriva Linux complies with this standard in most aspects.

FIFO
First In, First Out. A data structure or hardware buffer where items are taken out in the order they were put in. UNIX® pipes are the most common examples of FIFOs.

filesystem
Scheme used to store files on physical media (hard drive, floppy, etc.) in a consistent manner. Examples of file systems 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 single connection point to the outside network, and which filters and controls the activity on some ports, or makes sure that only some specific interfaces may have access to, or can be accessed from, the outside world.

flag
Is an indicator (usually a bit) that 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 of 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 that is currently running and has keyboard and screen control. 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 which is of a bursty or sporadic nature. Network costs are reduced by having many Frame Relay customers 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 applications to access the video RAM without the chore of having to talk to the card. All high-end graphical workstations use frame buffers.

FTP
File Transfer Protocol. This is the standard Internet protocol used to transfer files from one machine to another.

full-screen
This term is used to refer to applications that take up the entire visible area of your display.

gateway
Machine or device giving a local network access to an outside network.

GFDL
The GNU Free Documentation License. The license which applies to all Mandriva 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, the recommended solution is to replace them as much as possible by the PNG format.
See Also: PNG.

globbing
In the shell, the ability to group a certain set of file names 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.

GNU
GNU's Not Unix. The GNU project was initiated by Richard Stallman at the beginning of the 1980s, 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 to all proprietary licenses in that it applies no restrictions as to copying, modifying and redistributing the software, as long as the source code is made available. The only restriction is that the persons to whom 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. A great majority of users prefer a GUI to a CLI (*Command Line Interface*) for ease of use, even though the latter is far more versatile.

guru
An expert. Used to qualify someone particularly skilled, but also of valuable help for others.

hardware address
This is a number which 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
A file which can't be "seen" when doing a `ls` command without options. The names of hidden files begin with a `.` and are used to store the user's personal preferences and configurations for the different programs he uses. For example, bash's command history is saved into `.bash_history`, a hidden file.

home directory
Often abbreviated as "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 which are connected to a network.

HTML
HyperText Markup Language. The language used to create web documents.

HTTP
HyperText Transfer Protocol. The protocol used to connect to web sites and retrieve HTML documents or files.

icon
Is a little drawing (normally sized 16x16, 32x32, 48x48 and sometimes 64x 64 pixels) which in a graphical environment represents a document, a file or a program.

IDE
Integrated Drive Electronics. The most widely used bus on today's PCs for hard disks. An IDE bus may contain up to two devices, and the speed of the bus is limited by the device on the bus with the slowest command queue (and not the slowest transfer rate!).
See Also: ATAPI, SATA, S-ATA.

IMAP
Internet Message Access Protocol. A protocol which allows you to access your email messages on a remote server, without the need to transfer them locally first; as opposed to the POP mail retrieval protocol.
See Also: POP.

inode
Entry point leading to the contents of a file on a UNIX®-like filesystem. An inode is identified in a unique way with 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 any 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 such as the completion of an abbreviation, right justify at the end of the line, ...). One gets out of it pressing the **Esc** key, (or **Ctrl-[**).

Internet
Is a huge network which connects computers around the world.

IP address
Is a numeric address consisting (in version 4, also called IPv4) of four parts which identifies your computer on a network. IP addresses are structured in a hierarchical manner, with top level and national domains, domains, sub-domains and each machine's personal address. An IP address will 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 which never change, they are permanently assigned. Dynamic IP addresses mean that an IP address will change with each new connection to the network. Most home users typically have dynamic IP addresses while most corporate users typically have static IP addresses.

IP masquerading
This is a technique where a firewall is used to hide your computer's true IP address from the outside. Typically, any outside network connections you make through 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 on your internal network.

IRC
Internet Relay Chat. One of the few Internet standards for live speech. It allows for channel creation, private talks and file exchange. It also allows servers to 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 one channel are only visible to the people connected to that channel. Two or more users can 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 PCs, it is slowly being abandoned in favor of the PCI bus. ISA is still commonly found on SCSI cards supplied with scanners, CD writers and some other older hardware.

ISDN
Integrated Services Digital Network. A set of communication standards for voice, digital network services and video. It has been designed to eventually replace the current phone system, known as PSTN (*Public Switched Telephone Network*) or POTS (*Plain Old Telephone Service*). ISDN is known as a circuit switched data network.

ISO
International Standards Organization. A group of companies, consultants, universities and other sources which enumerate standards in various disciplines, including computing. The papers describing standards are numbered. The standard number iso9660, for example, describes the file system used on CD-ROMs.

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 to covers these needs.
See Also: ASCII, UTF-8.

ISP
Internet Service Provider. A company which sells Internet access to customers, either over telephone lines or high-bandwidth circuits such as dedicated T-1 circuits, DSL or cable.

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.

job
In a shell context, a job is a process running in the background. You can have several jobs running in the same shell and control each job independently.
See Also: foreground, background.

journaling
Journaling adds robustness to a file system, by making it transactional. Thus, instead of physically writing data at the moment it's asked for, a journal of the writes is kept, and data is written "in a block" at a later time which also has a great impact on performance and on the time needed to analyze and fix the file system, if needed.

kernel
Is the core 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 which 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 editor was started. The text areas may be recalled to be inserted again, and the structure is ring-like.

LAN
Local Area Network. Generic name given to a network of machines connected to the same physical wiring in a reduced geographical area, such as the same office or building.
See Also: WAN.

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
The last stage of the compilation process, consisting of linking together all object files in order to produce an executable file, and matching unresolved symbols with dynamic libraries (unless a static linkage has been requested, 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 which stores corresponding codes (or tags) and their meanings. 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 store a manufacturer's product codes and associated configuration information. This is one line from that table, giving information about item `CTL0001`

```
"CTL0001"       "sb"    "Creative Labs|SB16"    "sound" "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
Small documents containing the definitions of a command and its usage, to be consulted with the `man` command. The first thing one should (learn how to) read when learning about a command one isn't familiar with.

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.

minor
Number identifying the specific device we are talking about.

MPEG
Moving Picture 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...

mount point
Is the place or 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 CDs.

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 due to the "supermount" feature, so users do not need to manually mount removable media.
See Also: mount point.

MSS
The *Maximum Segment Size* is the largest quantity of data which can be transmitted at one time across an interface. If you want to prevent local fragmentation MSS would equal the MTU IP header.

MTU
The *Maximum Transmission Unit* is a parameter which determines the size of the largest datagram which 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 without fragmentation. 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 PPP interface.

multitasking
The ability of an operating system to share CPU time between several processes. At a 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 runs again. This operation is called a context switch, and is done several times per second, thereby making it 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: in preemptive multitasking the operating system is responsible for taking away the CPU and passing it to another process; cooperative multitasking is where the process itself gives back the CPU. The first variant, used by GNU/Linux, is obviously the better choice because no program can consume the entire CPU time and block other processes. 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 user 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 is 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.

NCP
NetWare Core Protocol. A protocol defined by **Novell** to access Novell NetWare file and print services.

NFS
Network File System. A network file system created by **Sun Microsystems** in order to share files across a network in a transparent way.

newsgroups
Discussion and news areas which 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 (linux), and specifically, Mandriva Linux (mandrake). Newsgroups are broken down in this fashion to make it easier to search for a particular topic.

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 may consist of an entire LAN, or just a part of it. It can export password databases, service databases, groups information and more.

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 which is made available to the 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 Amiga®OS, Mac OS®, FreeBSD®, OS/2®, UNIX®, and Windows® in all its variants.

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.

PAP
Password Authentication Protocol. A protocol used by many ISPs to authenticate their clients. In this scheme, the client (you) sends an identifier/password pair to the server, but none of the information is encrypted. CHAP is a more secure, and thus preferred, authentication protocol.
See Also: CHAP.

pager
A program which displays a text file one screen at a time, making it easy to move back and forth and search for strings in this file. We suggest you to use `less`.

password
Is a secret word or combination of words or letters which 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
A file containing a list of corrections to issue to 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** (or **full**) path is the position of a file or directory in relation to the root directory.

PCI
Peripheral Component Interconnect. A bus created by **Intel** which today is the standard bus for PC and other architectures. It is the successor to 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*...

pipe
A special UNIX® file type. One program writes data into the pipe, and another program reads the data from the other end. UNIX® pipes are FIFOs, 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 bitmap 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.

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.

PnP
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. All PCI devices are Plug'N'Play.

POP
Post Office Protocol. One common protocol used for retrieving mail from an ISP. IMAP is an example of another remote-access mail protocol.
See Also: IMAP.

porting
One of two ways to run a program on a system it was not originally intended for. For example, to be able to run a Windows®-native program under GNU/Linux (natively), it must first be ported to GNU/Linux.

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.

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 multiplication has higher precedence than the addition. If you want to evaluate the addition first, then you have to add parenthesis like this: `(4 + 3) * 2`. When you do this, you'll get 14 as the result since the parenthesis have higher precedence than the addition and the multiplication, so the operations in parenthesis get evaluated first.

preprocessors
Are compilation directives which 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 communications 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 (for example, HTTP and FTP). It maintains a cache of previous requests, so that a machine which makes a request for something which is already cached will receive it quickly, because it will get the information from the local cache. Proxies are very useful on low bandwidth networks (such as modem connections). Sometimes the proxy is the only machine able to access the outside network.

pull-down menu
Is a menu that is "rolled" with a button in one of its corners. When you press that button, the menu "unrolls" itself, showing you the full menu.

quota
Is a method of restricting disk usage and place limits on users. Administrators can restrict the size of home directories for a user by setting quota limits on specific file systems.

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 across an array of disks using different schemes. At first, this was implemented using low-cost, older, 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 may be directly accessed.

read-only mode
For a file means that the file cannot be written to. You may read its content but you cannot modify it.
See Also: read-write mode.

read-write mode
For a file, it means that the file can be written to. You may read its content 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 and others.

RFC
Request For Comments. RFCs 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.

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 contains the root directory.

route
Is the path which your datagrams take through the network to reach their destination. It is the path between one machine and another in a network.

RPM
RPM Package Manager. A packaging format developed by **Red Hat** in order to create software packages, it is used in many GNU/Linux distributions, including Mandriva Linux.

run level
Is a configuration of the system software which only allows certain selected processes to exist. Allowed processes are defined, for each runlevel, in the file `/etc/inittab`. Usually, there are seven defined runlevels: 0, 1, 2, 3, 4, 5, 6 and switching between them can only be achieved by a privileged user by means of executing the commands `init` and `telinit`.

SATA, S-ATA
Serial ATA. The successor to the ATA specification. First generation SATA has a bandwidth of 1.5Gbps, but the serial link and underlying technologies allow for much greater bandwidths, while parallel ATA has reached its practical limits with UDMA133.
See Also: ATAPI, IDE.

script
shell scripts are sequences of commands to be executed as if they were sequentially entered in the console. shell scripts are UNIX®'s (somewhat) equivalent of DOS batch files.

SCSI
Small Computers System Interface. A bus with a high throughput designed to allow for several types of peripherals to be connected to it. Unlike IDE, a SCSI bus is not limited by the speed at which the peripherals accept commands. Usually only high-end machines integrate a SCSI bus directly on the motherboard, therefore most PCs need add-on cards.

security levels
Mandriva Linux's unique feature which allows you to set different levels of restriction 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.

segmentation fault
A segmentation fault occurs when a program tries to access memory that is not allocated to it. This generally causes the program to stop immediately.

server
A program or computer which provides a feature or service and awaits connections from **clients** to execute their orders or give them the information they ask for. 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**.
See Also: client, client/server system.

shadow passwords
A password management suite on UNIX® systems in which the file containing the encrypted passwords is not world-readable, unlike that usually found with a 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 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, which only allows a single user to log into and use the system at any one time.

site dependent
Means that the information used by programs such as imake and `make` to compile some source file depends on the site, the computer architecture, the computer's installed libraries, and so on.

SMB
Server Message Block. Protocol used by Windows® machines 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 called SMTP servers.

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 as the file descriptor to which the process writes errors. It is usually the computer's screen.
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. It is usually the computer's keyboard.
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. It is usually the computer's screen.
See Also: standard error, standard input.

streamer
Is a device which takes "streams" (not interrupted or divided into shorter chunks) of characters as its input. A typical streamer is a tape drive.

SVGA
Super Video Graphics Array. The video display standard defined by VESA for the PC architecture. The resolution was at first 800x 600 x 16 colors, quickly extended to 1024x768 x 16 colors, and beyond.

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 which may be used, read the `man` pages or try to pass the `--help` switch to the program (i.e.. `program --help`).

symbolic links
Are special files, containing nothing but a string which references 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.

TCP
Transmission Control Protocol. This is the most common reliable protocol which 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 have established a connection before exchanging data.

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, such as `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.

TLDP
The Linux Documentation Project. A nonprofit organization which maintains GNU/Linux documentation. It's mostly known for documents such as HOWTOs, but it also maintains FAQs, and even a few books.
See Also: FAQ.

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 execute permission on this directory.

URL
Uniform Resource Locator. A string with a special format used to identify a resource on the Internet in a unique way. The resource may be a file, a server or other item. The syntax for a URL is `protocol://user:password@server.name[:port]/path/to/resource`.
When only a machine name is given and the protocol is `http://`, it defaults to retrieving the file that the server is configured to show by default, usually it is the `index.html` file.

username
Is a name (or more generally a word) which identifies a user on a system. Each username is attached to a unique and single UID (user ID)
See Also: login.

UTF-8
Unicode Transformation Format 8. It is an octet (8-bit) lossless encoding of Unicode characters. UTF-8 encodes each Unicode character as a variable number of 1 to 4 octets, where the number of octets depends on the integer value assigned to the Unicode character. It is an efficient encoding of Unicode documents which mostly use US-ASCII characters because it represents each character in the range U+0000 through U+007F as a single octet. UTF-8 is the default encoding for XML.
See Also: ISO 8859, ASCII.

variables
Are strings which 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 which 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.

VESA
Video Electronics Standards Association. An industry standards association aimed at the PC architecture. For example, it is the author of the SVGA standard.

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, **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 with 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 which depends on the window manager you're using.
See Also: window manager, desktop.

WAN
Wide Area Network. This network, although similar to a LAN, connects computers on networks which are not physically connected to the same wiring and are separated by a greater distance.
See Also: LAN.

wildcard
The '*' and '?' characters are used as wildcard characters and may 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.
In the context of a graphical user environment, a window is the rectangle which occupies a given running application which usually contains a title, a menu, a status bar, and the application's work area.

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, ...

workspace switcher
A little applet which allows you to switch between the available virtual desktops. It is also known as pager.
See Also: virtual desktops.

Index