Linux Fundamentals

Paul Cobbaut
Linux Fundamentals
Paul Cobbaut
It-2.0

Published Tue 17 Jul 2012 01:00:39 CEST

Abstract

This book is meant to be used in an instructor-led training. For self-study, the intent is to read this book next to a working Linux computer so you can immediately do every subject, practicing each command.

This book is aimed at novice Linux system administrators (and might be interesting and useful for home users that want to know a bit more about their Linux system). However, this book is not meant as an introduction to Linux desktop applications like text editors, browsers, mail clients, multimedia or office applications.


Feel free to contact the author:

• Paul Cobbaut: paul.cobbaut@gmail.com, http://www.linkedin.com/in/cobbaut

Contributors to the Linux Training project are:

• Serge van Ginderachter: serge@ginsys.eu, build scripts and infrastructure setup
• Ywein Van den Brande: ywein@crealaw.eu, license and legal sections
• Hendrik De Vloed: hendrik.devloed@ugent.be, buildheader.pl script

We'd also like to thank our reviewers:

• Wouter Verhelst: wo@uter.be, http://grep.be
• Geert Goossens: mail.goossens.geert@gmail.com, http://www.linkedin.com/in/geertgoossens
• Elie De Brauwer: elie@de-brauwer.be, http://www.de-brauwer.be
• Christophe Vandeplas: christophe@vandeplas.com, http://christophe.vandeplas.com
• Bert Desmet: bert@devnox.be, http://blog.bdesmet.be
• Rich Yonts: richyonts@gmail.com,

Copyright 2007-2012 Netsec BVBA, Paul Cobbaut

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled 'GNU Free Documentation License'.

# Table of Contents

I. introduction to Linux ................................................................. 1
  1. Linux history ........................................................................... 2
  2. distributions ........................................................................... 4
  3. licensing .................................................................................. 6
  4. getting Linux at home ............................................................ 10

II. first steps on the command line .................................................. 21
  5. man pages .............................................................................. 22
  6. working with directories ....................................................... 26
  7. working with files .................................................................. 35
  8. working with file contents ..................................................... 44
  9. the Linux file tree .................................................................... 51

III. shell expansion ........................................................................ 72
  10. commands and arguments ..................................................... 73
  11. control operators ................................................................... 83
  12. variables ............................................................................... 89
  13. shell history .......................................................................... 100
  14. file globbing .......................................................................... 106

IV. pipes and commands ................................................................. 113
  15. redirection and pipes ............................................................ 114
  16. filters .................................................................................... 123
  17. basic Unix tools ..................................................................... 136

V. vi .............................................................................................. 145
  18. Introduction to vi ................................................................. 146

VI. scripting ................................................................................. 156
  19. scripting introduction ........................................................... 157
  20. scripting loops ....................................................................... 163
  21. scripting parameters ........................................................... 170
  22. more scripting ....................................................................... 178

VII. local user management ............................................................ 186
  23. users ..................................................................................... 187
  24. groups .................................................................................. 207

VIII. file security ........................................................................... 213
  25. standard file permissions ..................................................... 214
  26. advanced file permissions ................................................... 225
  27. access control lists ............................................................... 231
  28. file links ............................................................................... 235

IX. Appendices ............................................................................ 242
  A. certifications ......................................................................... 243
  B. keyboard settings ................................................................. 245
  C. hardware .............................................................................. 247

Index .......................................................................................... 251
List of Tables

18.1. getting to command mode ................................................................. 147
18.2. switch to insert mode ................................................................. 147
18.3. replace and delete ................................................................. 148
18.4. undo and repeat ................................................................. 148
18.5. cut, copy and paste a line ................................................................. 148
18.6. cut, copy and paste lines ................................................................. 149
18.7. start and end of line ................................................................. 149
18.8. join two lines ................................................................. 149
18.9. words ................................................................. 150
18.10. save and exit vi ................................................................. 150
18.11. searching ................................................................. 151
18.12. replace ................................................................. 151
18.13. read files and input ................................................................. 151
18.14. text buffers ................................................................. 152
18.15. multiple files ................................................................. 152
18.16. abbreviations ................................................................. 152
23.1. Debian User Environment ................................................................. 206
23.2. Red Hat User Environment ................................................................. 206
25.1. Unix special files ................................................................. 216
25.2. standard Unix file permissions ................................................................. 217
25.3. Unix file permissions position ................................................................. 217
25.4. Octal permissions ................................................................. 220
Part I. introduction to Linux
Chapter 1. Linux history

Table of Contents

1.1. Linux history .................................................................................................... 3

This chapter briefly tells the history of Unix and where Linux fits in.

If you are eager to start working with Linux without this blah, blah, blah over history, distributions, and licensing then jump straight to Part II - Chapter 6. Working with Directories page 26.
1.1. Linux history

All modern operating systems have their roots in 1969 when Dennis Ritchie and Ken Thompson developed the C language and the Unix operating system at AT&T Bell Labs. They shared their source code (yes, there was open source back in the Seventies) with the rest of the world, including the hippies in Berkeley California. By 1975, when AT&T started selling Unix commercially, about half of the source code was written by others. The hippies were not happy that a commercial company sold software that they had written; the resulting (legal) battle ended in there being two versions of Unix in the Seventies: the official AT&T Unix, and the free BSD Unix.

In the Eighties many companies started developing their own Unix: IBM created AIX, Sun SunOS (later Solaris), HP HP-UX and about a dozen other companies did the same. The result was a mess of Unix dialects and a dozen different ways to do the same thing. And here is the first real root of Linux, when Richard Stallman aimed to end this era of Unix separation and everybody re-inventing the wheel by starting the GNU project (GNU is Not Unix). His goal was to make an operating system that was freely available to everyone, and where everyone could work together (like in the Seventies). Many of the command line tools that you use today on Linux or Solaris are GNU tools.

The Nineties started with Linus Torvalds, a Swedish speaking Finnish student, buying a 386 computer and writing a brand new POSIX compliant kernel. He put the source code online, thinking it would never support anything but 386 hardware. Many people embraced the combination of this kernel with the GNU tools, and the rest, as they say, is history.

Today more than 90 percent of supercomputers (including the complete top 10), more than half of all smartphones, many millions of desktop computers, around 70 percent of all web servers, a large chunk of tablet computers, and several appliances (dvd-players, washing machines, dsl modems, routers, ...) run Linux. It is by far the most commonly used operating system in the world.

Linux kernel version 3.2 was released in January 2012. Its source code grew by almost two hundred thousand lines (compared to version 3.1) thanks to contributions of over 4000 developers paid by about 200 commercial companies including Red Hat, Intel, Broadcom, Texas Instruments, IBM, Novell, Qualcomm, Samsung, Nokia, Oracle, Google and even Microsoft.

http://en.wikipedia.org/wiki/Dennis_Ritchie
http://en.wikipedia.org/wiki/Linus_Torvalds
http://kernel.org
http://lwn.net/Articles/472852/
http://www.linuxfoundation.org/
http://en.wikipedia.org/wiki/Linux
http://www.levenez.com/unix/ (a huge Unix history poster)
Chapter 2. distributions

Table of Contents

2.1. Red Hat ................................................................. 5
2.2. Ubuntu ................................................................. 5
2.3. Debian ................................................................. 5
2.4. Other ................................................................. 5
2.5. Which to choose ? ............................................... 5

This chapter gives a short overview of current Linux distributions.

A Linux distribution is a collection of (usually open source) software on top of a Linux kernel. A distribution (or short, distro) can bundle server software, system management tools, documentation and many desktop applications in a central secure software repository. A distro aims to provide a common look and feel, secure and easy software management and often a specific operational purpose.

Let's take a look at some popular distributions.
2.1. Red Hat

Red Hat is a billion dollar commercial Linux company that puts a lot of effort in developing Linux. They have hundreds of Linux specialists and are known for their excellent support. They give their products (Red Hat Enterprise Linux and Fedora) away for free. While Red Hat Enterprise Linux (RHEL) is well tested before release and supported for up to seven years after release, Fedora is a distro with faster updates but without support.

2.2. Ubuntu

Canonical started sending out free compact discs with Ubuntu Linux in 2004 and quickly became popular for home users (many switching from Microsoft Windows). Canonical wants Ubuntu to be an easy to use graphical Linux desktop without need to ever see a command line. Of course they also want to make a profit by selling support for Ubuntu.

2.3. Debian

There is no company behind Debian. Instead there are thousands of well organised developers that elect a Debian Project Leader every two years. Debian is seen as one of the most stable Linux distributions. It is also the basis of every release of Ubuntu. Debian comes in three versions: stable, testing and unstable. Every Debian release is named after a character in the movie Toy Story.

2.4. Other

Distributions like CentOS, Oracle Enterprise Linux and Scientific Linux are based on Red Hat Enterprise Linux and share many of the same principles, directories and system administration techniques. Linux Mint, Edubuntu and many other *buntu named distributions are based on Ubuntu and thus share a lot with Debian. There are hundreds of other Linux distributions.

2.5. Which to choose?

When you are new to Linux in 2012, go for the latest Ubuntu or Fedora. If you only want to practice the Linux command line then install one Ubuntu server and/or one CentOS server (without graphical interface).

redhat.com
ubuntu.com
debian.org
centos.org
distrowatch.com
Chapter 3. licensing

Table of Contents

3.1. about software licenses ................................................................. 7
3.2. public domain software and freeware ........................................... 7
3.3. Free Software or Open Source Software ....................................... 8
3.4. GNU General Public License ......................................................... 8
3.5. using GPLv3 software ................................................................. 8
3.6. BSD license .................................................................................. 9
3.7. other licenses ............................................................................... 9
3.8. combination of software licenses ............................................... 9

This chapter briefly explains the different licenses used for distributing operating systems software.

Many thanks go to Ywein Van den Brande for writing most of this chapter.

Ywein is an attorney at law, co-author of The International FOSS Law Book and author of Praktijkboek Informaticarecht (in Dutch).

http://ifosslawbook.org
http://www.crealaw.eu
3.1. about software licenses

There are two predominant software paradigms: Free and Open Source Software (FOSS) and proprietary software. The criteria for differentiation between these two approaches is based on control over the software. With proprietary software, control tends to lie more with the vendor, while with Free and Open Source Software it tends to be more weighted towards the end user. But even though the paradigms differ, they use the same copyright laws to reach and enforce their goals. From a legal perspective, Free and Open Source Software can be considered as software to which users generally receive more rights via their license agreement than they would have with a proprietary software license, yet the underlying license mechanisms are the same.

Legal theory states that the author of FOSS, contrary to the author of public domain software, has in no way whatsoever given up his rights on his work. FOSS supports on the rights of the author (the copyright) to impose FOSS license conditions. The FOSS license conditions need to be respected by the user in the same way as proprietary license conditions. Always check your license carefully before you use third party software.

Examples of proprietary software are AIX from IBM, HP-UX from HP and Oracle Database 11g. You are not authorised to install or use this software without paying a licensing fee. You are not authorised to distribute copies and you are not authorised to modify the closed source code.

3.2. public domain software and freeware

Software that is original in the sense that it is an intellectual creation of the author benefits copyright protection. Non-original software does not come into consideration for copyright protection and can, in principle, be used freely.

Public domain software is considered as software to which the author has given up all rights and on which nobody is able to enforce any rights. This software can be used, reproduced or executed freely, without permission or the payment of a fee. Public domain software can in certain cases even be presented by third parties as own work, and by modifying the original work, third parties can take certain versions of the public domain software out of the public domain again.

Freeware is not public domain software or FOSS. It is proprietary software that you can use without paying a license cost. However, the often strict license terms need to be respected.

Examples of freeware are Adobe Reader, Skype and Command and Conquer: Tiberian Sun (this game was sold as proprietary in 1999 and is since 2011 available as freeware).
3.3. Free Software or Open Source Software

Both the Free Software (translates to vrije software in Dutch and to Logiciel Libre in French) and the Open Source Software movement largely pursue similar goals and endorse similar software licenses. But historically, there has been some perception of differentiation due to different emphases. Where the Free Software movement focuses on the rights (the four freedoms) which Free Software provides to its users, the Open Source Software movement points to its Open Source Definition and the advantages of peer-to-peer software development.

Recently, the term free and open source software or FOSS has arisen as a neutral alternative. A lesser-used variant is free/libre/open source software (FLOSS), which uses libre to clarify the meaning of free as in freedom rather than as in at no charge.

Examples of free software are gcc, MySQL and gimp.

Detailed information about the four freedoms can be found here:

http://www.gnu.org/philosophy/free-sw.html

The open source definition can be found at:

http://www.opensource.org/docs/osd

The above definition is based on the Debian Free Software Guidelines available here:

http://www.debian.org/social_contract#guidelines

3.4. GNU General Public License

More and more software is being released under the GNU GPL (in 2006 Java was released under the GPL). This license (v2 and v3) is the main license endorsed by the Free Software Foundation. It’s main characteristic is the copyleft principle. This means that everyone in the chain of consecutive users, in return for the right of use that is assigned, needs to distribute the improvements he makes to the software and his derivative works under the same conditions to other users, if he chooses to distribute such improvements or derivative works. In other words, software which incorporates GNU GPL software, needs to be distributed in turn as GNU GPL software (or compatible, see below). It is not possible to incorporate copyright protected parts of GNU GPL software in a proprietary licensed work. The GPL has been upheld in court.

3.5. using GPLv3 software

You can use GPLv3 software almost without any conditions. If you solely run the software you even don’t have to accept the terms of the GPLv3. However, any other use - such as modifying or distributing the software - implies acceptance.
In case you use the software internally (including over a network), you may modify the software without being obliged to distribute your modification. You may hire third parties to work on the software exclusively for you and under your direction and control. But if you modify the software and use it otherwise than merely internally, this will be considered as distribution. You must distribute your modifications under GPLv3 (the copyleft principle). Several more obligations apply if you distribute GPLv3 software. Check the GPLv3 license carefully.

You create output with GPLv3 software: The GPLv3 does not automatically apply to the output.

### 3.6. BSD license

There are several versions of the original Berkeley Distribution License. The most common one is the 3-clause license ("New BSD License" or "Modified BSD License").

This is a permissive free software license. The license places minimal restrictions on how the software can be redistributed. This is in contrast to copyleft licenses such as the GPLv3 discussed above, which have a copyleft mechanism.

This difference is of less importance when you merely use the software, but kicks in when you start redistributing verbatim copies of the software or your own modified versions.

### 3.7. other licenses

FOSS or not, there are many kind of licenses on software. You should read and understand them before using any software.

### 3.8. combination of software licenses

When you use several sources or wishes to redistribute your software under a different license, you need to verify whether all licenses are compatible. Some FOSS licenses (such as BSD) are compatible with proprietary licenses, but most are not. If you detect a license incompatibility, you must contact the author to negotiate different license conditions or refrain from using the incompatible software.
Chapter 4. Getting Linux at home

Table of Contents

4.1. download a Linux CD image ............................................................ 11
4.2. download Virtualbox ................................................................. 11
4.3. create a virtual machine ......................................................... 12
4.4. attach the CD image .............................................................. 17
4.5. install Linux ........................................................................ 20

This book assumes you have access to a working Linux computer. Most companies have one or more Linux servers, if you have already logged on to it, then you’re all set (skip this chapter and go to the next).

Another option is to insert a Ubuntu Linux CD in a computer with (or without) Microsoft Windows and follow the installation. Ubuntu will resize (or create) partitions and setup a menu at boot time to choose Windows or Linux.

If you do not have access to a Linux computer at the moment, and if you are unable or unsure about installing Linux on your computer, then this chapter proposes a third option: installing Linux in a virtual machine.

Installation in a virtual machine (provided by Virtualbox) is easy and safe. Even when you make mistakes and crash everything on the virtual Linux machine, then nothing on the real computer is touched.

This chapter gives easy steps and screenshots to get a working Ubuntu server in a Virtualbox virtual machine. The steps are very similar to installing Fedora or CentOS or even Debian, and if you like you can also use VMWare instead of Virtualbox.
4.1. download a Linux CD image

Start by downloading a Linux CD image (an .ISO file) from the distribution of your choice from the Internet. Take care selecting the correct cpu architecture of your computer; choose **i386** if unsure. Choosing the wrong cpu type (like x86_64 when you have an old Pentium) will almost immediately fail to boot the CD.

4.2. download Virtualbox

Step two (when the .ISO file has finished downloading) is to download Virtualbox. If you are currently running Microsoft Windows, then download and install Virtualbox for Windows!
4.3. create a virtual machine

Now start Virtualbox. Contrary to the screenshot below, your left pane should be empty.

Click New to create a new virtual machine. We will walk together through the wizard. The screenshots below are taken on Mac OSX; they will be slightly different if you are running Microsoft Windows.
Name your virtual machine (and maybe select 32-bit or 64-bit).

Give the virtual machine some memory (512MB if you have 2GB or more, otherwise select 256MB).
Select to create a new disk (remember, this will be a virtual disk).

If you get the question below, choose vdi.
Choose **dynamically allocated** (fixed size is only useful in production or on really old, slow hardware).

Choose between 10GB and 16GB as the disk size.
Click **create** to create the virtual disk.

Click **create** to create the virtual machine.
4.4. attach the CD image

Before we start the virtual computer, let us take a look at some settings (click Settings).

Do not worry if your screen looks different, just find the button named storage.
Remember the .ISO file you downloaded? Connect this .ISO file to this virtual machine by clicking on the CD icon next to Empty.

Now click on the other CD icon and attach your ISO file to this virtual CD drive.
Verify that your download is accepted. If Virtualbox complains at this point, then you probably did not finish the download of the CD (try downloading it again).

It could be useful to set the network adapter to bridge instead of NAT. Bridged usually will connect your virtual computer to the Internet.
4.5. install Linux

The virtual machine is now ready to start. When given a choice at boot, select install and follow the instructions on the screen. When the installation is finished, you can log on to the machine and start practising Linux!
Part II. first steps on the command line
# Chapter 5. man pages

## Table of Contents

5.1. man $command ................................................................. 23  
5.2. man $configfile ................................................................. 23  
5.3. man $daemon ................................................................. 23  
5.4. man -k (apropos) ............................................................. 23  
5.5. whatis .............................................................. 23  
5.6. whereis ............................................................... 24  
5.7. man sections ............................................................... 24  
5.8. man $section $file .......................................................... 24  
5.9. man man ................................................................. 24  
5.10. mandb ............................................................... 25

This chapter will explain the use of **man** pages (also called **manual pages**) on your Unix or Linux computer.

You will learn the **man** command together with related commands like **whereis, whatis** and **mandb**.

Most Unix files and commands have pretty good man pages to explain their use. Man pages also come in handy when you are using multiple flavours of Unix or several Linux distributions since options and parameters sometimes vary.
5.1. man $command

Type man followed by a command (for which you want help) and start reading. Press q to quit the manpage. Some man pages contain examples (near the end).

paul@laika:~$ man whois
Reformatting whois(1), please wait...

5.2. man $configfile

Most configuration files have their own manual.

paul@laika:~$ man syslog.conf
Reformatting syslog.conf(5), please wait...

5.3. man $daemon

This is also true for most daemons (background programs) on your system.

paul@laika:~$ man syslogd
Reformatting syslogd(8), please wait...

5.4. man -k (apropos)

man -k (or apropos) shows a list of man pages containing a string.

paul@laika:~$ man -k syslog
lm-syslog-setup (8)  - configure laptop mode to switch syslog.conf ...
logger (1)           - a shell command interface to the syslog(3) ...
syslog-facility (8)  - Setup and remove LOCALx facility for syslogd
syslog.conf (5)      - syslogd(8) configuration file
syslogd (8)          - Linux system logging utilities.
syslogd-listfiles (8) - list system logfiles

5.5. whatis

To see just the description of a manual page, use whatis followed by a string.

paul@u810:~$ whatis route
route (8)            - show / manipulate the IP routing table
5.6. whereis

The location of a manpage can be revealed with `whereis`.

    paul@laika:~$ whereis -m whois
    whois: /usr/share/man/man1/whois.1.gz

This file is directly readable by `man`.

    paul@laika:~$ man /usr/share/man/man1/whois.1.gz

5.7. man sections

By now you will have noticed the numbers between the round brackets. `man man` will explain to you that these are section numbers. Executable programs and shell commands reside in section one.

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

5.8. man $section $file

Therefor, when referring to the man page of the passwd command, you will see it written as `passwd(1)`; when referring to the `passwd file`, you will see it written as `passwd(5)`. The screenshot explains how to open the man page in the correct section.

    [paul@RHEL52 ~]$ man passwd       # opens the first manual found
    [paul@RHEL52 ~]$ man 5 passwd     # opens a page from section 5

5.9. man man

If you want to know more about `man`, then Read The Fantastic Manual (RTFM).

`Unfortunately, manual pages do not have the answer to everything...`

    paul@laika:~$ man woman
    No manual entry for woman
5.10. mandb

Should you be convinced that a man page exists, but you can't access it, then try running `mandb`.

```
root@laika:~# mandb
0 man subdirectories contained newer manual pages.
0 manual pages were added.
0 stray cats were added.
0 old database entries were purged.
```
Chapter 6. working with directories

Table of Contents

6.1. pwd ................................................................. 27
6.2. cd ................................................................. 27
6.3. absolute and relative paths .............................. 28
6.4. path completion ............................................... 29
6.5. ls ................................................................. 29
6.6. mkdir ........................................................... 31
6.7. rmdir ............................................................ 31
6.8. practice: working with directories .................. 32
6.9. solution: working with directories ................... 33

To explore the Linux file tree, you will need some basic tools.

This chapter is a small overview of the most common commands to work with directories: `pwd`, `cd`, `ls`, `mkdir`, `rmdir`. These commands are available on any Linux (or Unix) system.

This chapter also discusses absolute and relative paths and path completion in the bash shell.
6.1. pwd

The **you are here** sign can be displayed with the `pwd` command (Print Working Directory). Go ahead, try it: Open a command line interface (like gnome-terminal, konsole, xterm, or a tty) and type `pwd`. The tool displays your current directory.

```
paul@laika:~$ pwd
/home/paul
```

6.2. cd

You can change your current directory with the `cd` command (Change Directory).

```
paul@laika$ cd /etc
paul@laika$ pwd
/etc
paul@laika$ cd /bin
paul@laika$ pwd
/bin
paul@laika$ cd /home/paul/
paul@laika$ pwd
/home/paul
```

**cd ~**

You can pull off a trick with cd. Just typing `cd` without a target directory, will put you in your home directory. Typing `cd ~` has the same effect.

```
paul@laika$ cd /etc
paul@laika$ pwd
/etc
paul@laika$ cd
paul@laika$ pwd
/home/paul
paul@laika$ cd ~
paul@laika$ pwd
/home/paul
```

**cd ..**

To go to the parent directory (the one just above your current directory in the directory tree), type `cd ..`.

```
paul@laika$ pwd
/usr/share/games
paul@laika$ cd ..
paul@laika$ pwd
/usr/share
```

*To stay in the current directory, type cd. ;-) We will see useful use of the . character representing the current directory later.*
cd -

Another useful shortcut with cd is to just type cd - to go to the previous directory.

```
paul@laika$ pwd
/home/paul
paul@laika$ cd /etc
paul@laika$ pwd
/etc
paul@laika$ cd -
/home/paul
paul@laika$ cd -
/etc
```

### 6.3. absolute and relative paths

You should be aware of **absolute and relative paths** in the file tree. When you type a path starting with a slash (/), then the **root** of the file tree is assumed. If you don't start your path with a slash, then the current directory is the assumed starting point.

The screenshot below first shows the current directory `/home/paul`. From within this directory, you have to type **cd /home** instead of **cd home** to go to the `/home` directory.

```
paul@laika$ pwd
/home/paul
paul@laika$ cd home
bash: cd: home: No such file or directory
paul@laika$ cd /home
paul@laika$ pwd
/home
```

When inside `/home`, you have to type **cd paul** instead of **cd /paul** to enter the subdirectory **paul** of the current directory `/home`.

```
paul@laika$ pwd
/home
paul@laika$ cd /paul
bash: cd: /paul: No such file or directory
paul@laika$ cd paul
paul@laika$ pwd
/home/paul
```

In case your current directory is the **root directory /**, then both **cd /home** and **cd home** will get you in the `/home` directory.

```
paul@laika$ pwd
/
paul@laika$ cd home
paul@laika$ pwd
/home
paul@laika$ cd /
paul@laika$ cd /home
paul@laika$ pwd
/home
```

This was the last screenshot with **pwd** statements. From now on, the current directory will often be displayed in the prompt. Later in this book we will explain how the shell variable **$PS1** can be configured to show this.
6.4. path completion

The tab key can help you in typing a path without errors. Typing `cd /et` followed by the tab key will expand the command line to `cd /etc/`. When typing `cd /Et` followed by the tab key, nothing will happen because you typed the wrong path (upper case E).

You will need fewer key strokes when using the tab key, and you will be sure your typed path is correct!

6.5. ls

You can list the contents of a directory with `ls`.

```
paul@pasha:~$ ls
allfiles.txt  dmesg.txt  httpd.conf  stuff  summer.txt
paul@pasha:~$
```

**ls -a**

A frequently used option with `ls` is `-a` to show all files. Showing all files means including the hidden files. When a file name on a Unix file system starts with a dot, it is considered a hidden file and it doesn't show up in regular file listings.

```
paul@pasha:~$ ls
allfiles.txt  dmesg.txt  httpd.conf  stuff  summer.txt
paul@pasha:~$ ls -a
.   allfiles.txt   .bash_profile  dmesg.txt   .lesshst  stuff
..  .bash_history  .bashrc        httpd.conf  .ssh      summer.txt
paul@pasha:~$
```

**ls -l**

Many times you will be using options with `ls` to display the contents of the directory in different formats or to display different parts of the directory. Typing just `ls` gives you a list of files in the directory. Typing `ls -l` (that is a letter L, not the number 1) gives you a long listing.

```
paul@pasha:~$ ls -l
total 23992
-rw-r--r-- 1 paul paul 24506857 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 14744 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8189 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul  4096 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul  0 2006-03-30 22:45 summer.txt
```
Another frequently used `ls` option is `-h`. It shows the numbers (file sizes) in a more human readable format. Also shown below is some variation in the way you can give the options to `ls`. We will explain the details of the output later in this book.

```
paul@pasha:~$ ls -l -h
total 24M
-rw-r--r-- 1 paul paul  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul  8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul    0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -lh
total 24M
-rw-r--r-- 1 paul paul  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul  8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul    0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -hl
total 24M
-rw-r--r-- 1 paul paul  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul  8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul    0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -h -l
total 24M
-rw-r--r-- 1 paul paul  24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul  15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul  8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul    0 2006-03-30 22:45 summer.txt
```
6.6. mkdir

Walking around the Unix file tree is fun, but it is even more fun to create your own directories with `mkdir`. You have to give at least one parameter to `mkdir`, the name of the new directory to be created. Think before you type a leading `/`.

```
paul@laika:~$ mkdir MyDir
paul@laika:~$ cd MyDir
paul@laika:~/MyDir$ ls -al
```

```
total 8
drwxr-xr-x 39 paul paul 4096 2007-01-10 21:13 ..
paul@laika:~/MyDir$ ls MyDir
```

```
paul@laika:~/MyDir$ mkdir stuff
paul@laika:~/MyDir$ mkdir otherstuff
paul@laika:~/MyDir$ ls -l
```

```
total 8
drwxr-xr-x 2 paul paul 4096 2007-01-10 21:14 otherstuff
drwxr-xr-x 2 paul paul 4096 2007-01-10 21:14 stuff
paul@laika:~/MyDir$
```

`mkdir -p`

When given the option `-p`, then `mkdir` will create parent directories as needed.

```
paul@laika:~$ mkdir -p MyDir2/MySubdir2/ThreeDeep
paul@laika:~$ ls MyDir2
MySubdir2
paul@laika:~$ ls MyDir2/MySubdir2
ThreeDeep
paul@laika:~$ ls MyDir2/MySubdir2/ThreeDeep/
```

6.7. rmdir

When a directory is empty, you can use `rmdir` to remove the directory.

```
paul@laika:~/MyDir$ rmdir otherstuff
paul@laika:~/MyDir$ ls stuff
paul@laika:~/MyDir$ cd..
paul@laika:~$ rmdir MyDir
rm: MyDir/: Directory not empty
```

```
paul@laika:~$ rmdir MyDir/stuff
paul@laika:~$ rmdir MyDir
```

`rmdir -p`

And similar to the `mkdir -p` option, you can also use `rmdir` to recursively remove directories.

```
paul@laika:~$ mkdir -p dir/subdir/subdir2
paul@laika:~$ rmdir -p dir/subdir/subdir2
paul@laika:~$
```

paul@laika:~$ rmdir -p dir/subdir/subdir2
paul@laika:~$ rmdir -p dir/subdir/subdir2
paul@laika:~$
6.8. practice: working with directories

1. Display your current directory.

2. Change to the /etc directory.

3. Now change to your home directory using only three key presses.

4. Change to the /boot/grub directory using only eleven key presses.

5. Go to the parent directory of the current directory.

6. Go to the root directory.

7. List the contents of the root directory.

8. List a long listing of the root directory.

9. Stay where you are, and list the contents of /etc.

10. Stay where you are, and list the contents of /bin and /sbin.

11. Stay where you are, and list the contents of ~.

12. List all the files (including hidden files) in your home directory.

13. List the files in /boot in a human readable format.

14. Create a directory testdir in your home directory.

15. Change to the /etc directory, stay here and create a directory newdir in your home directory.

16. Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2, and dir2 is a subdirectory from dir1).

17. Remove the directory testdir.

18. If time permits (or if you are waiting for other students to finish this practice), use and understand pushd and popd. Use the man page of bash to find information about these commands.
6.9. solution: working with directories

1. Display your current directory.
   `pwd`

2. Change to the /etc directory.
   `cd /etc`

3. Now change to your home directory using only three key presses.
   `cd (and the enter key)`

4. Change to the /boot/grub directory using only eleven key presses.
   `cd /boot/grub (use the tab key)`

5. Go to the parent directory of the current directory.
   `cd .. (with space between cd and ..)`

6. Go to the root directory.
   `cd /`

7. List the contents of the root directory.
   `ls`

8. List a long listing of the root directory.
   `ls -l`

9. Stay where you are, and list the contents of /etc.
   `ls /etc`

10. Stay where you are, and list the contents of /bin and /sbin.
    `ls /bin /sbin`

11. Stay where you are, and list the contents of ~.
    `ls ~`

12. List all the files (including hidden files) in your home directory.
    `ls -al ~`

13. List the files in /boot in a human readable format.
    `ls -lh /boot`

14. Create a directory testdir in your home directory.
    `mkdir ~/testdir`

15. Change to the /etc directory, stay here and create a directory newdir in your home directory.
cd /etc ; mkdir ~/newdir

16. Create in one command the directories ~/dir1/dir2/dir3 (dir3 is a subdirectory from dir2, and dir2 is a subdirectory from dir1 ).

mkdir -p ~/dir1/dir2/dir3

17. Remove the directory testdir.

rmdir testdir

18. If time permits (or if you are waiting for other students to finish this practice), use and understand pushd and popd. Use the man page of bash to find information about these commands.

man bash

The Bash shell has two built-in commands called pushd and popd. Both commands work with a common stack of previous directories. Pushd adds a directory to the stack and changes to a new current directory, popd removes a directory from the stack and sets the current directory.

```
paul@laika:/etc$ cd /bin
paul@laika:/bin$ pushd /lib
/lib /bin
paul@laika:/lib$ pushd /proc
/proc /lib /bin
paul@laika:/proc$
paul@laika:/proc$ popd
/lib /bin
paul@laika:/lib$
paul@laika:/lib$
paul@laika:/lib$ popd
/bin
paul@laika:/bin$
```
Chapter 7. working with files

Table of Contents

7.1. all files are case sensitive ......................................................... 36
7.2. everything is a file ................................................................. 36
7.3. file ....................................................................................... 36
7.4. touch .................................................................................. 37
7.5. rm ....................................................................................... 37
7.6. cp ....................................................................................... 38
7.7. mv ....................................................................................... 39
7.8. rename ............................................................................... 40
7.9. practice: working with files .................................................. 41
7.10. solution: working with files .................................................. 42

In this chapter we learn how to recognise, create, remove, copy and move files using commands like file, touch, rm, cp, mv and rename.
working with files

7.1. all files are case sensitive

Linux is case sensitive, this means that FILE1 is different from file1, and /etc/hosts is different from /etc/Hosts (the latter one does not exist on a typical Linux computer).

This screenshot shows the difference between two files, one with upper case W, the other with lower case w.

```
paul@laika:~/Linux$ ls
winter.txt  Winter.txt
paul@laika:~/Linux$ cat winter.txt
It is cold.
paul@laika:~/Linux$ cat Winter.txt
It is very cold!
```

7.2. everything is a file

A directory is a special kind of file, but it is still a (case sensitive!) file. Even a terminal window (/dev/pts/4) or a hard disk (/dev/sdb) is represented somewhere in the file system as a file. It will become clear throughout this course that everything on Linux is a file.

7.3. file

The file utility determines the file type. Linux does not use extensions to determine the file type. Your editor does not care whether a file ends in .TXT or .DOC. As a system administrator, you should use the file command to determine the file type. Here are some examples on a typical Linux system.

```
paul@laika:~$ file pic33.png
pic33.png: PNG image data, 3840 x 1200, 8-bit/color RGBA, non-interlaced
paul@laika:~$ file /etc/passwd
/etc/passwd: ASCII text
paul@laika:~$ file HelloWorld.c
HelloWorld.c: ASCII C program text
```

The file command uses a magic file that contains patterns to recognise file types. The magic file is located in /usr/share/file/magic. Type man 5 magic for more information.

It is interesting to point out file -s for special files like those in /dev and /proc.

```
root@debian6~# file /dev/sda
/dev/sda: block special
root@debian6~# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead...
root@debian6~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@debian6~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```
7.4. touch

One easy way to create a file is with `touch`. (We will see many other ways for creating files later in this book.)

    paul@laika:~/test$ touch file1
    paul@laika:~/test$ touch file2
    paul@laika:~/test$ touch file555
    paul@laika:~/test$ ls -l
    total 0
    -rw-r--r-- 1 paul paul 0 2007-01-10 21:40 file1
    -rw-r--r-- 1 paul paul 0 2007-01-10 21:40 file2
    -rw-r--r-- 1 paul paul 0 2007-01-10 21:40 file555

**touch -t**

Of course, `touch` can do more than just create files. Can you determine what by looking at the next screenshot? If not, check the manual for `touch`.

    paul@laika:~/test$ touch -t 200505050000 SinkoDeMayo
    paul@laika:~/test$ touch -t 130207111630 BigBattle
    paul@laika:~/test$ ls -l
    total 0
    -rw-r--r-- 1 paul paul 0 1302-07-11 16:30 BigBattle
    -rw-r--r-- 1 paul paul 0 2005-05-05 00:00 SinkoDeMayo

7.5. rm

When you no longer need a file, use `rm` to remove it. Unlike some graphical user interfaces, the command line in general does not have a waste bin or trash can to recover files. When you use `rm` to remove a file, the file is gone. Therefore, be careful when removing files!

    paul@laika:~/test$ ls
    BigBattle  SinkoDeMayo
    paul@laika:~/test$ rm BigBattle
    paul@laika:~/test$ ls
    SinkoDeMayo

**rm -i**

To prevent yourself from accidentally removing a file, you can type `rm -i`.

    paul@laika:~/Linux$ touch brel.txt
    paul@laika:~/Linux$ rm -i brel.txt
    rm: remove regular empty file `brel.txt'? y
    paul@laika:~/Linux$
rm -rf

By default, `rm -r` will not remove non-empty directories. However `rm` accepts several options that will allow you to remove any directory. The `rm -rf` statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with `rm -rf` (the `f` means force and the `r` means recursive) since being root implies that permissions don't apply to you. You can literally erase your entire file system by accident.

```
paul@laika:~$ ls test
SinkoDeMayo
paul@laika:~$ rm test
rm: cannot remove `test': Is a directory
paul@laika:~$ rm -rf test
paul@laika:~$ ls test
ls: test: No such file or directory
```

7.6. cp

To copy a file, use `cp` with a source and a target argument. If the target is a directory, then the source files are copied to that target directory.

```
paul@laika:~/test$ touch FileA
paul@laika:~/test$ ls
FileA
paul@laika:~/test$ cp FileA FileB
paul@laika:~/test$ ls
FileA  FileB
paul@laika:~/test$ mkdir MyDir
paul@laika:~/test$ ls
FileA  FileB  MyDir
paul@laika:~/test$ cp FileA MyDir/
paul@laika:~/test$ ls MyDir/
FileA
```

cp -r

To copy complete directories, use `cp -r` (the `-r` option forces recursive copying of all files in all subdirectories).

```
paul@laika:~/test$ ls
FileA  FileB  MyDir
paul@laika:~/test$ ls MyDir/
FileA
paul@laika:~/test$ cp -r MyDir MyDirB
paul@laika:~/test$ ls
FileA  FileB  MyDir  MyDirB
paul@laika:~/test$ ls MyDirB
FileA
```
working with files

**cp multiple files to directory**

You can also use `cp` to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
cp file1 file2 dir1/file3 dir1/file55 dir2
```

**cp -i**

To prevent `cp` from overwriting existing files, use the `-i` (for interactive) option.

```
paul@laika:~/test$ cp fire water
paul@laika:~/test$ cp -i fire water
cp: overwrite `water'? no
paul@laika:~/test$
```

**cp -p**

To preserve permissions and time stamps from source files, use `cp -p`.

```
paul@laika:~/perms$ cp file* cp
paul@laika:~/perms$ cp -p file* cpp
paul@laika:~/perms$ ll *
-rwx------ 1 paul paul    0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul    0 2008-08-25 13:26 file42

cp:
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:34 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:34 file42

cpp:
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:26 file42
```

**7.7. mv**

Use `mv` to rename a file or to move the file to another directory.

```
paul@laika:~/test$ touch file100
paul@laika:~/test$ ls
file100
paul@laika:~/test$ mv file100 ABC.txt
paul@laika:~/test$ ls
ABC.txt
paul@laika:~/test$
```

When you need to rename only one file then `mv` is the preferred command to use.
7.8. rename

The rename command can also be used but it has a more complex syntax to enable renaming of many files at once. Below are two examples, the first switches all occurrences of txt to png for all file names ending in .txt. The second example switches all occurrences of upper case ABC in lower case abc for all file names ending in .png . The following syntax will work on debian and ubuntu (prior to Ubuntu 7.10).

    paul@laika:~/test$ ls
    123.txt  ABC.txt
    paul@laika:~/test$ rename 's/txt/png/' *.txt
    paul@laika:~/test$ ls
    123.png  ABC.png
    paul@laika:~/test$ rename 's/ABC/abc/' *.png
    paul@laika:~/test$ ls
    123.png  abc.png
    paul@laika:~/test$

On Red Hat Enterprise Linux (and many other Linux distributions like Ubuntu 8.04), the syntax of rename is a bit different. The first example below renames all *.conf files replacing any occurrence of conf with bak. The second example renames all (*) files replacing one with ONE.

    [paul@RHEL4a test]$ ls
    one.conf  two.conf
    [paul@RHEL4a test]$ rename conf bak *.conf
    [paul@RHEL4a test]$ ls
    one.bak  two.bak
    [paul@RHEL4a test]$ rename one ONE *
    [paul@RHEL4a test]$ ls
    ONE.bak  two.bak
    [paul@RHEL4a test]$
7.9. practice: working with files

1. List the files in the /bin directory

2. Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd.


3b. Display the type of file of wolf.jpg and book.pdf

3c. Rename wolf.jpg to wolf.pdf (use mv).


4. Create a directory ~/touched and enter it.

5. Create the files today.txt and yesterday.txt in touched.

6. Change the date on yesterday.txt to match yesterday's date.

7. Copy yesterday.txt to copy.yesterday.txt

8. Rename copy.yesterday.txt to kim

9. Create a directory called ~/testbackup and copy all files from ~/touched in it.

10. Use one command to remove the directory ~/testbackup and all files in it.

11. Create a directory ~/etcbackup and copy all *.conf files from /etc in it. Did you include all subdirectories of /etc?

12. Use rename to rename all *.conf files to *.backup. (if you have more than one distro available, try it on all!)
7.10. solution: working with files

1. List the files in the /bin directory
   
   ls /bin

2. Display the type of file of /bin/cat, /etc/passwd and /usr/bin/passwd.
   
   file /bin/cat /etc/passwd /usr/bin/passwd


   wget http://linux-training.be/files/studentfiles/wolf.png

3b. Display the type of file of wolf.jpg and book.pdf
   
   file wolf.jpg book.pdf

3c. Rename wolf.jpg to wolf.pdf (use mv).
   
   mv wolf.jpg wolf.pdf

   
   file wolf.pdf book.pdf

4. Create a directory ~/touched and enter it.
   
   mkdir ~/touched ; cd ~/touched

5. Create the files today.txt and yesterday.txt in touched.
   
   touch today.txt yesterday.txt

6. Change the date on yesterday.txt to match yesterday's date.
   
   touch -t 200810251405 yesterday.txt (substitute 20081025 with yesterday)

7. Copy yesterday.txt to copy.yesterday.txt
   
   cp yesterday.txt copy.yesterday.txt

8. Rename copy.yesterday.txt to kim
   
   mv copy.yesterday.txt kim

9. Create a directory called ~/testbackup and copy all files from ~/touched in it.
   
   mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/

10. Use one command to remove the directory ~/testbackup and all files in it.
    
    rm -rf ~/testbackup

11. Create a directory ~/etcbackup and copy all *.conf files from /etc in it. Did you include all subdirectories of /etc ?
cp -r /etc/*.conf ~/etcbackup

Only *.conf files that are directly in /etc/ are copied.

12. Use rename to rename all *.conf files to *.backup. (if you have more than one distro available, try it on all!)

On RHEL: touch 1.conf 2.conf; rename conf backup *.conf

On Debian: touch 1.conf 2.conf; rename 's/conf/backup/' *.conf
Chapter 8. working with file contents

Table of Contents

8.1. head ................................................................. 45
8.2. tail ................................................................. 45
8.3. cat ................................................................. 46
8.4. tac ................................................................. 47
8.5. more and less ..................................................... 48
8.6. strings ............................................................. 48
8.7. practice: file contents .......................................... 49
8.8. solution: file contents ......................................... 50

In this chapter we will look at the contents of text files with head, tail, cat, tac, more, less and strings.

We will also get a glimpse of the possibilities of tools like cat on the command line.
8.1. head

You can use **head** to display the first ten lines of a file.

```
paul@laika:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
dev:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin/sync
```

The head command can also display the first n lines of a file.
```
paul@laika:~$ head -4 /etc/passwd
root:x:0:0:root:/root:/bin/bash
dev:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

Head can also display the first n bytes.
```
paul@laika:~$ head -c4 /etc/passwd
```

8.2. tail

Similar to head, the **tail** command will display the last ten lines of a file.

```
paul@laika:~$ tail /etc/services
vboxd 20012/udp
binkp 24554/tcp           # binkp fidonet protocol
asp 27374/tcp             # Address Search Protocol
asp 27374/udp
```

You can give **tail** the number of lines you want to see.

```
$ tail -3 count.txt
six
seven
eight
```

The **tail** command has other useful options, some of which we will use during this course.
8.3. cat

The **cat** command is one of the most universal tools. All it does is copy standard input to standard output. In combination with the shell this can be very powerful and diverse. Some examples will give a glimpse into the possibilities. The first example is simple, you can use cat to display a file on the screen. If the file is longer than the screen, it will scroll to the end.

```
paul@laika:~$ cat /etc/resolv.conf
nameserver 194.7.1.4
paul@laika:~$
```

**concatenate**

**cat** is short for **concatenate**. One of the basic uses of cat is to concatenate files into a bigger (or complete) file.

```
paul@laika:~$ echo one > part1
paul@laika:~$ echo two > part2
paul@laika:~$ echo three > part3
paul@laika:~$ cat part1 part2 part3
one
two
three
paul@laika:~$
```

**create files**

You can use **cat** to create flat text files. Type the `cat > winter.txt` command as shown in the screenshot below. Then type one or more lines, finishing each line with the enter key. After the last line, type and hold the Control (Ctrl) key and press d.

```
paul@laika:~/test$ cat > winter.txt
It is very cold today!
paul@laika:~/test$ cat winter.txt
It is very cold today!
paul@laika:~/test$
```

The **Ctrl d** key combination will send an **EOF** (End of File) to the running process ending the **cat** command.
custom end marker

You can choose an end marker for `cat` with `<<` as is shown in this screenshot. This construction is called a **here directive** and will end the `cat` command.

```
paul@laika:~/test$ cat > hot.txt <<stop
> It is hot today!
> Yes it is summer.
> stop
paul@laika:~/test$ cat hot.txt
It is hot today!
Yes it is summer.
paul@laika:~/test$
```

copy files

In the third example you will see that `cat` can be used to copy files. We will explain in detail what happens here in the bash shell chapter.

```
paul@laika:~/test$ cat winter.txt
It is very cold today!
paul@laika:~/test$ cat winter.txt > cold.txt
paul@laika:~/test$ cat cold.txt
It is very cold today!
paul@laika:~/test$
```

8.4. tac

Just one example will show you the purpose of `tac` (as the opposite of `cat`).

```
paul@laika:~/test$ cat count
one
two
three
four
paul@laika:~/test$ tac count
four
three
two
one
paul@laika:~/test$
```
8.5. more and less

The `more` command is useful for displaying files that take up more than one screen. More will allow you to see the contents of the file page by page. Use the space bar to see the next page, or `q` to quit. Some people prefer the `less` command to `more`.

8.6. strings

With the `strings` command you can display readable ascii strings found in (binary) files. This example locates the `ls` binary then displays readable strings in the binary file (output is truncated).

```bash
paul@laika:~$ which ls
/bin/ls
paul@laika:~$ strings /bin/ls
/lib/id-linux.so.2
/librt.so.1
__gmon_start__
_Jv_RegisterClasses
clock_gettime
libacl.so.1
...
```
8.7. practice: file contents

1. Display the first 12 lines of /etc/services.

2. Display the last line of /etc/passwd.

3. Use cat to create a file named count.txt that looks like this:

   One
   Two
   Three
   Four
   Five

4. Use cp to make a backup of this file to cnt.txt.

5. Use cat to make a backup of this file to catcnt.txt.

6. Display catcnt.txt, but with all lines in reverse order (the last line first).

7. Use more to display /var/log/messages.

8. Display the readable character strings from the /usr/bin/passwd command.

9. Use ls to find the biggest file in /etc.

10. Open two terminal windows (or tabs) and make sure you are in the same directory in both. Type echo this is the first line > tailing.txt in the first terminal, then issue tail -f tailing.txt in the second terminal. Now go back to the first terminal and type echo This is another line >> tailing.txt (note the double >>), verify that the tail -f in the second terminal shows both lines. Stop the tail -f with Ctrl-C.

11. Use cat to create a file named tailing.txt that contains the contents of tailing.txt followed by the contents of /etc/passwd.

12. Use cat to create a file named tailing.txt that contains the contents of tailing.txt preceded by the contents of /etc/passwd.
8.8. solution: file contents

1. Display the first 12 lines of /etc/services.
   
   head -12 /etc/services

2. Display the last line of /etc/passwd.
   
   tail -1 /etc/passwd

3. Use cat to create a file named count.txt that looks like this:
   
   cat > count.txt
   
   One
   
   Two
   
   Three
   
   Four
   
   Five (followed by Ctrl-d)

4. Use cp to make a backup of this file to cnt.txt.
   
   cp count.txt cnt.txt

5. Use cat to make a backup of this file to catcnt.txt.
   
   cat count.txt > catcnt.txt

6. Display catcnt.txt, but with all lines in reverse order (the last line first).
   
   tac catcnt.txt

7. Use more to display /var/log/messages.
   
   more /var/log/messages

8. Display the readable character strings from the /usr/bin/passwd command.
   
   strings /usr/bin/passwd

9. Use ls to find the biggest file in /etc.
   
   ls -lrS /etc

10. Open two terminal windows (or tabs) and make sure you are in the same directory in both. Type echo this is the first line > tailing.txt in the first terminal, then issue tail -f tailing.txt in the second terminal. Now go back to the first terminal and type echo This is another line >> tailing.txt (note the double >>), verify that the tail -f in the second terminal shows both lines. Stop the tail -f with Ctrl-C.

11. Use cat to create a file named tailing.txt that contains the contents of tailing.txt followed by the contents of /etc/passwd.
   
   cat /etc/passwd >> tailing.txt

12. Use cat to create a file named tailing.txt that contains the contents of tailing.txt preceded by the contents of /etc/passwd.
   
   mv tailing.txt tmp.txt ; cat /etc/passwd tmp.txt > tailing.txt
Chapter 9. the Linux file tree

Table of Contents

9.1. filesystem hierarchy standard ................................................................. 52
9.2. man hier .............................................................................................. 52
9.3. the root directory / .............................................................................. 52
9.4. binary directories ................................................................................. 53
9.5. configuration directories ................................................................. 55
9.6. data directories .................................................................................. 57
9.7. in memory directories ........................................................................ 59
9.8. /usr Unix System Resources ......................................................... 64
9.9. /var variable data .......................................................................... 66
9.10. practice: file system tree ............................................................... 68
9.11. solution: file system tree ............................................................... 70

This chapters takes a look at the most common directories in the Linux file tree. It also shows that on Unix everything is a file.
9.1. filesystem hierarchy standard

Many Linux distributions partially follow the Filesystem Hierarchy Standard. The FHS may help make more Unix/Linux file system trees conform better in the future. The FHS is available online at http://www.pathname.com/fhs/ where we read: "The filesystem hierarchy standard has been designed to be used by Unix distribution developers, package developers, and system implementers. However, it is primarily intended to be a reference and is not a tutorial on how to manage a Unix filesystem or directory hierarchy."

9.2. man hier

There are some differences in the filesystems between Linux distributions. For help about your machine, enter man hier to find information about the file system hierarchy. This manual will explain the directory structure on your computer.

9.3. the root directory /

All Linux systems have a directory structure that starts at the root directory. The root directory is represented by a forward slash, like this: /. Everything that exists on your Linux system can be found below this root directory. Let's take a brief look at the contents of the root directory.

[paul@RHELv4u3 ~]$ ls /
bin  dev  home  media  mnt  proc  sbin  srv  tftpboot  usr
boot  etc  lib  misc  opt  root  selinux  sys  tmp  var
9.4. binary directories

**Binaries** are files that contain compiled source code (or machine code). Binaries can be **executed** on the computer. Sometimes binaries are called **executables**.

/bin

The /bin directory contains **binaries** for use by all users. According to the FHS the /bin directory should contain /bin/cat and /bin/date (among others).

In the screenshot below you see common Unix/Linux commands like cat, cp, cpio, date, dd, echo, grep, and so on. Many of these will be covered in this book.

```
paul@laika:~$ ls /bin
archdetect       egrep             mt               setupcon
autopartition    false             mt-gnu           sh
bash             fgconsole         mv               sh.distrib
bunzip2          fgrep             nano             sleep
bzcchar          fuser             nc               stralign
bzcmp            fusermount        nc.traditional   stty
bzdiff           get_mountoptions  netcat           su
bzgrep           grep              netstat          sync
bzexe            gunzip            ntfs-3g          sysfs
bzfgrep          gzexe             ntfs-3g.probe    tailf
bzgrep           hostname          parted_devices   tar
bzzip2           hostname          parted_server    tempfile
bzzip2recover    hw-detect         partman          touch
bzless           ip                partman-commit   true
bzmore           kbd_mode          perform_recipe   ulockmgr
cat              kill              pidof            umount
...
```

other /bin directories

You can find a /bin **subdirectory** in many other directories. A user named serena could put her own programs in /home/serena/bin.

Some applications, often when installed directly from source will put themselves in /opt. A **samba server** installation can use /opt/samba/bin to store its binaries.

/sbin

/sbin contains binaries to configure the operating system. Many of the **system binaries** require root privilege to perform certain tasks.

Below a screenshot containing system binaries to change the ip address, partition a disk and create an ext4 file system.

```
paul@ubu1010:~$ ls -l /sbin/ifconfig /sbin/fdisk /sbin/mkfs.ext4
-rw-r-xr-x 1 root root 97172 2011-02-02 09:56 /sbin/fdisk
-rw-r-xr-x 1 root root 65708 2010-07-02 09:27 /sbin/ifconfig
-rw-r-xr-x 1 root root 55140 2010-08-18 18:01 /sbin/mkfs.ext4
```
the Linux file tree

/lib

Binaries found in /bin and /sbin often use shared libraries located in /lib. Below is a screenshot of the partial contents of /lib.

```
paul@laika:~$ ls /lib/libc*
/lib/libc-2.5.so /lib/libcfont.so.0.0.0 /lib/libcom_err.so.2.1
/lib/libcap.so.1 /lib/libcidn-2.5.so /lib/libconsole.so.0
/lib/libcap.so.1.10 /lib/libcidn.so.1 /lib/libconsole.so.0.0.0
/lib/libcfont.so.0 /lib/libcom_err.so.2 /lib/libcrypt-2.5.so
```

/lib/modules

Typically, the Linux kernel loads kernel modules from /lib/modules/$kernel-version/. This directory is discussed in detail in the Linux kernel chapter.

/lib32 and /lib64

We currently are in a transition between 32-bit and 64-bit systems. Therefore, you may encounter directories named /lib32 and /lib64 which clarify the register size used during compilation time of the libraries. A 64-bit computer may have some 32-bit binaries and libraries for compatibility with legacy applications. This screenshot uses the file utility to demonstrate the difference.

```
paul@laika:~$ file /lib32/libc-2.5.so
/lib32/libc-2.5.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), for GNU/Linux 2.6.0, stripped
paul@laika:~$ file /lib64/libcap.so.1.10
/lib64/libcap.so.1.10: ELF 64-bit LSB shared object, AMD x86-64, version 1 (SYSV), stripped
```

The ELF (Executable and Linkable Format) is used in almost every Unix-like operating system since System V.

/opt

The purpose of /opt is to store optional software. In many cases this is software from outside the distribution repository. You may find an empty /opt directory on many systems.

A large package can install all its files in /bin, /lib, /etc subdirectories within /opt/$packagename/. If for example the package is called wp, then it installs in /opt/wp, putting binaries in /opt/wp/bin and manpages in /opt/wp/man.
9.5. configuration directories

/boot

The /boot directory contains all files needed to boot the computer. These files don’t change very often. On Linux systems you typically find the /boot/grub directory here. /boot/grub contains /boot/grub/grub.cfg (older systems may still have /boot/grub/grub.conf) which defines the boot menu that is displayed before the kernel starts.

/etc

All of the machine-specific configuration files should be located in /etc. Historically /etc stood for etcetera, today people often use the Editable Text Configuration backronym.

Many times the name of a configuration files is the same as the application, daemon, or protocol with .conf added as the extension.

```
paul@laika:~$ ls /etc/*.conf
/etc/adduser.conf     /etc/ld.so.conf     /etc/scrollkeeper.conf
/etc/britty.conf     /etc/lftp.conf      /etc/sysctl.conf
/etc/certificates.conf /etc/libao.conf    /etc/syslog.conf
/etc/cvs-cron.conf   /etc/logrotate.conf /etc/ucf.conf
/etc/ddclient.conf   /etc/ltrace.conf    /etc/unconf.conf
/etc/debconf.conf    /etc/mke2fs.conf    /etc/updatedb.conf
/etc/deluser.conf    /etc/netscsid.conf  /etc/usplash.conf
/etc/ddclient.conf   /etc/nsswitch.conf  /etc/uswsusp.conf
/etc/ddclient.conf   /etc/pam.conf       /etc/wodim.conf
/etc/ddclient.conf   /etc/pnm2ppa.conf   /etc/wvdial.conf
/etc/ddclient.conf   /etc/pppoe.conf     /etc/wvdial.conf
/etc/ddclient.conf   /etc/resolv.conf   /etc/wvdial.conf
paul@laika:~$
```

There is much more to be found in /etc.

/etc/init.d/

A lot of Unix/Linux distributions have an /etc/init.d directory that contains scripts to start and stop daemons. This directory could disappear as Linux migrates to systems that replace the old init way of starting all daemons.

/etc/X11/

The graphical display (aka X Window System or just X) is driven by software from the X.org foundation. The configuration file for your graphical display is /etc/X11/xorg.conf.
The skeleton directory /etc/skel is copied to the home directory of a newly created user. It usually contains hidden files like a .bashrc script.

This directory, which is not mentioned in the FHS, contains a lot of Red Hat Enterprise Linux configuration files. We will discuss some of them in greater detail. The screenshot below is the /etc/sysconfig directory from RHELv4u4 with everything installed.

```
paul@RHELv4u4:~$ ls /etc/sysconfig/
apmd   firstboot  irda   network   saslauthd
apm-scripts grub  irqbalance networking selpinux
authconfig hidd  keyboard  ntpd   spamassassin
autofs  httpd  kudzu  openlib.conf squid
bluetooth hwdconf  lm_sensors pand   syslog
clock  i18n   mouse   pcmcia sys-config-sec
console init  mouse.B  pgsq1   sys-config-users
crond installinfo named prelink sys-logviewer
desktop ipmi  netdump rawdevices tux
diskdump iptables netdump_id_dsa rhn   vncservers
dund iptables-cfg netdump_id_dsa.p samba xinetd
paul@RHELv4u4:~$
```

The file /etc/sysconfig/firstboot tells the Red Hat Setup Agent not to run at boot time. If you want to run the Red Hat Setup Agent at the next reboot, then simply remove this file, and run chkconfig --level 5 firstboot on. The Red Hat Setup Agent allows you to install the latest updates, create a user account, join the Red Hat Network and more. It will then create the /etc/sysconfig/firstboot file again.

```
paul@RHELv4u4:~$ cat /etc/sysconfig/firstboot
RUN_FIRSTBOOT=NO
```

The /etc/sysconfig/harddisks file contains some parameters to tune the hard disks. The file explains itself.

You can see hardware detected by kudzu in /etc/sysconfig/hwconf. Kudzu is software from Red Hat for automatic discovery and configuration of hardware.

The keyboard type and keymap table are set in the /etc/sysconfig/keyboard file. For more console keyboard information, check the manual pages of keymaps(5), dumpkeys(1), loadkeys(1) and the directory /lib/kbd/keymaps/.

```
root@RHELv4u4:/etc/sysconfig# cat keyboard
KEYBOARDTYPE="pc"
KEYTABLE="us"
```

We will discuss networking files in this directory in the networking chapter.
9.6. data directories

/home

Users can store personal or project data under /home. It is common (but not mandatory by the fhs) practice to name the users home directory after the user name in the format /home/$USERNAME. For example:

```
paul@ubuntu606:~$ ls /home
geert  annik  sandra  paul  tom
```

Besides giving every user (or every project or group) a location to store personal files, the home directory of a user also serves as a location to store the user profile. A typical Unix user profile contains many hidden files (files whose file name starts with a dot). The hidden files of the Unix user profiles contain settings specific for that user.

```
paul@ubuntu606:~$ ls -d /home/paul/.*
/home/paul/.               /home/paul/.bash_profile  /home/paul/.ssh
/home/paul/..             /home/paul/.bashrc        /home/paul/.viminfo
/home/paul/.bash_history  /home/paul/.lesshst
```

/root

On many systems /root is the default location for personal data and profile of the root user. If it does not exist by default, then some administrators create it.

/srv

You may use /srv for data that is served by your system. The FHS allows locating cvs, rsync, ftp and www data in this location. The FHS also approves administrative naming in /srv, like /srv/project55/ftp and /srv/sales/www.

On Sun Solaris (or Oracle Solaris) /export is used for this purpose.

/media

The /media directory serves as a mount point for removable media devices such as CD-ROM's, digital cameras, and various usb-attached devices. Since /media is rather new in the Unix world, you could very well encounter systems running without this directory. Solaris 9 does not have it, Solaris 10 does. Most Linux distributions today mount all removable media in /media.

```
paul@debian5:~$ ls /media/
cdrom  cdrom0  usbdisk
```
The `/mnt` directory should be empty and should only be used for temporary mount points (according to the FHS).

Unix and Linux administrators used to create many directories here, like `/mnt/something/`. You likely will encounter many systems with more than one directory created and/or mounted inside `/mnt` to be used for various local and remote filesystems.

Applications and users should use `/tmp` to store temporary data when needed. Data stored in `/tmp` may use either disk space or RAM. Both of which are managed by the operating system. Never use `/tmp` to store data that is important or which you wish to archive.
9.7. in memory directories

/dev

Device files in /dev appear to be ordinary files, but are not actually located on the hard disk. The /dev directory is populated with files as the kernel is recognising hardware.

common physical devices

Common hardware such as hard disk devices are represented by device files in /dev. Below a screenshot of SATA device files on a laptop and then IDE attached drives on a desktop. (The detailed meaning of these devices will be discussed later.)

```
# SATA or SCSI or USB
# paul@laika:~$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3  /dev/sdb  /dev/sdb1  /dev/sdb2

# IDE or ATAPI
# paul@barry:~$ ls /dev/hd*
/dev/hda  /dev/hda1  /dev/hda2  /dev/hdb  /dev/hdb1  /dev/hdb2  /dev/hdc
```

Besides representing physical hardware, some device files are special. These special devices can be very useful.

/dev/tty and /dev/pts

For example, /dev/tty1 represents a terminal or console attached to the system. (Don't break your head on the exact terminology of 'terminal' or 'console', what we mean here is a command line interface.) When typing commands in a terminal that is part of a graphical interface like Gnome or KDE, then your terminal will be represented as /dev/pts/1 (1 can be another number).

/dev/null

On Linux you will find other special devices such as /dev/null which can be considered a black hole; it has unlimited storage, but nothing can be retrieved from it. Technically speaking, anything written to /dev/null will be discarded. /dev/null can be useful to discard unwanted output from commands. /dev/null is not a good location to store your backups ;-).
the Linux file tree

/proc conversation with the kernel

/proc is another special directory, appearing to be ordinary files, but not taking up disk space. It is actually a view of the kernel, or better, what the kernel manages, and is a means to interact with it directly. /proc is a proc filesystem.

```
paul@RHELv4u4:~$ mount -t proc
none on /proc type proc (rw)
```

When listing the /proc directory you will see many numbers (on any Unix) and some interesting files (on Linux)

```
mul@laika:~$ ls /proc
1       2339    4724    5418   6587   7201      cmdline   mounts
10175   2523    4729    5421   6596   7204      cpuinfo   mtrr
10211   2783    4741    5658   6599   7206      crypto    net
10239   2975    4873    5661   6638   7214      devices   pagetypeinfo
141     29775   4874    5665   6652   7216      diskstats partitions
15045   29792   4878    5927   6719   7218      dma        sched_debug
1519    2997    4879     6     6736   7223      driver    scsi
1548     3     4881    6032   6737   7224      execdomains self
1551    30228   4882    6033   6755   7227      fb        alabinfo
1554    3069     5     6145    6762   7260      filesystems stat
1557    31422   5073    6298    6774   7267      fs        swaps
1606    3149    5147    6414    6816   7275      ide        sys
180     31507   5203    6418    6991   7282      interrupts sysrq-trigger
181     3189    5206    6419    6993   7298      iomem     sysv ipc
182     3193    5228    6420    6996   7319      ioports timer_list
18898  3246    5272    6421    7157   7330      irq        timer_stats
19799  3248    5291    6422    7163   7345      kallsyms tty
19803  3253    5294    6423    7164   7353      kcore     uptime
19804  3372    5356    6424    7171   7525      key-users version
1987     4     5370    6425    7175   7529      kmsg     version_signature
1989    42     5379    6426    7188   9964      loadsavg vmcore
2      45     5380    6430    7189    252   locks    vmstat
20845  4542    5412    6450   7191   7930      asound meminfo vmware
221     46     5414    6551    7192   8930    buddyinfo misc   zoneinfo
2338   4704    5416    6568    7199   9930     bus     modules
```

Let’s investigate the file properties inside /proc. Looking at the date and time will display the current date and time showing the files are constantly updated (a view on the kernel).

```
paul@RHELv4u4:~$ date
Mon Jan 29 18:06:32 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 Jan 29 18:06 /proc/cpuinfo
paul@RHELv4u4:~$
paul@RHELv4u4:~$ ...time passes...
paul@RHELv4u4:~$
paul@RHELv4u4:~$ date
Mon Jan 29 18:10:00 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 Jan 29 18:10 /proc/cpuinfo
```
Most files in /proc are 0 bytes, yet they contain data--sometimes a lot of data. You can see this by executing `cat` on files like `/proc/cpuinfo`, which contains information about the CPU.

```
paul@RHELv4u4:~$ file /proc/cpuinfo
/proc/cpuinfo: empty
paul@RHELv4u4:~$ cat /proc/cpuinfo
processor : 0
vendor_id : AuthenticAMD
cpu family : 15
model : 43
model name : AMD Athlon(tm) 64 X2 Dual Core Processor 4600+
stepping : 1
cpu MHz : 2398.628
cache size : 512 KB
fdiv_bug : no
hlt_bug : no
f00f_bug : no
coma_bug : no
fpu : yes
fpu_exception : yes
cpuid level : 1
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge...
bogomips : 4803.54
```

*Just for fun, here is /proc/cpuinfo on a Sun Sunblade 1000...*

```
paul@pasha:~$ cat /proc/cpuinfo
cpu : TI UltraSparc III (Cheetah)
fpu : UltraSparc III integrated FPU
promlib : Version 3 Revision 2
prom : 4.2.2
type : sun4u
ncpus probed : 2
ncpus active : 2
Cpu0Bogo : 498.68
Cpu0ClkTck : 0000000002cb41780
Cpu1Bogo : 498.68
Cpu1ClkTck : 0000000002cb41780
MMU Type : Cheetah
State:
CPU0: online
CPU1: online
```

Most of the files in /proc are read only, some require root privileges, some files are writable, and many files in /proc/sys are writable. Let's discuss some of the files in /proc.
the Linux file tree

/proc/interrupts

On the x86 architecture, /proc/interrupts displays the interrupts.

paul@RHELv4u4:~$ cat /proc/interrupts

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 13876877</td>
<td></td>
</tr>
<tr>
<td>1:           15</td>
<td></td>
</tr>
<tr>
<td>8:           1</td>
<td></td>
</tr>
<tr>
<td>9:           0</td>
<td></td>
</tr>
<tr>
<td>12:          67</td>
<td></td>
</tr>
<tr>
<td>14:          128</td>
<td></td>
</tr>
<tr>
<td>15:        124320</td>
<td></td>
</tr>
<tr>
<td>169:        111993</td>
<td></td>
</tr>
<tr>
<td>177:        2428</td>
<td></td>
</tr>
<tr>
<td>NMI:          0</td>
<td></td>
</tr>
<tr>
<td>LOC: 13878037</td>
<td></td>
</tr>
<tr>
<td>ERR:          0</td>
<td></td>
</tr>
<tr>
<td>MIS:          0</td>
<td></td>
</tr>
</tbody>
</table>

On a machine with two CPU’s, the file looks like this.

paul@laika:~$ cat /proc/interrupts

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 860013</td>
<td>0</td>
</tr>
<tr>
<td>1: 4533</td>
<td>0</td>
</tr>
<tr>
<td>7: 0</td>
<td>0</td>
</tr>
<tr>
<td>8: 6588227</td>
<td>0</td>
</tr>
<tr>
<td>10: 2314</td>
<td>0</td>
</tr>
<tr>
<td>12: 133</td>
<td>0</td>
</tr>
<tr>
<td>14: 0</td>
<td>0</td>
</tr>
<tr>
<td>15: 72269</td>
<td>0</td>
</tr>
<tr>
<td>18: 1</td>
<td>0</td>
</tr>
<tr>
<td>19: 115036</td>
<td>0</td>
</tr>
<tr>
<td>20: 126871</td>
<td>0</td>
</tr>
<tr>
<td>21: 30204</td>
<td>0</td>
</tr>
<tr>
<td>22: 1334</td>
<td>0</td>
</tr>
<tr>
<td>24: 234739</td>
<td>0</td>
</tr>
<tr>
<td>NMI: 72</td>
<td>42</td>
</tr>
<tr>
<td>LOC: 860000</td>
<td>859994</td>
</tr>
<tr>
<td>ERR: 0</td>
<td></td>
</tr>
</tbody>
</table>

/proc/kcore

The physical memory is represented in /proc/kcore. Do not try to cat this file, instead use a debugger. The size of /proc/kcore is the same as your physical memory, plus four bytes.

paul@laika:~$ ls -lh /proc/kcore
-r-------- 1 root root 2.0G 2007-01-30 08:57 /proc/kcore
paul@laika:~$
/sys Linux 2.6 hot plugging

The /sys directory was created for the Linux 2.6 kernel. Since 2.6, Linux uses sysfs to support usb and IEEE 1394 (FireWire) hot plug devices. See the manual pages of udev(8) (the successor of devfs) and hotplug(8) for more info (or visit http://linux-hotplug.sourceforge.net/).

Basically the /sys directory contains kernel information about hardware.
9.8. /usr Unix System Resources

Although /usr is pronounced like user, remember that it stands for Unix System Resources. The /usr hierarchy should contain shareable, read only data. Some people choose to mount /usr as read only. This can be done from its own partition or from a read only NFS share.

/usr/bin

The /usr/bin directory contains a lot of commands.

paul@deb508:~$ ls /usr/bin | wc -l
1395

(On Solaris the /bin directory is a symbolic link to /usr/bin.)

/usr/include

The /usr/include directory contains general use include files for C.

paul@ubuntu10:~$ ls /usr/include/
aalib.h  expat_config.h  math.h  search.h  
af_vfs.h  expat_external.h  mcheck.h  semaphore.h  
alo.h  expat.h  memory.h  setjmp.h  
AL  fcntl.h  menu.h  sgtty.h  
aliases.h  features.h  mntent.h  shadow.h  
...

/usr/lib

The /usr/lib directory contains libraries that are not directly executed by users or scripts.

paul@deb508:~$ ls /usr/lib | head -7
4Suite
ao
apt
arj
aspell
avahi
bonobo

/usr/local

The /usr/local directory can be used by an administrator to install software locally.

paul@deb508:~$ ls /usr/local/
bin etc games include lib man sbin share src
paul@deb508:~$ du -sh /usr/local/
128K /usr/local/
The `/usr/share` directory contains architecture independent data. As you can see, this is a fairly large directory.

```
paul@deb508:~$ ls /usr/share/ | wc -l
263
paul@deb508:~$ du -sh /usr/share/
1.3G /usr/share/
```

This directory typically contains `/usr/share/man` for manual pages.

```
paul@deb508:~$ ls /usr/share/man
cs fr hu it.UTF-8 man2 man6 pl.ISO8859-2 sv
de fr.ISO8859-1 id ja man3 man7 pl.UTF-8 tr
es fr.UTF-8 it ko man4 man8 pt_BR zh_CN
fi gl it.ISO8859-1 man1 man5 pl ru zh_TW
```

And it contains `/usr/share/games` for all static game data (so no high-scores or play logs).

```
paul@ubuntu10:~$ ls /usr/share/games/
openTDD wesnoth
```

The `/usr/src` directory is the recommended location for kernel source files.

```
paul@deb508:~$ ls -l /usr/src/
total 12
```

```
drw-r-xr-x 4 root root 4096 2011-02-01 14:43 linux-headers-2.6.26-2-686
drw-r-xr-x 18 root root 4096 2011-02-01 14:43 linux-headers-2.6.26-2-common
drw-r-xr-x 3 root root 4096 2009-10-28 16:01 linux-kbuild-2.6.26
```
9.9. /var variable data

Files that are unpredictable in size, such as log, cache and spool files, should be located in /var.

/var/log

The /var/log directory serves as a central point to contain all log files.

```
[paul@RHEL4b ~]$ ls /var/log
acpid           cron.2    maillog.2   quagga           secure.4
amanda          cron.3    maillog.3   radius           spooler
anaconda.log    cron.4    maillog.4   rpmpkgs          spooler.1
anaconda.syslog cups      mailman   rpmpkgs.1        spooler.2
anaconda.xlog   dmesg     messages   rpmpkgs.2        spooler.3
audit           exim      messages.1 rpmpkgs.3         spooler.4
boot.log        gdm       messages.2 rpmpkgs.4        squid
```

/var/log/messages

A typical first file to check when troubleshooting on Red Hat (and derivatives) is the /var/log/messages file. By default this file will contain information on what just happened to the system. The file is called /var/log/syslog on Debian and Ubuntu.

```
[root@RHEL4b ~]# tail /var/log/messages
Jul 30 05:13:56 anacron: anacron startup succeeded
Jul 30 05:13:56 atd: atd startup succeeded
Jul 30 05:13:57 cups: cups-config-daemon: cups-config-daemon startup succeeded
Jul 30 05:13:58 haldaemon: haldaemon startup succeeded
Jul 30 05:14:00 fstab-sync[3560]: removed all generated mount points
Jul 30 05:14:01 fstab-sync[3628]: added mount point /media/cdrom for...
Jul 30 05:14:01 fstab-sync[3646]: added mount point /media/floppy for...
Jul 30 05:16:46 sshd(pam_unix)[3662]: session opened for user paul by...
Jul 30 06:06:37 su(pam_unix)[3904]: session opened for user root by paul
```

/var/cache

The /var/cache directory can contain cache data for several applications.

```
paul@ubu1010:~$ ls /var/cache/
apt       dictionaries-common     gdm       man       software-center
binfmts   flashplugin-installer  hald      pm-utils
```

---

66
The `/var/spool` directory typically contains spool directories for `mail` and `cron`, but also serves as a parent directory for other spool files (for example print spool files).

The `/var/lib` directory contains application state information.

Red Hat Enterprise Linux for example keeps files pertaining to `rpm` in `/var/lib/rpm`.

 `/var` also contains Process ID files in `/var/run` (soon to be replaced with `/run`) and temporary files that survive a reboot in `/var/tmp` and information about file locks in `/var/lock`. There will be more examples of `/var` usage further in this book.
9.10. practice: file system tree

1. Does the file `/bin/cat` exist? What about `/bin/dd` and `/bin/echo`. What is the type of these files?

2. What is the size of the Linux kernel file(s) (vmlinu*) in `/boot`?

3. Create a directory `~`/test. Then issue the following commands:
   ```
   cd ~test
   dd if=/dev/zero of=zeroes.txt count=1 bs=100
   od zeroes.txt
   ```
   *dd* will copy one times (count=1) a block of size 100 bytes (bs=100) from the file `/dev/zero` to `~/test/zeroes.txt`. Can you describe the functionality of `/dev/zero`?

4. Now issue the following command:
   ```
   dd if=/dev/random of=random.txt count=1 bs=100 ; od random.txt
   ```
   *dd* will copy one times (count=1) a block of size 100 bytes (bs=100) from the file `/dev/random` to `~/test/random.txt`. Can you describe the functionality of `/dev/random`?

5. Issue the following two commands, and look at the first character of each output line.
   ```
   ls -l /dev/sd* /dev/hd*
   ls -l /dev/tty* /dev/input/mou*
   ```
   The first `ls` will show block(b) devices, the second `ls` shows character(c) devices. Can you tell the difference between block and character devices?

6. Use `cat` to display `/etc/hosts` and `/etc/resolv.conf`. What is your idea about the purpose of these files?

7. Are there any files in `/etc/skel`? Check also for hidden files.

8. Display `/proc/cpuinfo`. On what architecture is your Linux running?

9. Display `/proc/interrupts`. What is the size of this file? Where is this file stored?

10. Can you enter the `/root` directory? Are there (hidden) files?

11. Are `ifconfig`, `fdisk`, `parted`, `shutdown` and `grub-install` present in `/sbin`? Why are these binaries in `/sbin` and not in `/bin`?

12. Is `/var/log` a file or a directory? What about `/var/spool`?

13. Open two command prompts (Ctrl-Shift-T in gnome-terminal) or terminals (Ctrl-Alt-F1, Ctrl-Alt-F2, ...) and issue the `who am i` in both. Then try to echo a word from one terminal to the other.
14. Read the man page of `random` and explain the difference between `/dev/random` and `/dev/urandom`. 
9.11. solution: file system tree

1. Does the file /bin/cat exist? What about /bin/dd and /bin/echo. What is the type of these files?

   ls /bin/cat ; file /bin/cat
   ls /bin/dd ; file /bin/dd
   ls /bin/echo ; file /bin/echo

2. What is the size of the Linux kernel file(s) (vmlinu*) in /boot?

   ls -lh /boot/vm*

3. Create a directory ~/test. Then issue the following commands:

   cd ~/test
   dd if=/dev/zero of=zeroes.txt count=1 bs=100
   od zeroes.txt

   **dd** will copy one times (count=1) a block of size 100 bytes (bs=100) from the file /dev/zero to ~/test/zeroes.txt. Can you describe the functionality of /dev/zero?

   /dev/zero is a Linux special device. It can be considered a source of zeroes. You cannot send something to /dev/zero, but you can read zeroes from it.

4. Now issue the following command:

   dd if=/dev/random of=random.txt count=1 bs=100 ; od random.txt

   **dd** will copy one times (count=1) a block of size 100 bytes (bs=100) from the file /dev/random to ~/test/random.txt. Can you describe the functionality of /dev/random?

   /dev/random acts as a **random number generator** on your Linux machine.

5. Issue the following two commands, and look at the first character of each output line.

   ls -l /dev/sd* /dev/hd*
   ls -l /dev/tty* /dev/input/mou*

   The first ls will show block(b) devices, the second ls shows character(c) devices. Can you tell the difference between block and character devices?

   Block devices are always written to (or read from) in blocks. For hard disks, blocks of 512 bytes are common. Character devices act as a stream of characters (or bytes). Mouse and keyboard are typical character devices.

6. Use cat to display /etc/hosts and /etc/resolv.conf. What is your idea about the purpose of these files?
the Linux file tree

/etc/hosts contains hostnames with their ip address
/etc/resolv.conf should contain the ip address of a DNS name server.

7. Are there any files in /etc/skel/ ? Check also for hidden files.
   Issue "ls -al /etc/skel/". Yes, there should be hidden files there.

8. Display /proc/cpuinfo. On what architecture is your Linux running ?
   The file should contain at least one line with Intel or other cpu.

9. Display /proc/interrupts. What is the size of this file ? Where is this file stored ?
   The size is zero, yet the file contains data. It is not stored anywhere because /proc is a virtual file system that allows you to talk with the kernel. (If you answered "stored in RAM-memory, that is also correct...).

10. Can you enter the /root directory ? Are there (hidden) files ?
    Try "cd /root". Yes there are (hidden) files there.

11. Are ifconfig, fdisk, parted, shutdown and grub-install present in /sbin ? Why are these binaries in /sbin and not in /bin ?
    Because those files are only meant for system administrators.

12. Is /var/log a file or a directory ? What about /var/spool ?
    Both are directories.

13. Open two command prompts (Ctrl-Shift-T in gnome-terminal) or terminals (Ctrl-Alt-F1, Ctrl-Alt-F2, ...) and issue the who am i in both. Then try to echo a word from one terminal to the other.
    tty-terminal: echo Hello > /dev/tty1
    pts-terminal: echo Hello > /dev/pts/1

14. Read the man page of random and explain the difference between /dev/random and /dev/urandom.
    man 4 random
Part III. shell expansion
Chapter 10. commands and arguments

Table of Contents

10.1. echo .............................................................................................................. 74
10.2. arguments ..................................................................................................... 74
10.3. commands .................................................................................................... 76
10.4. aliases ........................................................................................................... 77
10.5. displaying shell expansion ........................................................................... 78
10.6. practice: commands and arguments ............................................................. 79
10.7. solution: commands and arguments ............................................................. 81

This chapter introduces you to shell expansion by taking a close look at commands and arguments. Knowing shell expansion is important because many commands on your Linux system are processed and most likely changed by the shell before they are executed.

The command line interface or shell used on most Linux systems is called bash, which stands for Bourne again shell. The bash shell incorporates features from sh (the original Bourne shell), csh (the C shell), and ksh (the Korn shell).
10.1. echo

This chapter frequently uses the `echo` command to demonstrate shell features. The `echo` command is very simple: it echoes the input that it receives.

```
paul@laika:~$ echo Burtonville
Burtonville
paul@laika:~$ echo Smurfs are blue
Smurfs are blue
```

10.2. arguments

One of the primary features of a shell is to perform a **command line scan**. When you enter a command at the shell's command prompt and press the enter key, then the shell will start scanning that line, cutting it up in **arguments**. While scanning the line, the shell may make many changes to the **arguments** you typed. This process is called **shell expansion**. When the shell has finished scanning and modifying that line, then it will be executed.

**white space removal**

Parts that are separated by one or more consecutive **white spaces** (or tabs) are considered separate **arguments**, any white space is removed. The first **argument** is the command to be executed, the other **arguments** are given to the command. The shell effectively cuts your command into one or more arguments.

This explains why the following four different command lines are the same after **shell expansion**.

```
[paul@RHELv4u3 ~]$ echo Hello World
Hello World
[paul@RHELv4u3 ~]$ echo Hello   World
Hello World
[paul@RHELv4u3 ~]$ echo   Hello   World
Hello World
[paul@RHELv4u3 ~]$    echo      Hello      World
Hello World
```

The **echo** command will display each argument it receives from the shell. The **echo** command will also add a new white space between the arguments it received.

**single quotes**

You can prevent the removal of white spaces by quoting the spaces. The contents of the quoted string are considered as one argument. In the screenshot below the **echo** receives only one **argument**.

```
[paul@RHEL4b ~]$ echo 'A line with single quotes'
A line with single quotes
[paul@RHEL4b ~]$
```
double quotes

You can also prevent the removal of white spaces by double quoting the spaces. Same as above, `echo` only receives one **argument**.

```
[paul@RHEL4b ~]$ echo "A line with double quotes"
A line with double quotes
[paul@RHEL4b ~]$
```

Later in this book, when discussing **variables** we will see important differences between single and double quotes.

echo and quotes

Quoted lines can include special escaped characters recognised by the `echo` command (when using `echo -e`). The screenshot below shows how to use `\n` for a newline and `\t` for a tab (usually eight white spaces).

```
[paul@RHEL4b ~]$ echo -e "A line with \na newline"
A line with a newline
[paul@RHEL4b ~]$ echo -e 'A line with \na newline'
A line with a newline
[paul@RHEL4b ~]$ echo -e "A line with \t a tab"
A line with a tab
[paul@RHEL4b ~]$ echo -e 'A line with \t a tab'
A line with a tab
[paul@RHEL4b ~]$
```

The `echo` command can generate more than white spaces, tabs and newlines. Look in the man page for a list of options.
10.3. commands

external or builtin commands?

Not all commands are external to the shell, some are builtin. **External commands** are programs that have their own binary and reside somewhere in the file system. Many external commands are located in /bin or /sbin. **Builtin commands** are an integral part of the shell program itself.

**type**

To find out whether a command given to the shell will be executed as an **external command** or as a **builtin command**, use the **type** command.

```
paul@laika:~$ type cd
cd is a shell builtin
paul@laika:~$ type cat
cat is /bin/cat
```

As you can see, the **cd** command is **builtin** and the **cat** command is **external**.

You can also use this command to show you whether the command is **aliased** or not.

```
paul@laika:~$ type ls
ls is aliased to `ls --color=auto'
```

**running external commands**

Some commands have both builtin and external versions. When one of these commands is executed, the builtin version takes priority. To run the external version, you must enter the full path to the command.

```
paul@laika:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
paul@laika:~$ /bin/echo Running the external echo command...
```

**which**

The **which** command will search for binaries in the **$PATH** environment variable (variables will be explained later). In the screenshot below, it is determined that **cd** is **builtin**, and **ls, cp, rm, mv, mkdir, pwd**, and **which** are external commands.

```
[root@RHEL4b ~]# which cp ls cd mkdir pwd
/bin/cp
/bin/ls
/usr/bin/which: no cd in (/usr/kerberos/sbin:/usr/kerberos/bin:...
/bin/mkdir
/bin/pwd
```
10.4. aliases

create an alias

The shell allows you to create **aliases**.Aliases are often used to create an easier to remember name for an existing command or to easily supply parameters.

```bash
[paul@RHELv4u3 ~]$ cat count.txt
one
two
three
[paul@RHELv4u3 ~]$ alias dog=tac
[paul@RHELv4u3 ~]$ dog count.txt
three
two
one
```

abbreviate commands

An **alias** can also be useful to abbreviate an existing command.

```bash
paul@laika:~$ alias ll='ls -lh --color=auto'
paul@laika:~$ alias c='clear'
paul@laika:~$
```

default options

Aliases can be used to supply commands with default options. The example below shows how to set the `-i` option default when typing `rm`.

```bash
[paul@RHELv4u3 ~]$ rm -i winter.txt
rm: remove regular file `winter.txt'? no
[paul@RHELv4u3 ~]$ rm winter.txt
ls: winter.txt: No such file or directory
[paul@RHELv4u3 ~]$ ls winter.txt
[paul@RHELv4u3 ~]$ touch winter.txt
[paul@RHELv4u3 ~]$ alias rm='rm -i'
[paul@RHELv4u3 ~]$ rm winter.txt
rm: remove regular empty file `winter.txt'? no
[paul@RHELv4u3 ~]$ rm winter.txt
```

Some distributions enable default aliases to protect users from accidentally erasing files (`'rm -i', 'mv -i', 'cp -i'`)

viewing aliases

You can provide one or more aliases as arguments to the `alias` command to get their definitions. Providing no arguments gives a complete list of current aliases.

```bash
paul@laika:~$ alias c ll
alias c='clear'
alias ll='ls -lh --color=auto'
```
unalias

You can undo an alias with the unalias command.

```
[paul@RHEL4b ~]$ which rm
/bin/rm
[paul@RHEL4b ~]$ alias rm='rm -i'
[paul@RHEL4b ~]$ which rm
alias rm='rm -i'
 /bin/rm
[paul@RHEL4b ~]$ unalias rm
[paul@RHEL4b ~]$ which rm
 /bin/rm
[paul@RHEL4b ~]$
```

10.5. displaying shell expansion

You can display shell expansion with set -x, and stop displaying it with set +x. You might want to use this further on in this course, or when in doubt about exactly what the shell is doing with your command.

```
[paul@RHELv4u3 ~]$ set -x
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo $USER
+ echo paul
paul
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo \$USER
+ echo '\$USER'
\$USER
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ set +x
+ set +x
[paul@RHELv4u3 ~]$ echo $USER
paul
```
10.6. practice: commands and arguments

1. How many **arguments** are in this line (not counting the command itself).

```
touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt"
```

2. Is **tac** a shell builtin command?

3. Is there an existing alias for **rm**?

4. Read the man page of **rm**, make sure you understand the **-i** option of rm. Create and remove a file to test the **-i** option.

5. Execute: **alias rm='rm -i'**. Test your alias with a test file. Does this work as expected?


7a. Create an alias called 'city' that echoes your hometown.

7b. Use your alias to test that it works.

8. Execute **set -x** to display shell expansion for every command.

9. Test the functionality of **set -x** by executing your **city** and **rm** aliases.

10 Execute **set +x** to stop displaying shell expansion.

11. Remove your city alias.

12. What is the location of the **cat** and the **passwd** commands?

13. Explain the difference between the following commands:

```
  echo
  /bin/echo
```

14. Explain the difference between the following commands:

```
  echo Hello
  echo -n Hello
```

15. Display **A B C** with two spaces between B and C.

(optional)16. Complete the following command (do not use spaces) to display exactly the following output:

```
  4+4    =8
  10+14  =24
```

18. Use **echo** to display the following exactly:

```
  ??\/
```
19. Use one `echo` command to display three words on three lines.
10.7. solution: commands and arguments

1. How many arguments are in this line (not counting the command itself).
   
touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt"

   answer: three

2. Is tac a shell builtin command?
   
type tac

3. Is there an existing alias for rm?
   
alias rm

4. Read the man page of rm, make sure you understand the -i option of rm. Create and remove a file to test the -i option.
   
man rm

touch testfile

rm -i testfile

5. Execute: alias rm='rm -i'. Test your alias with a test file. Does this work as expected?
   
touch testfile

rm testfile (should ask for confirmation)

   
alias

7a. Create an alias called 'city' that echoes your hometown.
   
alias city='echo Antwerp'

7b. Use your alias to test that it works.
   
city (it should display Antwerp)

8. Execute set -x to display shell expansion for every command.
   
set -x

9. Test the functionality of set -x by executing your city and rm aliases.
   
shell should display the resolved aliases and then execute the command:

paul@deb503:~$ set -x
paul@deb503:~$ city
+ echo antwerp
antwerp

10. Execute set +x to stop displaying shell expansion.

set +x

11. Remove your city alias.
12. What is the location of the **cat** and the **passwd** commands?

which cat (probably /bin/cat)
which passwd (probably /usr/bin/passwd)

13. Explain the difference between the following commands:

```
/ /n
```

14. Explain the difference between the following commands:

```
/ /n
```

(The -n option might not work in the Korn shell.)

15. Display **A B C** with two spaces between B and C.

```
A B C
```

16. Complete the following command (do not use spaces) to display exactly the following output:

```
4+4 =8
10+14 =24
```

The solution is to use tabs with \\.  

18. Use **echo** to display the following exactly:

```
??\\
```

19. Use one **echo** command to display three words on three lines.

```
/ /n/ /n
```

# commands and arguments

unalias city

12. What is the location of the **cat** and the **passwd** commands?

which cat (probably /bin/cat)
which passwd (probably /usr/bin/passwd)

13. Explain the difference between the following commands:

```
/ /n
```

14. Explain the difference between the following commands:

```
/ /n
```

(The -n option might not work in the Korn shell.)

15. Display **A B C** with two spaces between B and C.

```
A B C
```

16. Complete the following command (do not use spaces) to display exactly the following output:

```
4+4 =8
10+14 =24
```

The solution is to use tabs with \\.  

18. Use **echo** to display the following exactly:

```
??\\
```

19. Use one **echo** command to display three words on three lines.

```
/ /n/ /n
```
Chapter 11. control operators

Table of Contents

11.1. ; semicolon ................................................................. 84
11.2. & ampersand ............................................................... 84
11.3. $? dollar question mark ............................................... 84
11.4. && double ampersand .................................................. 85
11.5. || double vertical bar .................................................... 85
11.6. combining && and || ...................................................... 85
11.7. # pound sign ............................................................... 86
11.8. \ escaping special characters ......................................... 86
11.9. practice: control operators ............................................ 87
11.10. solution: control operators .......................................... 88

In this chapter we put more than one command on the command line using control operators. We also briefly discuss related parameters ($?) and similar special characters(&).
11.1. ; semicolon

You can put two or more commands on the same line separated by a semicolon ;. The shell will scan the line until it reaches the semicolon. All the arguments before this semicolon will be considered a separate command from all the arguments after the semicolon. Both series will be executed sequentially with the shell waiting for each command to finish before starting the next one.

[paul@RHELv4u3 ~]$ echo Hello
Hello
[paul@RHELv4u3 ~]$ echo World
World
[paul@RHELv4u3 ~]$ echo Hello ; echo World
Hello
World
[paul@RHELv4u3 ~]$

11.2. & ampersand

When a line ends with an ampersand &, the shell will not wait for the command to finish. You will get your shell prompt back, and the command is executed in background. You will get a message when this command has finished executing in background.

[paul@RHELv4u3 ~]$ sleep 20 &
[1] 7925
[paul@RHELv4u3 ~]$ ...
...wait 20 seconds...
[paul@RHELv4u3 ~]$
[1]+ Done sleep 20

The technical explanation of what happens in this case is explained in the chapter about processes.

11.3. $? dollar question mark

The exit code of the previous command is stored in the shell variable $?.

paul@debian5:~/test$ touch file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
rm: cannot remove `file1': No such file or directory
paul@debian5:~/test$ echo $?
1
paul@debian5:~/test$
11.4. && double ampersand

The shell will interpret && as a logical AND. When using && the second command is executed only if the first one succeeds (returns a zero exit status).

```
paul@barry:~$ echo first && echo second
first
second
paul@barry:~$ zecho first && echo second
-bash: zecho: command not found
```

Another example of the same logical AND principle. This example starts with a working cd followed by ls, then a non-working cd which is not followed by ls.

```
[paul@RHELv4u3 ~]$ cd gen && ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[paul@RHELv4u3 gen]$ cd gen && ls
-bash: cd: gen: No such file or directory
```

11.5. || double vertical bar

The || represents a logical OR. The second command is executed only when the first command fails (returns a non-zero exit status).

```
paul@barry:~$ echo first || echo second ; echo third
first
third
paul@barry:~$ zecho first || echo second ; echo third
-bash: zecho: command not found
second
third
paul@barry:~$
```

Another example of the same logical OR principle.

```
[paul@RHELv4u3 ~]$ cd gen || ls
[paul@RHELv4u3 gen]$ cd gen || ls
-bash: cd: gen: No such file or directory
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
```

11.6. combining && and ||

You can use this logical AND and logical OR to write an if-then-else structure on the command line. This example uses echo to display whether the rm command was successful.

```
paul@laika:~/test$ rm file1 && echo It worked! || echo It failed!
It worked!
paul@laika:~/test$ rm file1 && echo It worked! || echo It failed!
rm: cannot remove 'file1': No such file or directory
It failed!
paul@laika:~/test$
```
11.7. # pound sign

Everything written after a **pound sign (#)** is ignored by the shell. This is useful to write a **shell comment**, but has no influence on the command execution or shell expansion.

```
paul@debian4:~$ mkdir test    # we create a directory
paul@debian4:~$ cd test       ### we enter the directory
paul@debian4:~/test$ ls        # is it empty ?
paul@debian4:~/test$
```

11.8. \ escaping special characters

The backslash \ character enables the use of control characters, but without the shell interpreting it, this is called **escaping** characters.

```
[paul@RHELv4u3 ~]$ echo hello \; world
hello ; world
[paul@RHELv4u3 ~]$ echo hello\ \ \ world
hello  world
[paul@RHELv4u3 ~]$ echo escaping \\ \#/ \&\ "\ \\
escaping \ # & " \\
[paul@RHELv4u3 ~]$ echo escaping \\?*"'
escaping \?*"'
```

**end of line backslash**

Lines ending in a backslash are continued on the next line. The shell does not interpret the newline character and will wait on shell expansion and execution of the command line until a newline without backslash is encountered.

```
[paul@RHEL4b ~]$ echo This command line \\
> is split in three \\
> parts
This command line is split in three parts
[paul@RHEL4b ~]$
11.9. practice: control operators

0. Each question can be answered by one command line!

1. When you type `passwd`, which file is executed?

2. What kind of file is that?

3. Execute the `pwd` command twice. (remember 0.)

4. Execute `ls` after `cd /etc`, but only if `cd /etc` did not error.

5. Execute `cd /etc` after `cd etc`, but only if `cd etc` fails.

6. Echo `it worked` when `touch test42` works, and echo `it failed` when the `touch` failed. All on one command line as a normal user (not root). Test this line in your home directory and in `/bin/`.

7. Execute `sleep 6`, what is this command doing?

8. Execute `sleep 200` in background (do not wait for it to finish).

9. Write a command line that executes `rm file55`. Your command line should print 'success' if file55 is removed, and print 'failed' if there was a problem.

(optional) 10. Use echo to display "Hello World with strange' characters \* [ ] ~ \" (including all quotes)
11.10. solution: control operators

0. Each question can be answered by one command line!

1. When you type **passwd**, which file is executed?
   ```
   which passwd
   ```

2. What kind of file is that?
   ```
   file /usr/bin/passwd
   ```

3. Execute the **pwd** command twice. (remember 0.)
   ```
   pwd ; pwd
   ```

4. Execute **ls** after **cd /etc**, but only if **cd /etc** did not error.
   ```
   cd /etc && ls
   ```

5. Execute **cd /etc** after **cd etc**, but only if **cd etc** fails.
   ```
   cd etc || cd /etc
   ```

6. Echo it **worked** when **touch test42** works, and echo it **failed** when the **touch** failed. All on one command line as a normal user (not root). Test this line in your home directory and in **/bin/**.
   ```
   paul@deb503:~$ cd ; touch test42 && echo it worked || echo it failed
   paul@deb503:~$ cd /bin; touch test42 && echo it worked || echo it failed
   touch: cannot touch `test42': Permission denied
   it failed
   ```

7. Execute **sleep 6**, what is this command doing?
   ```
   pausing for six seconds
   ```

8. Execute **sleep 200** in background (do not wait for it to finish).
   ```
   sleep 200 &
   ```

9. Write a command line that executes **rm file55**. Your command line should print 'success' if file55 is removed, and print 'failed' if there was a problem.
   ```
   rm file55 && echo success || echo failed
   ```

   (optional)10. Use echo to display "Hello World with strange' characters \* [ } ~ \." (including all quotes)
   ```
   echo ""Hello World with strange' characters \* [ } ~ \." or
   ```
   ```
   echo """Hello World with strange' characters \* [ } ~ \.""
Chapter 12. variables

Table of Contents

12.1. about variables ................................................................. 90
12.2. quotes .............................................................................. 92
12.3. set ..................................................................................... 92
12.4. unset ............................................................................... 92
12.5. env .................................................................................. 93
12.6. export ............................................................................. 93
12.7. delineate variables ......................................................... 94
12.8. unbound variables .......................................................... 94
12.9. shell options ................................................................. 95
12.10. shell embedding ............................................................ 96
12.11. practice: shell variables ............................................... 97
12.12. solution: shell variables .............................................. 98

In this chapter we learn to manage environment variables in the shell. These variables are often read by applications.

We also take a brief look at child shells, embedded shells and shell options.
12.1. about variables

$ dollar sign

Another important character interpreted by the shell is the dollar sign $. The shell will look for an environment variable named like the string following the dollar sign and replace it with the value of the variable (or with nothing if the variable does not exist).

These are some examples using $HOSTNAME, $USER, $UID, $SHELL, and $HOME.

[paul@RHELv4u3 ~]$ echo This is the $SHELL shell
This is the /bin/bash shell
[paul@RHELv4u3 ~]$ echo This is $SHELL on computer $HOSTNAME
This is /bin/bash on computer RHELv4u3.localdomain
[paul@RHELv4u3 ~]$ echo The userid of $USER is $UID
The userid of paul is 500
[paul@RHELv4u3 ~]$ echo My homedir is $HOME
My homedir is /home/paul

case sensitive

This example shows that shell variables are case sensitive!

[paul@RHELv4u3 ~]$ echo Hello $USER
Hello paul
[paul@RHELv4u3 ~]$ echo Hello $user
Hello

$PS1

The $PS1 variable determines your shell prompt. You can use backslash escaped special characters like \u for the username or \w for the working directory. The bash manual has a complete reference.

In this example we change the value of $PS1 a couple of times.

paul@deb503:~$ PS1=prompt
prompt
promptPS1='prompt '
prompt
prompt PS1='>'
>
> PS1='\u@\h$'
paul@deb503$ paul@deb503$ PS1='\u@\h:~'
paul@deb503:~$

To avoid unrecoverable mistakes, you can set normal user prompts to green and the root prompt to red. This picture shows (one way) to do this.
The \$PATH variable is determines where the shell is looking for commands to execute (unless the command is builtin or aliased). This variable contains a list of directories, separated by colons.

The shell will not look in the current directory for commands to execute! (Looking for executables in the current directory provided an easy way to hack PC-DOS computers). If you want the shell to look in the current directory, then add a . at the end of your \$PATH.

Your path might be different when using su instead of su - because the latter will take on the environment of the target user. The root user typically has /sbin directories added to the \$PATH variable.

This example creates the variable \$MyVar and sets its value. It then uses echo to verify the value.

```
[paul@RHELv4u3 gen]$ MyVar=555
[paul@RHELv4u3 gen]$ echo $MyVar
555
[paul@RHELv4u3 gen]$
```
12.2. quotes

Notice that double quotes still allow the parsing of variables, whereas single quotes prevent this.

```
[paul@RHELv4u3 ~]$ MyVar=555
[paul@RHELv4u3 ~]$ echo $MyVar
555
[paul@RHELv4u3 ~]$ echo "$MyVar"
555
[paul@RHELv4u3 ~]$ echo '$MyVar'
$MyVar
```

The bash shell will replace variables with their value in double quoted lines, but not in single quoted lines.

```
paul@laika:~$ city=Burtonville
paul@laika:~$ echo "We are in $city today."
We are in Burtonville today.
paul@laika:~$ echo 'We are in $city today.'
We are in $city today.
```

12.3. set

You can use the `set` command to display a list of environment variables. On Ubuntu and Debian systems, the `set` command will also list shell functions after the shell variables. Use `set | more` to see the variables then.

12.4. unset

Use the `unset` command to remove a variable from your shell environment.

```
[paul@RHEL4b ~]$ MyVar=8472
[paul@RHEL4b ~]$ echo $MyVar
8472
[paul@RHEL4b ~]$ unset MyVar
[paul@RHEL4b ~]$ echo $MyVar
```

$
12.5. env

The `env` command without options will display a list of **exported variables**. The difference with `set` with options is that `set` lists all variables, including those not exported to child shells.

But `env` can also be used to start a clean shell (a shell without any inherited environment). The `env -i` command clears the environment for the subshell.

Notice in this screenshot that `bash` will set the `$SHELL` variable on startup.

```bash
[paul@RHEL4b ~]$ bash -c 'echo $SHELL $HOME $USER'
/bin/bash /home/paul paul
[paul@RHEL4b ~]$ env -i bash -c 'echo $SHELL $HOME $USER'
/bin/bash
[paul@RHEL4b ~]$
```

You can use the `env` command to set the `$LANG`, or any other, variable for just one instance of `bash` with one command. The example below uses this to show the influence of the `$LANG` variable on file globbing (see the chapter on file globbing).

```bash
[paul@RHEL4b test]$ env LANG=C bash -c 'ls File[a-z]'
Filea  Fileb
[paul@RHEL4b test]$ env LANG=en_US.UTF-8 bash -c 'ls File[a-z]'
Filea  FileA  Fileb  FileB
[paul@RHEL4b test]$
```

12.6. export

You can export shell variables to other shells with the `export` command. This will export the variable to child shells.

```bash
[paul@RHEL4b ~]$ var3=three
[paul@RHEL4b ~]$ var4=four
[paul@RHEL4b ~]$ export var4
[paul@RHEL4b ~]$ echo $var3 $var4
three four
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
four five
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
three four
[paul@RHEL4b ~]$
```

But it will not export to the parent shell (previous screenshot continued).

```bash
[paul@RHEL4b ~]$ export var5=five
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
four five
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
three four
[paul@RHEL4b ~]$
```
variables

12.7. delineate variables

Until now, we have seen that bash interprets a variable starting from a dollar sign, continuing until the first occurrence of a non-alphanumeric character that is not an underscore. In some situations, this can be a problem. This issue can be resolved with curly braces like in this example.

```
[paul@RHEL4b ~]$ prefix=Super
[paul@RHEL4b ~]$ echo Hello $prefixman and $prefixgirl
Hello and
[paul@RHEL4b ~]$ echo Hello ${prefix}man and ${prefix}girl
Hello Superman and Supergirl
[paul@RHEL4b ~]$
```

12.8. unbound variables

The example below tries to display the value of the $MyVar variable, but it fails because the variable does not exist. By default the shell will display nothing when a variable is unbound (does not exist).

```
[paul@RHELv4u3 gen]$ echo $MyVar
[paul@RHELv4u3 gen]$
```

There is, however, the nounset shell option that you can use to generate an error when a variable does not exist.

```
paul@laika:~$ set -u
paul@laika:~$ echo $Myvar
bash: Myvar: unbound variable
paul@laika:~$ set +u
paul@laika:~$ echo $Myvar
paul@laika:~$
```

In the bash shell `set -u` is identical to `set -o nounset` and likewise `set +u` is identical to `set +o nounset`. 
12.9. shell options

Both set and unset are builtin shell commands. They can be used to set options of the bash shell itself. The next example will clarify this. By default, the shell will treat unset variables as a variable having no value. By setting the -u option, the shell will treat any reference to unset variables as an error. See the man page of bash for more information.

[paul@RHEL4b ~]$ echo $var123
[paul@RHEL4b ~]$ set -u
[paul@RHEL4b ~]$ echo $var123
-bash: var123: unbound variable
[paul@RHEL4b ~]$ set +u
[paul@RHEL4b ~]$ echo $var123

[paul@RHEL4b ~]$

To list all the set options for your shell, use echo -. The noclobber (or -C) option will be explained later in this book (in the I/O redirection chapter).

[paul@RHEL4b ~]$ echo -
himBH
[paul@RHEL4b ~]$ set -C ; set -u
[paul@RHEL4b ~]$ echo -
himuBCH
[paul@RHEL4b ~]$ set +C ; set +u
[paul@RHEL4b ~]$ echo -
himBH
[paul@RHEL4b ~]$

When typing set without options, you get a list of all variables without function when the shell is on posix mode. You can set bash in posix mode typing set -o posix.
12.10. shell embedding

Shells can be embedded on the command line, or in other words, the command line scan can spawn new processes containing a fork of the current shell. You can use variables to prove that new shells are created. In the screenshot below, the variable $var1 only exists in the (temporary) sub shell.

[paul@RHELv4u3 gen]$ echo $var1
[paul@RHELv4u3 gen]$ echo $(var1=5;echo $var1)
 5
[paul@RHELv4u3 gen]$ echo $var1
[paul@RHELv4u3 gen]$

You can embed a shell in an embedded shell, this is called nested embedding of shells.

This screenshot shows an embedded shell inside an embedded shell.

paul@deb503:~$ A=shell
paul@deb503:~$ echo $C$B$A $(B=sub;echo $C$B$A; echo $(A=sub;echo $C$B$A))
shell subshell subsub

backticks

Single embedding can be useful to avoid changing your current directory. The screenshot below uses backticks instead of dollar-bracket to embed.

[paul@RHELv4u3 ~]$ echo `cd /etc; ls -d * | grep pass`
passwd passwd- passwd.OLD
[paul@RHELv4u3 ~]$

You can only use the $() notation to nest embedded shells, backticks cannot do this.

backticks or single quotes

Placing the embedding between backticks uses one character less than the dollar and parenthesis combo. Be careful however, backticks are often confused with single quotes. The technical difference between ' and ` is significant!

[paul@RHELv4u3 gen]$ echo `var1=5;echo $var1`
 5
[paul@RHELv4u3 gen]$ echo `var1=5;echo $var1`
var1=5;echo $var1
[paul@RHELv4u3 gen]$
12.11. practice: shell variables

1. Use echo to display Hello followed by your username. (use a bash variable!)

2. Create a variable \texttt{answer} with a value of 42.

3. Copy the value of $\text{LANG}$ to $\text{MyLANG}$.

4. List all current shell variables.

5. List all exported shell variables.

6. Do the \texttt{env} and \texttt{set} commands display your variable?

6. Destroy your \texttt{answer} variable.

7. Find the list of shell options in the man page of \texttt{bash}. What is the difference between \texttt{set -u} and \texttt{set -o nounset}?

8. Create two variables, and \texttt{export} one of them.

9. Display the exported variable in an interactive child shell.

10. Create a variable, give it the value 'Dumb', create another variable with value 'do'. Use \texttt{echo} and the two variables to echo Dumbledore.

11. Activate \texttt{nounset} in your shell. Test that it shows an error message when using non-existing variables.

12. Deactivate nounset.

13. Find the list of backslash escaped characters in the manual of bash. Add the time to your \texttt{PS1} prompt.

14. Execute \texttt{cd /var} and \texttt{ls} in an embedded shell.

15. Create the variable \texttt{embvar} in an embedded shell and echo it. Does the variable exist in your current shell now?

16. Explain what "set -x" does. Can this be useful?

(Optional)17. Given the following screenshot, add exactly four characters to that command line so that the total output is FirstMiddleLast.

```
[paul@RHEL4b ~]$ echo First; echo Middle; echo Last
```

18. Display a \texttt{long listing} (\texttt{ls -l}) of the \texttt{passwd} command using the \texttt{which} command inside back ticks.
12.12. solution: shell variables

1. Use echo to display Hello followed by your username. (use a bash variable!)

```bash
echo Hello $USER
```

2. Create a variable `answer` with a value of 42.

```bash
answer=42
```

3. Copy the value of `$LANG` to `$MyLANG`.

```bash
MyLANG=$LANG
```

4. List all current shell variables.

```bash
set
set|more on Ubuntu/Debian
```

5. List all exported shell variables.

```bash
env
```

6. Do the `env` and `set` commands display your variable ?

```bash
env | more
set | more
```

6. Destroy your `answer` variable.

```bash
unset answer
```

7. Find the list of shell options in the man page of `bash`. What is the difference between `set -u` and `set -o nounset`?

read the manual of bash (man bash), search for nounset -- both mean the same thing.

8. Create two variables, and `export` one of them.

```bash
var1=1; export var2=2
```

9. Display the exported variable in an interactive child shell.

```bash
bash
echo $var2
```

10. Create a variable, give it the value 'Dumb', create another variable with value 'do'. Use `echo` and the two variables to echo Dumbledore.

```bash
varx=Dumb; vary=do

echo $varx'le${vary}'re
solution by Yves from Dexia : echo $varx'le'$vary're'
solution by Erwin from Telenet : echo "$varx"'le"$vary"re
```

11. Activate `nounset` in your shell. Test that it shows an error message when using non-existing variables.
Both these lines have the same effect.

12. Deactivate nounset.

```bash
set +u
OR
set +o nounset
```

13. Find the list of backslash escaped characters in the manual of bash. Add the time to your $PS1$ prompt.

```bash
PS1='\t \u@\h \W$ '
```

14. Execute `cd /var` and `ls` in an embedded shell.

```bash
echo $(cd /var ; ls)
```

The `echo` command is only needed to show the result of the `ls` command. Omitting will result in the shell trying to execute the first file as a command.

15. Create the variable `embvar` in an embedded shell and echo it. Does the variable exist in your current shell now?

```bash
$(embvar=emb;echo $embvar) ; echo $embvar (the last echo fails).
```

```
$embvar does not exist in your current shell
```

16. Explain what "set -x" does. Can this be useful?

It displays shell expansion for troubleshooting your command.

(optional)17. Given the following screenshot, add exactly four characters to that command line so that the total output is FirstMiddleLast.

```bash
[paul@RHEL4b ~]$ echo  First; echo  Middle; echo  Last
echo -n First; echo -n Middle; echo Last
```

18. Display a long listing (ls -l) of the `passwd` command using the `which` command inside back tics.

```bash
ls -l `which passwd`
```
Chapter 13. shell history

Table of Contents

13.1. repeating the last command ................................................................. 101
13.2. repeating other commands ................................................................. 101
13.3. history ................................................................................................. 101
13.4. \n ........................................................................................................... 101
13.5. Ctrl-r ................................................................................................... 102
13.6. $HISTSIZE .......................................................................................... 102
13.7. $HISTFILE ......................................................................................... 102
13.8. $HISTFILESIZE ............................................................................... 102
13.9. (optional)regular expressions ............................................................. 103
13.10. (optional)repeating commands in ksh ............................................... 103
13.11. practice: shell history ...................................................................... 104
13.12. solution: shell history ...................................................................... 105

The shell makes it easy for us to repeat commands, this chapter explains how.
13.1. repeating the last command

To repeat the last command in bash, type `!!`. This is pronounced as *bang bang*.

```
paul@debian5:~/test42$ echo this will be repeated > file42.txt
paul@debian5:~/test42$ !!
echo this will be repeated > file42.txt
```

13.2. repeating other commands

You can repeat other commands using one *bang* followed by one or more characters. The shell will repeat the last command that started with those characters.

```
paul@debian5:~/test42$ touch file42
paul@debian5:~/test42$ cat file42
paul@debian5:~/test42$ !to
   touch file42
```

13.3. history

To see older commands, use `history` to display the shell command history (or use `history n` to see the last n commands).

```
paul@debian5:~/test$ history 10
38  mkdir test
39  cd test
40  touch file1
41  echo hello > file2
42  echo It is very cold today > winter.txt
43  ls
44  ls -l
45  cp winter.txt summer.txt
46  ls -l
47  history 10
```

13.4. !n

When typing `!` followed by the number preceding the command you want repeated, then the shell will echo the command and execute it.

```
paul@debian5:~/test$ !43
   ls
   file1  file2  summer.txt  winter.txt
```
13.5. Ctrl-r

Another option is to use `ctrl-r` to search in the history. In the screenshot below I only typed `ctrl-r` followed by four characters `apti` and it finds the last command containing these four consecutive characters.

```
paul@debian5:~$ (reverse-i-search)`apti': sudo aptitude install screen
```

13.6. $HISTSIZE

The `$HISTSIZE` variable determines the number of commands that will be remembered in your current environment. Most distributions default this variable to 500 or 1000.

```
paul@debian5:~$ echo $HISTSIZE
500

You can change it to any value you like.

paul@debian5:~$ HISTSIZE=15000
paul@debian5:~$ echo $HISTSIZE
15000
```

13.7. $HISTFILE

The `$HISTFILE` variable points to the file that contains your history. The `bash` shell defaults this value to `~/.bash_history`.

```
paul@debian5:~$ echo $HISTFILE
/home/paul/.bash_history

A session history is saved to this file when you `exit` the session!

Closing a gnome-terminal with the mouse, or typing `reboot` as root will NOT save your terminal's history.
```

13.8. $HISTFILESIZE

The number of commands kept in your history file can be set using `$HISTFILESIZE`.

```
paul@debian5:~$ echo $HISTFILESIZE
15000
```
13.9. (optional) regular expressions

It is possible to use regular expressions when using the bang to repeat commands. The screenshot below switches 1 into 2.

```
paul@debian5:~/test$ cat file1
paul@debian5:~/test$ !c:s/1/2
                cat file2
                hello
paul@debian5:~/test$
```

13.10. (optional) repeating commands in ksh

Repeating a command in the Korn shell is very similar. The Korn shell also has the history command, but uses the letter r to recall lines from history.

This screenshot shows the history command. Note the different meaning of the parameter.

```
$ history 17
17  clear
18  echo hoi
19  history 12
20  echo world
21  history 17

Repeating with r can be combined with the line numbers given by the history command, or with the first few letters of the command.

```
$ r e
      echo world
      world
$ cd /etc
$ r
      cd /etc
$```
13.11. practice: shell history

1. Issue the command `echo The answer to the meaning of life, the universe and everything is 42.`

2. Repeat the previous command using only two characters (there are two solutions!)

3. Display the last 5 commands you typed.

4. Issue the long `echo` from question 1 again, using the line numbers you received from the command in question 3.

5. How many commands can be kept in memory for your current shell session?

6. Where are these commands stored when exiting the shell?

7. How many commands can be written to the **history file** when exiting your current shell session?

8. Make sure your current bash shell remembers the next 5000 commands you type.

9. Open more than one console (press Ctrl-shift-t in gnome-terminal) with the same user account. When is command history written to the history file?
13.12. solution: shell history

1. Issue the command `echo The answer to the meaning of life, the universe and everything is 42`.

   `echo The answer to the meaning of life, the universe and everything is 42`

2. Repeat the previous command using only two characters (there are two solutions!)

   `!!`
   OR
   `
   `!e`

3. Display the last 5 commands you typed.

   ```
   paul@ubu1010:~$ history 5
   52  ls -l
   53  ls
   54  df -h | grep sda
   55  echo The answer to the meaning of life, the universe and everything is 42
   56  history 5
   ```

   You will receive different line numbers.

4. Issue the long `echo` from question 1 again, using the line numbers you received from the command in question 3.

   ```
   paul@ubu1010:~$ !56
   echo The answer to the meaning of life, the universe and everything is 42
   The answer to the meaning of life, the universe and everything is 42
   ```

5. How many commands can be kept in memory for your current shell session?

   `echo $HISTSIZE`

6. Where are these commands stored when exiting the shell?

   `echo $HISTFILE`

7. How many commands can be written to the history file when exiting your current shell session?

   `echo $HISTFILESIZE`

8. Make sure your current bash shell remembers the next 5000 commands you type.

   `HISTSIZE=5000`

9. Open more than one console (press Ctrl-shift-t in gnome-terminal) with the same user account. When is command history written to the history file?

   ```
   when you type `exit`
   ```
Chapter 14. file globbing

Table of Contents

14.1. * asterisk ................................................................. 107
14.2. ? question mark .......................................................... 107
14.3. [] square brackets ...................................................... 107
14.4. a-z and 0-9 ranges ..................................................... 108
14.5. $LANG and square brackets ....................................... 108
14.6. preventing file globbing ............................................. 109
14.7. practice: shell globbing .............................................. 110
14.8. solution: shell globbing .............................................. 111

The shell is also responsible for file globbing (or dynamic filename generation). This chapter will explain file globbing.
14.1. * asterisk

The asterisk * is interpreted by the shell as a sign to generate filenames, matching the asterisk to any combination of characters (even none). When no path is given, the shell will use filenames in the current directory. See the man page of `glob(7)` for more information. (This is part of LPI topic 1.103.3.)

```
[paul@RHELv4u3 gen]$ ls
file1 file2 file3 File4 File55 FileA fileab Fileab FileAB fileabc
[paul@RHELv4u3 gen]$ ls File*
File4 File55 FileA Fileab FileAB
[paul@RHELv4u3 gen]$ ls *ile55
File55
[paul@RHELv4u3 gen]$ ls File*ile55
File55
[paul@RHELv4u3 gen]$ ls File*55
File55
[paul@RHELv4u3 gen]$ ls *ile55
File55
[paul@RHELv4u3 gen]$
```

14.2. ? question mark

Similar to the asterisk, the question mark ? is interpreted by the shell as a sign to generate filenames, matching the question mark with exactly one character.

```
[paul@RHELv4u3 gen]$ ls
file1 file2 file3 File4 File55 FileA fileab Fileab FileAB fileabc
[paul@RHELv4u3 gen]$ ls File?
File4 FileA
[paul@RHELv4u3 gen]$ ls Fil?4
File4
[paul@RHELv4u3 gen]$ ls Fil??
File4 FileA
[paul@RHELv4u3 gen]$ ls File??
File55 Fileab FileAB
[paul@RHELv4u3 gen]$
```

14.3. [] square brackets

The square bracket [ is interpreted by the shell as a sign to generate filenames, matching any of the characters between [ and the first subsequent ]. The order in this list between the brackets is not important. Each pair of brackets is replaced by exactly one character.

```
[paul@RHELv4u3 gen]$ ls
file1 file2 file3 File4 File55 FileA fileab Fileab FileAB fileabc
[paul@RHELv4u3 gen]$ ls File[5A]
FileA
[paul@RHELv4u3 gen]$ ls File[A5]
FileA
[paul@RHELv4u3 gen]$ ls File[A5][5b]
File55
[paul@RHELv4u3 gen]$ ls File[a5][5b]
File55 Fileab
```

107
file globbing

[paul@RHELv4u3 gen]$ ls File[a5][5b][abcdefghijklm]
ls: File[a5][5b][abcdefghijklm]: No such file or directory
[paul@RHELv4u3 gen]$ ls file[a5][5b][abcdefghijklm]
fileabc
[paul@RHELv4u3 gen]$

You can also exclude characters from a list between square brackets with the exclamation mark !. And you are allowed to make combinations of these wild cards.

[paul@RHELv4u3 gen]$ ls file[a5][!Z]
fileab
[paul@RHELv4u3 gen]$ ls file[!5]*
file1  file2  file3  fileab  fileabc
[paul@RHELv4u3 gen]$ ls file[!5]?
fileab
[paul@RHELv4u3 gen]$

14.4. a-z and 0-9 ranges

The bash shell will also understand ranges of characters between brackets.

[paul@RHELv4u3 gen]$ ls file1  file3  File55  fileA  fileab  FileAB  fileabc
[paul@RHELv4u3 gen]$ ls file[a-z]*
fileab  fileab2  fileabc
[paul@RHELv4u3 gen]$ ls file[0-9]
file1  file2  file3
[paul@RHELv4u3 gen]$ ls file[a-z][a-z][0-9]*
fileab2
[paul@RHELv4u3 gen]$

14.5. $LANG and square brackets

But, don't forget the influence of the LANG variable. Some languages include lower case letters in an upper case range (and vice versa).

paul@RHELv4u4:~/test$ ls [A-Z]ile?
file1  file2  file3  File4
paul@RHELv4u4:~/test$ ls [a-z]ile?
file1  file2  file3  File4
paul@RHELv4u4:~/test$ echo $LANG
en_US.UTF-8
paul@RHELv4u4:~/test$ LANG=C
paul@RHELv4u4:~/test$ echo $LANG
C
paul@RHELv4u4:~/test$ ls [a-z]ile?
file1  file2  file3
paul@RHELv4u4:~/test$ ls [A-Z]ile?
File4
paul@RHELv4u4:~/test$
14.6. preventing file globbing

The screenshot below should be no surprise. The `echo *` will echo a * when in an empty directory. And it will echo the names of all files when the directory is not empty.

```
paul@ubu1010:~$ mkdir test42
paul@ubu1010:~$ cd test42
paul@ubu1010:~/test42$ echo *
*
paul@ubu1010:~/test42$ touch file42 file33
paul@ubu1010:~/test42$ echo *
file33 file42
```

Globbing can be prevented using quotes or by escaping the special characters, as shown in this screenshot.

```
paul@ubu1010:~/test42$ echo *
file33 file42
paul@ubu1010:~/test42$ echo \*
*
paul@ubu1010:~/test42$ echo '**'
*
paul@ubu1010:~/test42$ echo ***
*
14.7. practice: shell globbing

1. Create a test directory and enter it.

2. Create files file1 file10 file11 file2 File2 File3 file33 fileAB filea fileA fileAAA file( file 2 (the last one has 6 characters including a space)

3. List (with ls) all files starting with file

4. List (with ls) all files starting with File

5. List (with ls) all files starting with file and ending in a number.

6. List (with ls) all files starting with file and ending with a letter

7. List (with ls) all files starting with File and having a digit as fifth character.

8. List (with ls) all files starting with File and having a digit as fifth character and nothing else.

9. List (with ls) all files starting with a letter and ending in a number.

10. List (with ls) all files that have exactly five characters.

11. List (with ls) all files that start with f or F and end with 3 or A.

12. List (with ls) all files that start with f have i or R as second character and end in a number.

13. List all files that do not start with the letter F.

14. Copy the value of $LANG to $MyLANG.

15. Show the influence of $LANG in listing A-Z or a-z ranges.

16. You receive information that one of your servers was cracked, the cracker probably replaced the ls command. You know that the echo command is safe to use. Can echo replace ls ? How can you list the files in the current directory with echo ?

17. Is there another command besides cd to change directories ?
14.8. solution: shell globbing

1. Create a test directory and enter it.

```sh
mkdir testdir; cd testdir
```

2. Create files file1 file10 file11 file2 File2 File3 file33 fileAB filea fileA fileAAA file( file 2 (the last one has 6 characters including a space)

```sh
touch file1 file10 file11 file2 File2 File3
touch file33 fileAB filea fileA fileAAA
touch "file("
touch "file 2"
```

3. List (with `ls`) all files starting with file

```sh
ls file*
```

4. List (with `ls`) all files starting with File

```sh
ls File*
```

5. List (with `ls`) all files starting with file and ending in a number.

```sh
ls file*[0-9]
```

6. List (with `ls`) all files starting with file and ending with a letter

```sh
ls file*[a-z]
```

7. List (with `ls`) all files starting with File and having a digit as fifth character.

```sh
ls File[0-9]*
```

8. List (with `ls`) all files starting with File and having a digit as fifth character and nothing else.

```sh
ls File[0-9]
```

9. List (with `ls`) all files starting with a letter and ending in a number.

```sh
ls [a-z]*[0-9]
```

10. List (with `ls`) all files that have exactly five characters.

```sh
ls ?? ? ?
```

11. List (with `ls`) all files that start with f or F and end with 3 or A.

```sh
ls [fF]*[3A]
```

12. List (with `ls`) all files that start with f have i or R as second character and end in a number.

```sh
ls f[iR]*[0-9]
```

13. List all files that do not start with the letter F.

```sh
ls ![F]*
```
14. Copy the value of $LANG to $MyLANG.

```
MyLANG=$LANG
```

15. Show the influence of $LANG in listing A-Z or a-z ranges.

```
see example in book
```

16. You receive information that one of your servers was cracked, the cracker probably replaced the `ls` command. You know that the `echo` command is safe to use. Can `echo` replace `ls` ? How can you list the files in the current directory with `echo` ?

```
echo *
```

17. Is there another command besides `cd` to change directories ?

```
pushd popd
```
Part IV. pipes and commands
Chapter 15. redirection and pipes

Table of Contents

15.1. stdin, stdout, and stderr ................................................................. 115
15.2. output redirection .............................................................................. 115
15.3. error redirection .............................................................................. 117
15.4. input redirection .............................................................................. 118
15.5. confusing redirection ........................................................................ 119
15.6. quick file clear .................................................................................. 119
15.7. swapping stdout and stderr ............................................................. 119
15.8. pipes ................................................................................................. 120
15.9. practice: redirection and pipes ........................................................ 121
15.10. solution: redirection and pipes ....................................................... 122

One of the powers of the Unix command line is the use of redirection and pipes.

This chapter first explains redirection of input, output and error streams. It then introduces pipes that consist of several commands.
15.1. stdin, stdout, and stderr

The shell (and almost every other Linux command) takes input from stdin (stream 0) and sends output to stdout (stream 1) and error messages to stderr (stream 2).

The keyboard often serves as stdin, stdout and stderr both go to the display. The shell allows you to redirect these streams.

15.2. output redirection

> stdout

stdout can be redirected with a greater than sign. While scanning the line, the shell will see the > sign and will clear the file.

[paul@RHELv4u3 ~]$ echo It is cold today!
It is cold today!
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$

Note that the > notation is in fact the abbreviation of 1> (stdout being referred to as stream 1).

output file is erased

To repeat: While scanning the line, the shell will see the > sign and will clear the file! This means that even when the command fails, the file will be cleared!

[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ zcho It is cold today! > winter.txt
-bash: zcho: command not found
[paul@RHELv4u3 ~]$ cat winter.txt
[paul@RHELv4u3 ~]$

noclobber

Erasing a file while using > can be prevented by setting the noclobber option.

[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
-bash: winter.txt: cannot overwrite existing file
[paul@RHELv4u3 ~]$ set +o noclobber
[paul@RHELv4u3 ~]$

redirection and pipes

115
overruling noclobber

The noclobber can be overruled with >].

[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
-bash: winter.txt: cannot overwrite existing file
[paul@RHELv4u3 ~]$ echo It is very cold today! >| winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is very cold today!
[paul@RHELv4u3 ~]$ 

>> append

Use >> to append output to a file.

[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ echo Where is the summer ? >> winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
Where is the summer ?
[paul@RHELv4u3 ~]$
15.3. error redirection

2> stderr

Redirecting stderr is done with 2>. This can be very useful to prevent error messages from cluttering your screen. The screenshot below shows redirection of stdout to a file, and stderr to /dev/null. Writing 1> is the same as >.

[paul@RHELv4u3 ~]$ find / > allfiles.txt 2> /dev/null
[paul@RHELv4u3 ~]$ 

2>&1

To redirect both stdout and stderr to the same file, use 2>&1.

[paul@RHELv4u3 ~]$ find / > allfiles_and_errors.txt 2>&1
[paul@RHELv4u3 ~]$ 

Note that the order of redirections is significant. For example, the command

ls > dirlst 2>&1

directs both standard output (file descriptor 1) and standard error (file descriptor 2) to the file dirlst, while the command

ls 2>&1 > dirlst

directs only the standard output to file dirlst, because the standard error made a copy of the standard output before the standard output was redirected to dirlst.
15.4. input redirection

< stdin

Redirecting stdin is done with < (short for 0<).

[paul@RHEL4b ~]$ cat < text.txt
one
two
[paul@RHEL4b ~]$ tr 'onetw' 'ONEZZ' < text.txt
ONE
ZZO
[paul@RHEL4b ~]$

<< here document

The here document (sometimes called here-is-document) is a way to append input until a certain sequence (usually EOF) is encountered. The EOF marker can be typed literally or can be called with Ctrl-D.

[paul@RHEL4b ~]$ cat <<EOF > text.txt
> one
> two
> EOF
[paul@RHEL4b ~]$ cat text.txt
one
two
[paul@RHEL4b ~]$ cat <<brol > text.txt
> brel
> brol
[paul@RHEL4b ~]$ cat text.txt
brel
[paul@RHEL4b ~]$

<<< here string

The here string can be used to directly pass strings to a command. The result is the same as using `echo string | command` (but you have one less process running).

paul@ubu1110~$ base64 <<< linux-training.be
bGludXgtdHJhaW5pbmcuYmUK
paul@ubu1110~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK
linux-training.be

See rfc 3548 for more information about base64.
15.5. confusing redirection

The shell will scan the whole line before applying redirection. The following command line is very readable and is correct.

```
cat winter.txt > snow.txt 2> errors.txt
```

But this one is also correct, but less readable.

```
2> errors.txt cat winter.txt > snow.txt
```

Even this will be understood perfectly by the shell.

```
< winter.txt > snow.txt 2> errors.txt cat
```

15.6. quick file clear

So what is the quickest way to clear a file?

```
>foo
```

And what is the quickest way to clear a file when the `noclobber` option is set?

```
>|bar
```

15.7. swapping stdout and stderr

When filtering an output stream, e.g. through a regular pipe ( | ) you only can filter `stdout`. Say you want to filter out some unimportant error, out of the `stderr` stream. This cannot be done directly, and you need to 'swap' `stdout` and `stderr`. This can be done by using a 4th stream referred to with number 3:

```
3>&1 1>&2 2>&3
```

This Tower Of Hanoi like construction uses a temporary stream 3, to be able to swap `stdout` (1) and `stderr` (2). The following is an example of how to filter out all lines in the `stderr` stream, containing `$error`.

```
$command 3>&1 1>&2 2>&3 | grep -v "$error" 3>&1 1>&2 2>&3
```

But in this example, it can be done in a much shorter way, by using a pipe on STDERR:

```
/usr/bin/$somecommand |& grep -v "$error"
```
15.8. pipes

One of the most powerful advantages of Linux is the use of pipes.

A pipe takes stdout from the previous command and sends it as stdin to the next command. All commands in a pipe run simultaneously.

| vertical bar

Consider the following example.

```
paul@debian5:~/test$ ls /etc > etcfiles.txt
paul@debian5:~/test$ tail -4 etcfiles.txt
X11
xdg
xml
xpdf
paul@debian5:~/test$
```

This can be written in one command line using a pipe.

```
paul@debian5:~/test$ ls /etc | tail -4
X11
xdg
xml
xpdf
paul@debian5:~/test$
```

The pipe is represented by a vertical bar | between two commands.

multiple pipes

One command line can use multiple pipes. All commands in the pipe can run at the same time.

```
paul@deb503:~/test$ ls /etc | tail -4 | tac
xpdf
xml
xdg
X11
```
15.9. practice: redirection and pipes

1. Use `ls` to output the contents of the `/etc/` directory to a file called `etc.txt`.

2. Activate the `noclobber` shell option.

3. Verify that `noclobber` is active by repeating your `ls` on `/etc/`.

4. When listing all shell options, which character represents the `noclobber` option?

5. Deactivate the `noclobber` option.

6. Make sure you have two shells open on the same computer. Create an empty `tailing.txt` file. Then type `tail -f tailing.txt`. Use the second shell to `append` a line of text to that file. Verify that the first shell displays this line.

7. Create a file that contains the names of five people. Use `cat` and output redirection to create the file and use a `here document` to end the input.
15.10. solution: redirection and pipes

1. Use `ls` to output the contents of the `/etc/` directory to a file called `etc.txt`.

```bash
ls /etc > etc.txt
```

2. Activate the `noclobber` shell option.

```bash
set -o noclobber
```

3. Verify that `noclobber` is active by repeating your `ls` on `/etc/`.

```bash
ls /etc > etc.txt (should not work)
```

4. When listing all shell options, which character represents the `noclobber` option?

```bash
echo $- (noclobber is visible as C)
```

5. Deactivate the `noclobber` option.

```bash
set +o noclobber
```

6. Make sure you have two shells open on the same computer. Create an empty `tailing.txt` file. Then type `tail -f tailing.txt`. Use the second shell to append a line of text to that file. Verify that the first shell displays this line.

```bash
paul@deb503:~$ > tailing.txt
paul@deb503:~$ tail -f tailing.txt
hello
world

in the other shell:
paul@deb503:~$ echo hello >> tailing.txt
paul@deb503:~$ echo world >> tailing.txt
```

7. Create a file that contains the names of five people. Use `cat` and output redirection to create the file and use a here document to end the input.

```bash
paul@deb503:~$ cat > tennis.txt << ace
> Justine Henin
> Venus Williams
> Serena Williams
> Martina Hingis
> Kim Clijsters
> ace
paul@deb503:~$ cat tennis.txt
Justine Henin
Venus Williams
Serena Williams
Martina Hingis
Kim Clijsters
paul@deb503:~$
```
Chapter 16. filters

Table of Contents

16.1. cat ............................................................................................................... 124
16.2. tee ............................................................................................................... 124
16.3. grep ............................................................................................................. 124
16.4. cut ............................................................................................................... 126
16.5. tr ................................................................................................................. 126
16.6. wc ............................................................................................................... 127
16.7. sort .............................................................................................................. 128
16.8. uniq ........................................................................................................... 129
16.9. comm ........................................................................................................... 129
16.10. od .............................................................................................................. 130
16.11. sed ............................................................................................................ 131
16.12. pipe examples .......................................................................................... 132
16.13. practice: filters ......................................................................................... 133

Commands that are created to be used with a pipe are often called filters. These filters are very small programs that do one specific thing very efficiently. They can be used as building blocks.

This chapter will introduce you to the most common filters. The combination of simple commands and filters in a long pipe allows you to design elegant solutions.
16.1. cat

When between two pipes, the cat command does nothing (except putting stdin on stdout).

[paul@RHEL4b pipes]$ tac count.txt | cat | cat | cat | cat | cat
five
four
three
two
one
[paul@RHEL4b pipes]$

16.2. tee

Writing long pipes in Unix is fun, but sometimes you might want intermediate results. This is where tee comes in handy. The tee filter puts stdin on stdout and also into a file. So tee is almost the same as cat, except that it has two identical outputs.

[paul@RHEL4b pipes]$ tac count.txt | tee temp.txt | tac
one
two
three
four
five
[paul@RHEL4b pipes]$ cat temp.txt
five
four
three
two
one
[paul@RHEL4b pipes]$

16.3. grep

The grep filter is famous among Unix users. The most common use of grep is to filter lines of text containing (or not containing) a certain string.

[paul@RHEL4b pipes]$ cat tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$ cat tennis.txt | grep Williams
Serena Williams, usa
Venus Williams, USA

You can write this without the cat.

[paul@RHEL4b pipes]$ grep Williams tennis.txt
Serena Williams, usa
Venus Williams, USA

One of the most useful options of grep is grep -i which filters in a case insensitive way.
Another very useful option is **grep -v** which outputs lines not matching the string.

```
[paul@RHEL4b pipes]$ grep -v Fra tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$
```

And of course, both options can be combined to filter all lines not containing a case insensitive string.

```
[paul@RHEL4b pipes]$ grep -vi usa tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

With **grep -A1** one line *after* the result is also displayed.

```
paul@debian5:~/pipes$ grep -A1 Henin tennis.txt
Justine Henin, Bel
Serena Williams, usa
```

With **grep -B1** one line *before* the result is also displayed.

```
paul@debian5:~/pipes$ grep -B1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
```

With **grep -C1** (context) one line *before* and one *after* are also displayed. All three options (A, B, and C) can display any number of lines (using e.g. A2, B4 or C20).

```
paul@debian5:~/pipes$ grep -C1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
```
16.4. cut

The cut filter can select columns from files, depending on a delimiter or a count of bytes. The screenshot below uses cut to filter for the username and userid in the /etc/passwd file. It uses the colon as a delimiter, and selects fields 1 and 3.

```
[paul@RHEL4b pipes]$ cut -d: -f1,3 /etc/passwd | tail -4
Figo:510
Pfaff:511
Harry:516
Hermione:517
[paul@RHEL4b pipes]$
```

When using a space as the delimiter for cut, you have to quote the space.

```
[paul@RHEL4b pipes]$ cut -d" " -f1 tennis.txt
Amelie
Kim
Justine
Serena
Venus
[paul@RHEL4b pipes]$
```

This example uses cut to display the second to the seventh character of /etc/passwd.

```
[paul@RHEL4b pipes]$ cut -c2-7 /etc/passwd | tail -4
igo:x:
fauff:x
arry:x
ermion
[paul@RHEL4b pipes]$
```

16.5. tr

You can translate characters with tr. The screenshot shows the translation of all occurrences of e to E.

```
[paul@RHEL4b pipes]$ cat tennis.txt | tr 'e' 'E'
AmElie MaureSmo, Fra
Kim ClijkstErs, BEL
JustiEnE HEnin, Bel
SErEna Williams, usa
VENus Williams, USA
[paul@RHEL4b pipes]$
```

Here we set all letters to uppercase by defining two ranges.

```
[paul@RHEL4b pipes]$ cat tennis.txt | tr 'a-z' 'A-Z'
AMELIE MAURESMO, FRA
KIM CLIJSTERS, BEL
JUSTINE HENIN, BEL
SERENA WILLIAMS, USA
VENUS WILLIAMS, USA
[paul@RHEL4b pipes]$
```

Here we translate all newlines to spaces.

```
[paul@RHEL4b pipes]$ cat count.txt
one
```

one
two
three
four
five
[paul@RHEL4b pipes]$ cat count.txt | tr '"' '
' one two three four five [paul@RHEL4b pipes]$

The **tr** -*s* filter can also be used to squeeze multiple occurrences of a character to one.

[paul@RHEL4b pipes]$ cat spaces.txt one two three four five six [paul@RHEL4b pipes]$ cat spaces.txt | tr -s '"' one two three four five six [paul@RHEL4b pipes]$

You can also use **tr** to 'encrypt' texts with **rot13**.

[paul@RHEL4b pipes]$ cat count.txt | tr 'a-z' 'nopqrstuvwxyzabcdefghijklm' bar gjb guerr sbhe svir [paul@RHEL4b pipes]$ cat count.txt | tr 'a-z' 'n-za-m' bar gjb guerr sbhe svir [paul@RHEL4b pipes]$

This last example uses **tr** -*d* to delete characters.

paul@debian5:~/pipes$ cat tennis.txt | tr -d e Amli Maursmo, Fra Kim Clijsters, BEL Justin Hnin, Bl Srna Williams, usa Vnus Williams, USA

**16.6. wc**

Counting words, lines and characters is easy with **wc**.

[paul@RHEL4b pipes]$ wc tennis.txt 5 15 100 tennis.txt [paul@RHEL4b pipes]$ wc -l tennis.txt 5 tennis.txt [paul@RHEL4b pipes]$ wc -w tennis.txt 15 tennis.txt [paul@RHEL4b pipes]$ wc -c tennis.txt 100 tennis.txt [paul@RHEL4b pipes]$

127
16.7. sort

The `sort` filter will default to an alphabetical sort.

```
paul@debian5:~/pipes$ cat music.txt
Queen
Brel
Led Zeppelin
Abba
paul@debian5:~/pipes$ sort music.txt
Abba
Brel
Led Zeppelin
Queen
```

But the `sort` filter has many options to tweak its usage. This example shows sorting different columns (column 1 or column 2).

```
[paul@RHEL4b pipes]$ sort -k1 country.txt
Belgium, Brussels, 10
France, Paris, 60
Germany, Berlin, 100
Iran, Teheran, 70
Italy, Rome, 50
[paul@RHEL4b pipes]$ sort -k2 country.txt
Germany, Berlin, 100
Belgium, Brussels, 10
France, Paris, 60
Italy, Rome, 50
Iran, Teheran, 70
```

The screenshot below shows the difference between an alphabetical sort and a numerical sort (both on the third column).

```
[paul@RHEL4b pipes]$ sort -k3 country.txt
Belgium, Brussels, 10
Germany, Berlin, 100
Italy, Rome, 50
France, Paris, 60
Iran, Teheran, 70
[paul@RHEL4b pipes]$ sort -n -k3 country.txt
Belgium, Brussels, 10
Italy, Rome, 50
France, Paris, 60
Iran, Teheran, 70
Germany, Berlin, 100
```
16.8. uniq

With uniq you can remove duplicates from a sorted list.

```
paul@debian5:/pipes$ cat music.txt
Queen
Brel
Queen
Abba
paul@debian5:/pipes$ sort music.txt
Abba
Brel
Queen
Queen
paul@debian5:/pipes$ sort music.txt | uniq
Abba
Brel
Queen
```

uniq can also count occurrences with the -c option.

```
paul@debian5:/pipes$ sort music.txt | uniq -c
  1 Abba
  1 Brel
  2 Queen
```

16.9. comm

Comparing streams (or files) can be done with the comm. By default comm will output three columns. In this example, Abba, Cure and Queen are in both lists, Bowie and Sweet are only in the first file, Turner is only in the second.

```
paul@debian5:/pipes$ cat > list1.txt
Abba
Bowie
Cure
Queen
Sweet
paul@debian5:/pipes$ cat > list2.txt
Abba
Cure
Queen
Turner
paul@debian5:/pipes$ comm list1.txt list2.txt
Abba
Bowie
    Cure
    Queen
Sweet
    Turner
```
The output of `comm` can be easier to read when outputting only a single column. The digits point out which output columns should not be displayed.

```
paul@debian5:~/pipes$ comm -12 list1.txt list2.txt
Abba
Cure
Queen
paul@debian5:~/pipes$ comm -13 list1.txt list2.txt
Turner
paul@debian5:~/pipes$ comm -23 list1.txt list2.txt
Bowie
Sweet
```

## 16.10. od

European humans like to work with ascii characters, but computers store files in bytes. The example below creates a simple file, and then uses `od` to show the contents of the file in hexadecimal bytes.

```
paul@laika:~/test$ cat > text.txt
abcdefg
1234567
paul@laika:~/test$ od -t x1 text.txt
0000000 61 62 63 64 65 66 67 0a 31 32 33 34 35 36 37 0a
0000020
```

The same file can also be displayed in octal bytes.

```
paul@laika:~/test$ od -b text.txt
0000000 141 142 143 144 145 146 147 012 061 062 063 064 065 066 067 012
0000020
```

And here is the file in ascii (or backslashed) characters.

```
paul@laika:~/test$ od -c text.txt
0000000  a  b  c  d  e  f  g  \n  1  2  3  4  5  6  7  \n0000020
```
16.11. sed

The stream editor **sed** can perform editing functions in the stream, using **regular expressions**.

```
paul@debian5:~/pipes$ echo level5 | sed 's/5/42/'
level42
paul@debian5:~/pipes$ echo level5 | sed 's/level/jump/'
jump5
```

Add **g** for global replacements (all occurrences of the string per line).

```
paul@debian5:~/pipes$ echo level5 level7 | sed 's/level/jump/'
jump5 level7
paul@debian5:~/pipes$ echo level5 level7 | sed 's/level/jump/g'
jump5 jump7
```

With **d** you can remove lines from a stream containing a character.

```
paul@debian5:~/test42$ cat tennis.txt
Venus Williams, USA
Martina Hingis, SUI
Justine Henin, BE
Serena Williams, USA
Kim Clijsters, BE
Yanina Wickmayer, BE
paul@debian5:~/test42$ cat tennis.txt | sed '/BE/d'
Venus Williams, USA
Martina Hingis, SUI
Serena Williams, USA
```

16.12. pipe examples

who | wc

How many users are logged on to this system?

[paul@RHEL4b pipes]$ who
root    tty1         Jul 25 10:50
paul    pts/0        Jul 25 09:29 (laika)
Harry   pts/1        Jul 25 12:26 (barry)
paul    pts/2        Jul 25 12:26 (pasha)
[paul@RHEL4b pipes]$ who | wc -l
4

who | cut | sort

Display a sorted list of logged on users.

[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort
Harry
paul
paul
root

Display a sorted list of logged on users, but every user only once.

[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort | uniq
Harry
paul
root

grep | cut

Display a list of all bash user accounts on this computer. Users accounts are explained in detail later.

paul@debian5:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
paul:x:1000:1000: paul,,,:/home/paul:/bin/bash
serena:x:1001:1001::/home/serena:/bin/bash
paul@debian5:~$ grep bash /etc/passwd | cut -d: -f1
root
paul
serena
16.13. practice: filters

1. Put a sorted list of all bash users in bashusers.txt.

2. Put a sorted list of all logged on users in onlineusers.txt.

3. Make a list of all filenames in /etc that contain the string samba.

4. Make a sorted list of all files in /etc that contain the case insensitive string samba.

5. Look at the output of /sbin/ifconfig. Write a line that displays only ip address and the subnet mask.

6. Write a line that removes all non-letters from a stream.

7. Write a line that receives a text file, and outputs all words on a separate line.

8. Write a spell checker on the command line. (There might be a dictionary in /usr/share/dict/.)
16.14. solution: filters

1. Put a sorted list of all bash users in bashusers.txt.
   
   ```bash
grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt
   ```
   
2. Put a sorted list of all logged on users in onlineusers.txt.
   
   ```bash
who | cut -d' ' -f1 | sort > onlineusers.txt
   ```
   
3. Make a list of all filenames in `/etc` that contain the string `samba`.
   
   ```bash
ls /etc | grep samba
   ```
   
4. Make a sorted list of all files in `/etc` that contain the case insensitive string `samba`.
   
   ```bash
ls /etc | grep -i samba | sort
   ```
   
5. Look at the output of `/sbin/ifconfig`. Write a line that displays only IP address and the subnet mask.
   
   ```bash
/sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5
   ```
   
6. Write a line that removes all non-letters from a stream.
   
   ```bash
paul@deb503:~$ cat text
This is, yes really! , a text with ?&* too many str$ange# characters ;-)
paul@deb503:~$ cat text | tr -d ',!$?.*&^%#@;()-'
This is yes really a text with too many strange characters
   ```
   
7. Write a line that receives a text file, and outputs all words on a separate line.
   
   ```bash
paul@deb503:~$ cat text2
it is very cold today without the sun
paul@deb503:~$ cat text2 | tr ' ' '
'
it
is
very
cold
today
without
the
sun
   ```
   
8. Write a spell checker on the command line. (There might be a dictionary in `/usr/share/dict/`.)
   
   ```bash
paul@rhel ~$ echo "The zun is shining today" > text
paul@rhel ~$ cat > DICT
is
shining
sun
the
today
   ```
paul@rhel ~$ cat text | tr 'A-Z' 'a-z\n' | sort | uniq | comm -23 - DICT
zun

You could also add the solution from question number 6 to remove non-letters, and `tr -s ''` to remove redundant spaces.
Chapter 17. basic Unix tools

Table of Contents

17.1. find .................................................................................................................. 137
17.2. locate ............................................................................................................... 138
17.3. date .................................................................................................................. 138
17.4. cal .................................................................................................................... 139
17.5. sleep ............................................................................................................... 139
17.6. time .................................................................................................................. 139
17.7. gzip - gunzip ................................................................................................. 140
17.8. zcat - zmore ................................................................................................. 140
17.9. bzip2 - bunzip2 ............................................................................................. 141
17.10. bzcat - bzmore ............................................................................................ 141
17.11. practice: basic Unix tools ........................................................................... 142
17.12. solution: basic Unix tools ............................................................................ 143

This chapter introduces commands to find or locate files and to compress files, together with other common tools that were not discussed before. While the tools discussed here are technically not considered filters, they can be used in pipes.
17.1. find

The `find` command can be very useful at the start of a pipe to search for files. Here are some examples. You might want to add `2>/dev/null` to the command lines to avoid cluttering your screen with error messages.

Find all files in `/etc` and put the list in etcfiles.txt

```
find /etc > etcfiles.txt
```

Find all files of the entire system and put the list in allfiles.txt

```
find / > allfiles.txt
```

Find files that end in `.conf` in the current directory (and all subdirs).

```
fund . -name "*.conf"
```

Find files of type file (not directory, pipe or etc.) that end in `.conf`.

```
fund . -type f -name "*.conf"
```

Find files of type directory that end in `.bak`.

```
fund /data -type d -name "*.bak"
```

Find files that are newer than file42.txt

```
fund . -newer file42.txt
```

Find can also execute another command on every file found. This example will look for `*.odf` files and copy them to `/backup/`.

```
fund /data -name "*.odf" -exec cp {} /backup/ \;
```

Find can also execute, after your confirmation, another command on every file found. This example will remove `*.odf` files if you approve of it for every file found.

```
fund /data -name "*.odf" -ok rm {} \;
```
17.2. locate

The `locate` tool is very different from `find` in that it uses an index to locate files. This is a lot faster than traversing all the directories, but it also means that it is always outdated. If the index does not exist yet, then you have to create it (as root on Red Hat Enterprise Linux) with the `updatedb` command.

```
[paul@RHEL4b ~]$ locate Samba
warning: locate: could not open database: /var/lib/slocate/slocate.db:...
warning: You need to run the 'updatedb' command (as root) to create th...
Please have a look at /etc/updatedb.conf to enable the daily cron job.
[paul@RHEL4b ~]$ updatedb
fatal error: updatedb: You are not authorized to create a default sloc...
[paul@RHEL4b ~]$ su -
Password:
[root@RHEL4b ~]# updatedb
[root@RHEL4b ~]#
```

Most Linux distributions will schedule the `updatedb` to run once every day.

17.3. date

The `date` command can display the date, time, time zone and more.

```
paul@rhel55 ~$ date
Sat Apr 17 12:44:30 CEST 2010
```

A date string can be customised to display the format of your choice. Check the man page for more options.

```
paul@rhel55 ~$ date +"%A %d-%m-%Y"
Saturday 17-04-2010
```

Time on any Unix is calculated in number of seconds since 1969 (the first second being the first second of the first of January 1970). Use `date +%s` to display Unix time in seconds.

```
paul@rhel55 ~$ date +%s
1271501080
```

When will this seconds counter reach two thousand million ?

```
paul@rhel55 ~$ date -d '1970-01-01 + 2000000000 seconds'
Wed May 18 04:33:20 CEST 2033
```
17.4. cal

The `cal` command displays the current month, with the current day highlighted.

```
paul@rhel55 ~$ cal
April 2010
Su Mo Tu We Th Fr Sa
  1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

You can select any month in the past or the future.

```
paul@rhel55 ~$ cal 2 1970
February 1970
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

17.5. sleep

The `sleep` command is sometimes used in scripts to wait a number of seconds. This example shows a five second `sleep`.

```
paul@rhel55 ~$ sleep 5
paul@rhel55 ~$
```

17.6. time

The `time` command can display how long it takes to execute a command. The `date` command takes only a little time.

```
paul@rhel55 ~$ time date
Sat Apr 17 13:08:27 CEST 2010
real    0m0.014s
user    0m0.008s
sys     0m0.006s
```

The `sleep 5` command takes five `real` seconds to execute, but consumes little `cpu` `time`.

```
paul@rhel55 ~$ time sleep 5
real    0m5.018s
user    0m0.005s
sys     0m0.011s
```
basic Unix tools

This **bzip2** command compresses a file and uses a lot of **cpu time**.

```
paul@rhe155 ~$ time bzip2 text.txt
real    0m2.368s
user    0m0.847s
sys     0m0.539s
```

### 17.7. gzip - gunzip

Users never have enough disk space, so compression comes in handy. The **gzip** command can make files take up less space.

```
paul@rhe155 ~> ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
paul@rhe155 ~> gzip text.txt
paul@rhe155 ~> ls -lh text.txt.gz
-rw-rw-r-- 1 paul paul 760K Apr 17 13:11 text.txt.gz
```

You can get the original back with **gunzip**.

```
paul@rhe155 ~> gunzip text.txt.gz
paul@rhe155 ~> ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
```

### 17.8. zcat - zmore

Text files that are compressed with **gzip** can be viewed with **zcat** and **zmore**.

```
paul@rhe155 ~> head -4 text.txt
/ 
/opt/VBoxGuestAdditions-3.1.6 
/opt/VBoxGuestAdditions-3.1.6/routines.sh
paul@rhe155 ~> gzip text.txt
paul@rhe155 ~> zcat text.txt.gz | head -4
/ 
/opt/VBoxGuestAdditions-3.1.6 
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```
basic Unix tools

17.9. bzip2 - bunzip2

Files can also be compressed with **bzip2** which takes a little more time than **gzip**, but compresses better.

```
paul@rhe155 ~$ bzip2 text.txt
paul@rhe155 ~$ ls -lh text.txt.bz2
-rw-rw-r-- 1 paul paul 569K Apr 17 13:11 text.txt.bz2
```

Files can be uncompressed again with **bunzip2**.

```
paul@rhe155 ~$ bunzip2 text.txt.bz2
paul@rhe155 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
```

17.10. bzcat - bzmore

And in the same way **bzcat** and **bzmore** can display files compressed with **bzip2**.

```
paul@rhe155 ~$ bzip2 text.txt
paul@rhe155 ~$ bzcat text.txt.bz2 | head -4
/  
/opt/VBoxGuestAdditions-3.1.6  
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```
17.11. practice: basic Unix tools

1. Explain the difference between these two commands. This question is very important. If you don't know the answer, then look back at the shell chapter.

   `find /data -name "*.txt"`
   `find /data -name *.txt`

2. Explain the difference between these two statements. Will they both work when there are 200 .odf files in /data? How about when there are 2 million .odf files?

   `find /data -name "*.odf" > data_odf.txt`
   `find /data/*.odf > data_odf.txt`

3. Write a find command that finds all files created after January 30th 2010.

4. Write a find command that finds all *.odf files created in September 2009.

5. Count the number of *.conf files in /etc and all its subdirs.

6. Two commands that do the same thing: copy *.odf files to /backup/. What would be a reason to replace the first command with the second? Again, this is an important question.

   `cp -r /data/*.odf /backup/
   find /data -name "*.odf" -exec cp {} /backup/ \;`

7. Create a file called loctest.txt. Can you find this file with locate? Why not? How do you make locate find this file?

8. Use find and -exec to rename all .htm files to .html.

9. Issue the date command. Now display the date in YYYY/MM/DD format.

10. Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special?
17.12. solution: basic Unix tools

1. Explain the difference between these two commands. This question is very important. If you don’t know the answer, then look back at the shell chapter.

```
find /data -name "*.txt"
find /data -name *.txt
```

When `*.txt` is quoted then the shell will not touch it. The `find` tool will look in the `/data` for all files ending in `.txt`.

When `*.txt` is not quoted then the shell might expand this (when one or more files that ends in `.txt` exist in the current directory). The `find` might show a different result, or can result in a syntax error.

2. Explain the difference between these two statements. Will they both work when there are 200 `.odf` files in `/data`? How about when there are 2 million `.odf` files?

```
find /data -name "*.odf" > data_odf.txt
find /data/*.odf > data_odf.txt
```

The first `find` will output all `.odf` filenames in `/data` and all subdirectories. The shell will redirect this to a file.

The second `find` will output all files named `.odf` in `/data` and will also output all files that exist in directories named `*.odf` (in `/data`).

With two million files the command line would be expanded beyond the maximum that the shell can accept. The last part of the command line would be lost.

3. Write a `find` command that finds all files created after January 30th 2010.

```
touch -t 201001302359 marker_date
find . -type f -newer marker_date
```

There is another solution:

```
fine . -type f -newerat "20100130 23:59:59"
```

4. Write a `find` command that finds all `*.odf` files created in September 2009.

```
touch -t 200908312359 marker_start
touch -t 200910010000 marker_end
find . -type f -name "*.odf" -newer marker_start ! -newer marker_end
```

The exclamation mark `!` `-newer` can be read as `not newer`.

5. Count the number of `*.conf` files in `/etc` and all its subdirs.

```
find /etc -type f -name "*.conf" | wc -l
```

6. Two commands that do the same thing: copy `*.odf` files to `/backup/`. What would be a reason to replace the first command with the second? Again, this is an important question.

```
cp -r /data/*.odf /backup/
```
find /data -name "*.odf" -exec cp {} /backup/ \\;

The first might fail when there are too many files to fit on one command line.

7. Create a file called loctest.txt. Can you find this file with locate? Why not? How do you make locate find this file?

You cannot locate this with locate because it is not yet in the index.
updatedb

8. Use find and -exec to rename all .htm files to .html.

paul@rhel55 ~$ find . -name '*.htm'
./one.htm
./two.htm
paul@rhel55 ~$ find . -name '*.htm' -exec mv {} {}l \\;
paul@rhel55 ~$ find . -name '*.htm*' 
./one.html
./two.html

9. Issue the date command. Now display the date in YYYY/MM/DD format.

date +%Y/%m/%d

10. Issue the cal command. Display a calendar of 1582 and 1752. Notice anything special?

cal 1582

The calendars are different depending on the country. Check http://linux-training.be/files/studentfiles/dates.txt
Chapter 18. Introduction to vi

Table of Contents

18.1. command mode and insert mode ............................................................... 147
18.2. start typing (a A i I o O) ........................................................................... 147
18.3. replace and delete a character (r x X) ....................................................... 148
18.4. undo and repeat (u .) .................................................................................. 148
18.5. cut, copy and paste a line (dd yy p P) ....................................................... 148
18.6. cut, copy and paste lines (3dd 2yy) ........................................................ 149
18.7. start and end of a line (0 or ^ and $) ....................................................... 149
18.8. join two lines (J) and more ........................................................................ 149
18.9. words (w b) ................................................................................................ 150
18.10. save (or not) and exit (:w :q :q! ) ............................................................. 150
18.11. Searching (/ ?) .......................................................................................... 151
18.12. replace all (:1,$ s/foo/bar/g ) ................................................................. 151
18.13. reading files (:r :r !cmd) ......................................................................... 151
18.14. text buffers ............................................................................................... 152
18.15. multiple files ............................................................................................ 152
18.16. abbreviations ........................................................................................... 152
18.17. key mappings ........................................................................................... 153
18.18. setting options .......................................................................................... 153
18.19. practice: vi(m) .......................................................................................... 154
18.20. solution: vi(m) .......................................................................................... 155

The vi editor is installed on almost every Unix. Linux will very often install vim (vi improved) which is similar. Every system administrator should know vi(m), because it is an easy tool to solve problems.

The vi editor is not intuitive, but once you get to know it, vi becomes a very powerful application. Most Linux distributions will include the vintutor which is a 45 minute lesson in vi(m).
18.1. command mode and insert mode

The vi editor starts in **command mode**. In command mode, you can type commands. Some commands will bring you into **insert mode**. In insert mode, you can type text. The escape key will return you to command mode.

<table>
<thead>
<tr>
<th>key</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esc</td>
<td>set vi(m) in command mode.</td>
</tr>
</tbody>
</table>

18.2. start typing (a A i I o O)

The difference between a A i I o and O is the location where you can start typing. a will append after the current character and A will append at the end of the line. i will insert before the current character and I will insert at the beginning of the line. o will put you in a new line after the current line and O will put you in a new line before the current line.

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>start typing after the current character</td>
</tr>
<tr>
<td>A</td>
<td>start typing at the end of the current line</td>
</tr>
<tr>
<td>i</td>
<td>start typing before the current character</td>
</tr>
<tr>
<td>I</td>
<td>start typing at the start of the current line</td>
</tr>
<tr>
<td>o</td>
<td>start typing on a new line after the current line</td>
</tr>
<tr>
<td>O</td>
<td>start typing on a new line before the current line</td>
</tr>
</tbody>
</table>
18.3. replace and delete a character (r x X)

When in command mode (it doesn't hurt to hit the escape key more than once) you can use the x key to delete the current character. The big X key (or shift x) will delete the character left of the cursor. Also when in command mode, you can use the r key to replace one single character. The r key will bring you in insert mode for just one key press, and will return you immediately to command mode.

Table 18.3. replace and delete

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>delete the character below the cursor</td>
</tr>
<tr>
<td>X</td>
<td>delete the character before the cursor</td>
</tr>
<tr>
<td>r</td>
<td>replace the character below the cursor</td>
</tr>
<tr>
<td>p</td>
<td>paste after the cursor (here the last deleted character)</td>
</tr>
<tr>
<td>xp</td>
<td>switch two characters</td>
</tr>
</tbody>
</table>

18.4. undo and repeat (u .)

When in command mode, you can undo your mistakes with u. You can do your mistakes twice with . (in other words, the . will repeat your last command).

Table 18.4. undo and repeat

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>undo the last action</td>
</tr>
<tr>
<td>.</td>
<td>repeat the last action</td>
</tr>
</tbody>
</table>

18.5. cut, copy and paste a line (dd yy p P)

When in command mode, dd will cut the current line. yy will copy the current line. You can paste the last copied or cut line after (p) or before (P) the current line.

Table 18.5. cut, copy and paste a line

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd</td>
<td>cut the current line</td>
</tr>
<tr>
<td>yy</td>
<td>(yank yank) copy the current line</td>
</tr>
<tr>
<td>p</td>
<td>paste after the current line</td>
</tr>
<tr>
<td>P</td>
<td>paste before the current line</td>
</tr>
</tbody>
</table>
18.6. cut, copy and paste lines (3dd 2yy)

When in command mode, before typing dd or yy, you can type a number to repeat the command a number of times. Thus, 5dd will cut 5 lines and 4yy will copy (yank) 4 lines. That last one will be noted by vi in the bottom left corner as "4 line yanked".

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3dd</td>
<td>cut three lines</td>
</tr>
<tr>
<td>4yy</td>
<td>copy four lines</td>
</tr>
</tbody>
</table>

18.7. start and end of a line (0 or ^ and $)

When in command mode, the 0 and the caret ^ will bring you to the start of the current line, whereas the $ will put the cursor at the end of the current line. You can add 0 and $ to the d command. d0 will delete every character between the current character and the start of the line. Likewise d$ will delete everything from the current character till the end of the line. Similarly y0 and y$ will yank till start and end of the current line.

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>jump to start of current line</td>
</tr>
<tr>
<td>^</td>
<td>jump to start of current line</td>
</tr>
<tr>
<td>$</td>
<td>jump to end of current line</td>
</tr>
<tr>
<td>d0</td>
<td>delete until start of line</td>
</tr>
<tr>
<td>d$</td>
<td>delete until end of line</td>
</tr>
</tbody>
</table>

18.8. join two lines (J) and more

When in command mode, pressing J will append the next line to the current line. With yyp you duplicate a line and with ddp you switch two lines.

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>join two lines</td>
</tr>
<tr>
<td>yyp</td>
<td>duplicate a line</td>
</tr>
<tr>
<td>ddp</td>
<td>switch two lines</td>
</tr>
</tbody>
</table>
18.9. words (w b)

When in command mode, \texttt{w} will jump to the next word and \texttt{b} will move to the previous word. \texttt{w} and \texttt{b} can also be combined with \texttt{d} and \texttt{y} to copy and cut words (\texttt{dw db yw yb}).

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{w}</td>
<td>forward one word</td>
</tr>
<tr>
<td>\texttt{b}</td>
<td>back one word</td>
</tr>
<tr>
<td>\texttt{3w}</td>
<td>forward three words</td>
</tr>
<tr>
<td>\texttt{dw}</td>
<td>delete one word</td>
</tr>
<tr>
<td>\texttt{yw}</td>
<td>yank (copy) one word</td>
</tr>
<tr>
<td>\texttt{5yb}</td>
<td>yank five words back</td>
</tr>
<tr>
<td>\texttt{7dw}</td>
<td>delete seven words</td>
</tr>
</tbody>
</table>

18.10. save (or not) and exit (:w :q :q!)

Pressing the colon : will allow you to give instructions to \texttt{vi} (technically speaking, typing the colon will open the \texttt{ex} editor). \texttt{:w} will write (save) the file, \texttt{:q} will quit an unchanged file without saving, and \texttt{:q!} will quit \texttt{vi} discarding any changes. \texttt{:wq} will save and quit and is the same as typing \texttt{ZZ} in command mode.

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{:w}</td>
<td>save (write)</td>
</tr>
<tr>
<td>\texttt{:w fname}</td>
<td>save as fname</td>
</tr>
<tr>
<td>\texttt{:q}</td>
<td>quit</td>
</tr>
<tr>
<td>\texttt{:wq}</td>
<td>save and quit</td>
</tr>
<tr>
<td>\texttt{ZZ}</td>
<td>save and quit</td>
</tr>
<tr>
<td>\texttt{:q!}</td>
<td>quit (discarding your changes)</td>
</tr>
<tr>
<td>\texttt{:w!}</td>
<td>save (and write to non-writable file!)</td>
</tr>
</tbody>
</table>

The last one is a bit special. With \texttt{:w!} \texttt{vi} will try to \texttt{chmod} the file to get write permission (this works when you are the owner) and will \texttt{chmod} it back when the write succeeds. This should always work when you are root (and the file system is writable).
18.11. Searching (/ ?)

When in command mode typing / will allow you to search in vi for strings (can be a regular expression). Typing /foo will do a forward search for the string foo and typing ?bar will do a backward search for bar.

Table 18.11. searching

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/string</td>
<td>forward search for string</td>
</tr>
<tr>
<td>?string</td>
<td>backward search for string</td>
</tr>
<tr>
<td>n</td>
<td>go to next occurrence of search string</td>
</tr>
<tr>
<td>/^string</td>
<td>forward search string at beginning of line</td>
</tr>
<tr>
<td>/string$</td>
<td>forward search string at end of line</td>
</tr>
<tr>
<td>/br[aeio]l</td>
<td>search for bral brel bril and brol</td>
</tr>
<tr>
<td>&lt;he&gt;</td>
<td>search for the word he (and not for here or the)</td>
</tr>
</tbody>
</table>

18.12. replace all ( :1,$ s/foo/bar/g )

To replace all occurrences of the string foo with bar, first switch to ex mode with :. Then tell vi which lines to use, for example 1,$ will do the replace all from the first to the last line. You can write 1,5 to only process the first five lines. The s/foo/bar/g will replace all occurrences of foo with bar.

Table 18.12. replace

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>:4,8 s/foo/bar/g</td>
<td>replace foo with bar on lines 4 to 8</td>
</tr>
<tr>
<td>:1,$ s/foo/bar/g</td>
<td>replace foo with bar on all lines</td>
</tr>
</tbody>
</table>

18.13. reading files (:r :r !cmd)

When in command mode, :r foo will read the file named foo, :r !foo will execute the command foo. The result will be put at the current location. Thus :r !ls will put a listing of the current directory in your text file.

Table 18.13. read files and input

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>:r fname</td>
<td>(read) file fname and paste contents</td>
</tr>
<tr>
<td>:r !cmd</td>
<td>execute cmd and paste its output</td>
</tr>
</tbody>
</table>
18.14. text buffers

There are 36 buffers in vi to store text. You can use them with the " character.

Table 18.14. text buffers

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;add&quot;</td>
<td>delete current line and put text in buffer a</td>
</tr>
<tr>
<td>&quot;g7yy&quot;</td>
<td>copy seven lines into buffer g</td>
</tr>
<tr>
<td>&quot;ap&quot;</td>
<td>paste from buffer a</td>
</tr>
</tbody>
</table>

18.15. multiple files

You can edit multiple files with vi. Here are some tips.

Table 18.15. multiple files

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi file1 file2 file3</td>
<td>start editing three files</td>
</tr>
<tr>
<td>:args</td>
<td>lists files and marks active file</td>
</tr>
<tr>
<td>:n</td>
<td>start editing the next file</td>
</tr>
<tr>
<td>:e</td>
<td>toggle with last edited file</td>
</tr>
<tr>
<td>:rew</td>
<td>rewind file pointer to first file</td>
</tr>
</tbody>
</table>

18.16. abbreviations

With :ab you can put abbreviations in vi. Use :una to undo the abbreviation.

Table 18.16. abbreviations

<table>
<thead>
<tr>
<th>command</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ab str long string</td>
<td>abbreviate str to be 'long string'</td>
</tr>
<tr>
<td>:una str</td>
<td>un-abbreviate str</td>
</tr>
</tbody>
</table>
18.17. key mappings

Similarly to their abbreviations, you can use mappings with :map for command mode and :map! for insert mode.

This example shows how to set the F6 function key to toggle between set number and set nonumber. The <bar> separates the two commands, set number! toggles the state and set number? reports the current state.

:map <F6> :set number!<bar>set number?<CR>

18.18. setting options

Some options that you can set in vim.

:set number  ( also try :se nu )
:set nonumber
:syntax on
:syntax off
:set all  (list all options)
:set tabstop=8
:set tx   (CR/LF style endings)
:set notx

You can set these options (and much more) in ~/.vimrc for vim or in ~/.exrc for standard vi.

paul@barry:$ cat ~/.vimrc
set number
set tabstop=8
set textwidth=78
map <F6> :set number!<bar>set number?<CR>
paul@barry:$
18.19. practice: vi(m)

1. Start the vimtutor and do some or all of the exercises. You might need to run `aptitude install vim` on xubuntu.

2. What 3 key combination in command mode will duplicate the current line.

3. What 3 key combination in command mode will switch two lines' place (line five becomes line six and line six becomes line five).

4. What 2 key combination in command mode will switch a character's place with the next one.

5. vi can understand macro's. A macro can be recorded with q followed by the name of the macro. So qa will record the macro named a. Pressing q again will end the recording. You can recall the macro with @ followed by the name of the macro. Try this example: i 1 'Escape Key' qa yyp 'Ctrl a' q 5@a (Ctrl a will increase the number with one).

6. Copy /etc/passwd to your ~/passwd. Open the last one in vi and press Ctrl v. Use the arrow keys to select a Visual Block, you can copy this with y or delete it with d. Try pasting it.

7. What does dwwP do when you are at the beginning of a word in a sentence?
18.20. solution: vi(m)

1. Start the vimtutor and do some or all of the exercises. You might need to run
   `aptitude install vim` on xubuntu.
   
   `vimtutor`

2. What 3 key combination in command mode will duplicate the current line.
   `yyp`

3. What 3 key combination in command mode will switch two lines' place (line five
   becomes line six and line six becomes line five).
   `ddp`

4. What 2 key combination in command mode will switch a character's place with
   the next one.
   `xp`

5. vi can understand macro's. A macro can be recorded with q followed by the name
   of the macro. So qa will record the macro named a. Pressing q again will end the
   recording. You can recall the macro with @ followed by the name of the macro. Try
   this example: i 1 'Escape Key' qa yyp 'Ctrl a' q 5@a (Ctrl a will increase the number
   with one).

6. Copy /etc/passwd to your ~/passwd. Open the last one in vi and press Ctrl v. Use
   the arrow keys to select a Visual Block, you can copy this with y or delete it with
   d. Try pasting it.
   
   `cp /etc/passwd ~`
   `vi passwd`
   `(press Ctrl-V)`

7. What does `dwwP` do when you are at the beginning of a word in a sentence?
   
   `dwwP` can switch the current word with the next word.
Part VI. scripting
Chapter 19. scripting introduction

Table of Contents

19.1. prerequisites ................................................................. 158
19.2. hello world ................................................................. 158
19.3. she-bang ................................................................. 158
19.4. comment ................................................................. 159
19.5. variables ................................................................. 159
19.6. sourcing a script ......................................................... 159
19.7. troubleshooting a script ............................................. 159
19.8. prevent setuid root spoofing .................................... 160
19.9. practice: introduction to scripting ................................. 161
19.10. solution: introduction to scripting ............................. 162

Shells like bash and Korn have support for programming constructs that can be saved as scripts. These scripts in turn then become more shell commands. Many Linux commands are scripts. User profile scripts are run when a user logs on and init scripts are run when a daemon is stopped or started.

This means that system administrators also need basic knowledge of scripting to understand how their servers and their applications are started, updated, upgraded, patched, maintained, configured and removed, and also to understand how a user environment is built.

The goal of this chapter is to give you enough information to be able to read and understand scripts. Not to become a writer of complex scripts.
19.1. prerequisites

You should have read and understood part III shell expansion and part IV pipes and commands before starting this chapter.

19.2. hello world

Just like in every programming course, we start with a simple hello_world script. The following script will output Hello World.

```bash
echo Hello World
```

After creating this simple script in vi or with echo, you'll have to chmod +x hello_world to make it executable. And unless you add the scripts directory to your path, you'll have to type the path to the script for the shell to be able to find it.

```
[paul@RHEL4a ~]$ echo echo Hello World > hello_world
[paul@RHEL4a ~]$ chmod +x hello_world
[paul@RHEL4a ~]$ ./hello_world
Hello World
[paul@RHEL4a ~]$
```

19.3. she-bang

Let's expand our example a little further by putting `#!/bin/bash` on the first line of the script. The `#!` is called a she-bang (sometimes called sha-bang), where the she-bang is the first two characters of the script.

```bash
#!/bin/bash
echo Hello World
```

You can never be sure which shell a user is running. A script that works flawlessly in bash might not work in ksh, csh, or dash. To instruct a shell to run your script in a certain shell, you can start your script with a she-bang followed by the shell it is supposed to run in. This script will run in a bash shell.

```bash
#!/bin/bash
echo -n hello
echo A bash subshell `echo -n hello`
```

This script will run in a Korn shell (unless /bin/ksh is a hard link to /bin/bash). The /etc/shells file contains a list of shells on your system.

```bash
#!/bin/ksh
echo -n hello
echo A Korn subshell `echo -n hello`
```
19.4. comment

Let's expand our example a little further by adding comment lines.

```bash
#!/bin/bash
#
# Hello World Script
#
echo Hello World
```

19.5. variables

Here is a simple example of a variable inside a script.

```bash
#!/bin/bash
#
# simple variable in script
#
var1=4
echo var1 = $var1
```

Scripts can contain variables, but since scripts are run in their own shell, the variables do not survive the end of the script.

```
[paul@RHEL4a ~]$ echo $var1
[paul@RHEL4a ~]$ ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
var1 = 4
[paul@RHEL4a ~]$
```

19.6. sourcing a script

Luckily, you can force a script to run in the same shell; this is called sourcing a script.

```
[paul@RHEL4a ~]$ source ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```

The above is identical to the below.

```
[paul@RHEL4a ~]$ ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```
19.7. troubleshooting a script

Another way to run a script in a separate shell is by typing `bash` with the name of the script as a parameter.

```bash
paul@debian6~/test$ bash runme
42
```

Expanding this to `bash -x` allows you to see the commands that the shell is executing (after shell expansion).

```bash
paul@debian6~/test$ bash -x runme
+ var4=42
+ echo 42
42
paul@debian6~/test$ cat runme
# the runme script
var4=42
echo $var4
paul@debian6~/test$
```

Notice the absence of the commented (#) line, and the replacement of the variable before execution of `echo`.

19.8. prevent setuid root spoofing

Some user may try to perform setuid based script root spoofing. This is a rare but possible attack. To improve script security and to avoid interpreter spoofing, you need to add `--` after the `#!/bin/bash`, which disables further option processing so the shell will not accept any options.

```bash
#!/bin/bash --
```

Any arguments after the `--` are treated as filenames and arguments. An argument of `-` is equivalent to `--`. 
19.9. practice: introduction to scripting

0. Give each script a different name, keep them for later!

1. Write a script that outputs the name of a city.

2. Make sure the script runs in the bash shell.

3. Make sure the script runs in the Korn shell.

4. Create a script that defines two variables, and outputs their value.

5. The previous script does not influence your current shell (the variables do not exist outside of the script). Now run the script so that it influences your current shell.

6. Is there a shorter way to source the script?

7. Comment your scripts so that you know what they are doing.
19.10. solution: introduction to scripting

0. Give each script a different name, keep them for later!

1. Write a script that outputs the name of a city.

   $ echo 'echo Antwerp' > first.bash
   $ chmod +x first.bash
   $ ./first.bash
   Antwerp

2. Make sure the script runs in the bash shell.

   $ cat first.bash
   #!/bin/bash
   echo Antwerp

3. Make sure the script runs in the Korn shell.

   $ cat first.bash
   #!/bin/ksh
   echo Antwerp

   Note that while first.bash will technically work as a Korn shell script, the name ending
   in .bash is confusing.

4. Create a script that defines two variables, and outputs their value.

   $ cat second.bash
   #!/bin/bash
   var33=300
   var42=400

   echo $var33 $var42

5. The previous script does not influence your current shell (the variables do not exist
   outside of the script). Now run the script so that it influences your current shell.

   source second.bash

6. Is there a shorter way to source the script?

   . ./second.bash

7. Comment your scripts so that you know what they are doing.

   $ cat second.bash
   #!/bin/bash
   # script to test variables and sourcing

   # define two variables
   var33=300
   var42=400

   # output the value of these variables
   echo $var33 $var42
Chapter 20. scripting loops

Table of Contents

20.1. test [ ] ......................................................................................................... 164
20.2. if then else .................................................................................................... 165
20.3. if then elif .................................................................................................. 165
20.4. for loop ....................................................................................................... 165
20.5. while loop .................................................................................................. 166
20.6. until loop .................................................................................................... 166
20.7. practice: scripting tests and loops .............................................................. 167
20.8. solution: scripting tests and loops .............................................................. 168
20.1. test [ ]

The **test** command can test whether something is true or false. Let's start by testing whether 10 is greater than 55.

```
[paul@RHEL4b ~]$ test 10 -gt 55 ; echo $?
1
[paul@RHEL4b ~]$
```

The test command returns 1 if the test fails. And as you see in the next screenshot, test returns 0 when a test succeeds.

```
[paul@RHEL4b ~]$ test 56 -gt 55 ; echo $?
0
[paul@RHEL4b ~]$
```

If you prefer true and false, then write the test like this.

```
[paul@RHEL4b ~]$ test 56 -gt 55 && echo true || echo false
true
[paul@RHEL4b ~]$ test 6 -gt 55 && echo true || echo false
false
```

The test command can also be written as square brackets, the screenshot below is identical to the one above.

```
[paul@RHEL4b ~]$ [ 56 -gt 55 ] && echo true || echo false
true
[paul@RHEL4b ~]$ [ 6 -gt 55 ] && echo true || echo false
false
```

Below are some example tests. Take a look at **man test** to see more options for tests.

```
[ -d foo ]             Does the directory foo exist ?
[ -e bar ]             Does the file bar exist ?
[ '/etc' = $PWD ]      Is the string /etc equal to the variable $PWD ?
[ $1 != 'secret' ]     Is the first parameter different from secret ?
[ 55 -lt $bar ]        Is 55 less than the value of $bar ?
[ $foo -ge 1000 ]      Is the value of $foo greater or equal to 1000 ?
[ "abc" < $bar ]       Does abc sort before the value of $bar ?
[ -f foo ]             Is foo a regular file ?
[ -r bar ]             Is bar a readable file ?
[ foo -nt bar ]        Is file foo newer than file bar ?
[ -o nounset ]         Is the shell option nounset set ?
```

Tests can be combined with logical AND and OR.

```
paul@RHEL4b:~$ [ 66 -gt 55 -a 66 -lt 500 ] && echo true || echo false
true
paul@RHEL4b:~$ [ 66 -gt 55 -a 660 -lt 500 ] && echo true || echo false
false
paul@RHEL4b:~$ [ 66 -gt 55 -o 660 -lt 500 ] && echo true || echo false
true
```
20.2. if then else

The if then else construction is about choice. If a certain condition is met, then execute something, else execute something else. The example below tests whether a file exists, and if the file exists then a proper message is echoed.

```bash
#!/bin/bash
if [ -f isit.txt ]
then echo isit.txt exists!
else echo isit.txt not found!
fi
```

If we name the above script 'choice', then it executes like this.

```
[paul@RHEL4a scripts]$ ./choice
isit.txt not found!
[paul@RHEL4a scripts]$ touch isit.txt
[paul@RHEL4a scripts]$ ./choice
isit.txt exists!
[paul@RHEL4a scripts]$
```

20.3. if then elif

You can nest a new if inside an else with elif. This is a simple example.

```bash
#!/bin/bash
count=42
if [ $count -eq 42 ]
then
 echo "42 is correct."
elif [ $count -gt 42 ]
then
 echo "Too much."
else
 echo "Not enough."
fi
```

20.4. for loop

The example below shows the syntax of a classical for loop in bash.

```bash
for i in 1 2 4
do
 echo $i
done
```

An example of a for loop combined with an embedded shell.

```bash
#!/bin/ksh
for counter in `seq 1 20`
do
 echo counting from 1 to 20, now at $counter
 sleep 1
done
```
The same example as above can be written without the embedded shell using the bash `{from..to}` shorthand.

```bash
#!/bin/bash
for counter in {1..20}
  do
echo counting from 1 to 20, now at $counter
sleep 1
done
```

This `for` loop uses file globbing (from the shell expansion). Putting the instruction on the command line has identical functionality.

```bash
kahlan@solexp11$ ls
count.ksh  go.ksh
kahlan@solexp11$ for file in *.ksh ; do cp $file $file.backup ; done
kahlan@solexp11$ ls
count.ksh  count.ksh.backup  go.ksh  go.ksh.backup
```

## 20.5. while loop

Below a simple example of a `while` loop.

```bash
i=100;
while [ $i -ge 0 ] ;
do
  echo Counting down, from 100 to 0, now at $i;
  let i--;
done
```

Endless loops can be made with `while true` or `while :`, where the `colon` is the equivalent of `no operation` in the Korn and bash shells.

```bash
#!/bin/ksh
# endless loop
while :
doinfo hello
sleep 1
done
```

## 20.6. until loop

Below a simple example of an `until` loop.

```bash
let i=100;
until [ $i -le 0 ] ;
do
  echo Counting down, from 100 to 1, now at $i;
  let i--;
done
```
20.7. practice: scripting tests and loops

1. Write a script that uses a for loop to count from 3 to 7.
2. Write a script that uses a for loop to count from 1 to 17000.
3. Write a script that uses a while loop to count from 3 to 7.
4. Write a script that uses an until loop to count down from 8 to 4.
5. Write a script that counts the number of files ending in .txt in the current directory.
6. Wrap an if statement around the script so it is also correct when there are zero files ending in .txt.
20.8. solution: scripting tests and loops

1. Write a script that uses a for loop to count from 3 to 7.

```bash
#!/bin/bash

for i in 3 4 5 6 7
do
    echo Counting from 3 to 7, now at $i
done
```

2. Write a script that uses a for loop to count from 1 to 17000.

```bash
#!/bin/bash

for i in `seq 1 17000`
do
    echo Counting from 1 to 17000, now at $i
done
```

3. Write a script that uses a while loop to count from 3 to 7.

```bash
#!/bin/bash

i=3
while [ $i -le 7 ]
do
    echo Counting from 3 to 7, now at $i
    let i=i+1
done
```

4. Write a script that uses an until loop to count down from 8 to 4.

```bash
#!/bin/bash

i=8
until [ $i -lt 4 ]
do
    echo Counting down from 8 to 4, now at $i
    let i=i-1
done
```

5. Write a script that counts the number of files ending in .txt in the current directory.

```bash
#!/bin/bash

let i=0
for file in *.txt
do
    let i++
done
echo "There are $i files ending in .txt"
```

6. Wrap an if statement around the script so it is also correct when there are zero files ending in .txt.

```bash
#!/bin/bash

ls *.txt > /dev/null 2>&1
if [ $? -ne 0 ]
```
then echo "There are 0 files ending in .txt"
else
  let i=0
  for file in *.txt
do
    let i++
done
  echo "There are $i files ending in .txt"
fi
# Chapter 21. scripting parameters

## Table of Contents

- 21.1. script parameters .......................................................... 171
- 21.2. shift through parameters .............................................. 172
- 21.3. runtime input ................................................................. 172
- 21.4. sourcing a config file ...................................................... 173
- 21.5. get script options with getopts ....................................... 174
- 21.6. get shell options with shopt .......................................... 175
- 21.7. practice: parameters and options ................................. 176
- 21.8. solution: parameters and options ................................. 177
A **bash** shell script can have parameters. The numbering you see in the script below continues if you have more parameters. You also have special parameters containing the number of parameters, a string of all of them, and also the process id, and the last return code. The man page of **bash** has a full list.

```bash
#!/bin/bash
echo The first argument is $1
echo The second argument is $2
echo The third argument is $3

echo \$ $$  PID of the script
echo \# $#  count arguments
echo \? $?  last return code
echo \* $*  all the arguments
```

Below is the output of the script above in action.

```
[paul@RHEL4a scripts]$ ./pars one two three
The first argument is one
The second argument is two
The third argument is three
$ 5610 PID of the script
# 3 count arguments
? 0 last return code
* one two three all the arguments
```

Once more the same script, but with only two parameters.

```
[paul@RHEL4a scripts]$ ./pars 1 2
The first argument is 1
The second argument is 2
The third argument is
$ 5612 PID of the script
# 2 count arguments
? 0 last return code
* 1 2 all the arguments
[paul@RHEL4a scripts]$
```

Here is another example, where we use **$0**. The **$0** parameter contains the name of the script.

```
paul@debian6~$ cat myname
this script is called $0
paul@debian6~$ ./myname
this script is called ./myname
paul@debian6~$ mv myname test42
paul@debian6~$ ./test42
this script is called ./test42
```
21.2. shift through parameters

The \texttt{shift} statement can parse all \texttt{parameters} one by one. This is a sample script.

```
kahlan@solexp11$ cat shift.ksh
#!/bin/ksh
if [ "#$" == "0" ]
then
  echo You have to give at least one parameter.
  exit 1
fi
while (( $# ))
  do
    echo You gave me $1
    shift
  done
```

Below is some sample output of the script above.

```
kahlan@solexp11$ ./shift.ksh one
You gave me one
kahlan@solexp11$ ./shift.ksh one two three 1201 "33 42"
You gave me one
You gave me two
You gave me three
You gave me 1201
You gave me 33 42
kahlan@solexp11$ ./shift.ksh
You have to give at least one parameter.
```

21.3. runtime input

You can ask the user for input with the \texttt{read} command in a script.

```
#!/bin/bash
echo -n Enter a number:
read number
```
21.4. sourcing a config file

The source (as seen in the shell chapters) can be used to source a configuration file.

Below a sample configuration file for an application.

[paul@RHEL4a scripts]$ cat myApp.conf
# The config file of myApp

# Enter the path here
myAppPath=/var/myApp

# Enter the number of quines here
quines=5

And here an application that uses this file.

[paul@RHEL4a scripts]$ cat myApp.bash
#!/bin/bash
#
# Welcome to the myApp application
#
. ./myApp.conf

echo There are $quines quines

The running application can use the values inside the sourced configuration file.

[paul@RHEL4a scripts]$ ./myApp.bash
There are 5 quines
[paul@RHEL4a scripts]$
21.5. get script options with getopt

The **getopt** function allows you to parse options given to a command. The following script allows for any combination of the options a, f and z.

```bash
kahlan@solexp11$ cat options.ksh
#!/bin/ksh

while getopt ",:afz" option;
do
  case $option in
    a)
      echo received -a
      ;;
    f)
      echo received -f
      ;;
    z)
      echo received -z
      ;;
    *)
      echo "invalid option -$OPTARG"
      ;;
  esac
done

This is sample output from the script above. First we use correct options, then we enter twice an invalid option.

kahlan@solexp11$ ./options.ksh
kahlan@solexp11$ ./options.ksh -af
received -a
received -f
kahlan@solexp11$ ./options.ksh -zfg
received -z
received -f
invalid option -g
kahlan@solexp11$ ./options.ksh -a -b -z
received -a
invalid option -b
received -z
```
You can also check for options that need an argument, as this example shows.

```
kahlan@solexp11$ cat argoptions.ksh
#!/bin/ksh
while getopts ":af:z" option;
    do
case $option in
        a)
            echo received -a
            ;;
f)
            echo received -f with $OPTARG
            ;;
z)
            echo received -z
            ;;
    esac
    esac
esac
done
```

This is sample output from the script above.

```
kahlan@solexp11$ ./argoptions.ksh -a -f hello -z
received -a
received -f with hello
received -z
kahlan@solexp11$ ./argoptions.ksh -zaf 42
received -z
received -a
received -f with 42
kahlan@solexp11$ ./argoptions.ksh -zf
received -z
option -f needs an argument
```

**21.6. get shell options with shopt**

You can toggle the values of variables controlling optional shell behaviour with the `shopt` built-in shell command. The example below first verifies whether the cdspell option is set; it is not. The next shopt command sets the value, and the third shopt command verifies that the option really is set. You can now use minor spelling mistakes in the cd command. The man page of bash has a complete list of options.

```
paul@laika:~$ shopt -q cdspell ; echo $?
1
paul@laika:~$ shopt -s cdspell
paul@laika:~$ shopt -q cdspell ; echo $?
0
paul@laika:~$ cd /Etc
/etc
```
21.7. practice: parameters and options

1. Write a script that receives four parameters, and outputs them in reverse order.

2. Write a script that receives two parameters (two filenames) and outputs whether those files exist.

3. Write a script that asks for a filename. Verify existence of the file, then verify that you own the file, and whether it is writable. If not, then make it writable.

4. Make a configuration file for the previous script. Put a logging switch in the config file, logging means writing detailed output of everything the script does to a log file in /tmp.
21.8. solution: parameters and options

1. Write a script that receives four parameters, and outputs them in reverse order.

   ```bash
   echo $4 $3 $2 $1
   ```

2. Write a script that receives two parameters (two filenames) and outputs whether those files exist.

   ```bash
   #!/bin/bash
   
   if [ -f $1 ]
   then echo $1 exists!
   else echo $1 not found!
   fi
   
   if [ -f $2 ]
   then echo $2 exists!
   else echo $2 not found!
   fi
   ```

3. Write a script that asks for a filename. Verify existence of the file, then verify that you own the file, and whether it is writable. If not, then make it writable.

4. Make a configuration file for the previous script. Put a logging switch in the config file, logging means writing detailed output of everything the script does to a log file in /tmp.
Chapter 22. more scripting

Table of Contents

22.1. eval ................................................................. 179
22.2. (( )) ................................................................. 179
22.3. let ................................................................. 180
22.4. case ................................................................. 181
22.5. shell functions .................................................. 182
22.6. practice : more scripting .................................... 183
22.7. solution : more scripting .................................... 184
22.1. eval

**eval** reads arguments as input to the shell (the resulting commands are executed). This allows using the value of a variable as a variable.

```
paul@deb503:~/test42$ answer=42
paul@deb503:~/test42$ word=answer
paul@deb503:~/test42$ eval x=\$$word ; echo $x
42
```

Both in **bash** and **Korn** the arguments can be quoted.

```
kahlan@solexp11$ answer=42
kahlan@solexp11$ word=answer
kahlan@solexp11$ eval "y=\$$word" ; echo $y
42
```

Sometimes the **eval** is needed to have correct parsing of arguments. Consider this example where the **date** command receives one parameter **1 week ago**.

```
paul@debian6~$ date --date="1 week ago"
Thu Mar  8 21:36:25 CET 2012

When we set this command in a variable, then executing that variable fails unless we use **eval**.

```
paul@debian6~$ lastweek='date --date="1 week ago"'
paul@debian6~$ $lastweek
date: extra operand 'ago''
Try `date --help' for more information.
paul@debian6~$ eval $lastweek
Thu Mar  8 21:36:39 CET 2012
```

22.2. (( ))

The (( ))) allows for evaluation of numerical expressions.

```
paul@deb503:~/test42$ (( 42 > 33 )) && echo true || echo false
true
paul@deb503:~/test42$ (( 42 > 1201 )) && echo true || echo false
false
paul@deb503:~/test42$ var42=42
paul@deb503:~/test42$ (( 42 == var42 )) && echo true || echo false
true
paul@deb503:~/test42$ (( 42 == $var42 )) && echo true || echo false
true
paul@deb503:~/test42$ var42=33
paul@deb503:~/test42$ (( 42 == var42 )) && echo true || echo false
false
paul@deb503:~/test42$ (( 42 == $var42 )) && echo true || echo false
false
```
22.3. let

The **let** built-in shell function instructs the shell to perform an evaluation of arithmetic expressions. It will return 0 unless the last arithmetic expression evaluates to 0.

```bash
[paul@RHEL4b ~]$ let x="3 + 4" ; echo $x
7
[paul@RHEL4b ~]$ let x="10 + 100/10" ; echo $x
20
[paul@RHEL4b ~]$ let x="10-2+100/10" ; echo $x
18
[paul@RHEL4b ~]$ let x="10*2+100/10" ; echo $x
30
```

The **shell** can also convert between different bases.

```bash
[paul@RHEL4b ~]$ let x="0xFF" ; echo $x
255
[paul@RHEL4b ~]$ let x="0xC0" ; echo $x
192
[paul@RHEL4b ~]$ let x="0xA8" ; echo $x
168
[paul@RHEL4b ~]$ let x="8#70" ; echo $x
56
[paul@RHEL4b ~]$ let x="8#77" ; echo $x
63
[paul@RHEL4b ~]$ let x="16#c0" ; echo $x
192
```

There is a difference between assigning a variable directly, or using **let** to evaluate the arithmetic expressions (even if it is just assigning a value).

```bash
kahlan@solexp11$ dec=15 ; oct=017 ; hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 017 0x0f
kahlan@solexp11$ let dec=15 ; let oct=017 ; let hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 15 15
```
22.4. case

You can sometimes simplify nested if statements with a case construct.

[paul@RHEL4b ~]$ ./help
What animal did you see ? lion
You better start running fast!
[paul@RHEL4b ~]$ ./help
What animal did you see ? dog
Don't worry, give it a cookie.
[paul@RHEL4b ~]$ cat help
#!/bin/bash
#
# Wild Animals Helpdesk Advice
#
# echo -n "What animal did you see ? 
read animal
case $animal in
  "lion" | "tiger")
    echo "You better start running fast!"
  ;;
  "cat")
    echo "Let that mouse go..."
  ;;
  "dog")
    echo "Don't worry, give it a cookie."
  ;;
  "chicken" | "goose" | "duck"
    echo "Eggs for breakfast!"
  ;;
  "liger")
    echo "Approach and say 'Ah you big fluffy kitty...'."
  ;;
  "babelfish")
    echo "Did it fall out your ear ?"
  ;;
  *
    echo "You discovered an unknown animal, name it!"
  ;;
esac
[paul@RHEL4b ~]$
22.5. shell functions

Shell functions can be used to group commands in a logical way.

```bash
kahlan@solexp11$ cat funcs.ksh
#!/bin/ksh

function greetings {
  echo Hello World!
  echo and hello to $USER to!
}

echo We will now call a function
greetings
echo The end

This is sample output from this script with a function.

kahlan@solexp11$ ./funcs.ksh
We will now call a function
Hello World!
and hello to kahlan to!
The end

A shell function can also receive parameters.

kahlan@solexp11$ cat addfunc.ksh
#!/bin/ksh

function plus {
  let result="$1 + $2"
  echo $1 + $2 = $result
}

plus 3 10
plus 20 13
plus 20 22

This script produces the following output.

kahlan@solexp11$ ./addfunc.ksh
3 + 10 = 13
20 + 13 = 33
20 + 22 = 42
```
22.6. practice : more scripting

1. Write a script that asks for two numbers, and outputs the sum and product (as shown here).

   Enter a number: 5
   Enter another number: 2

   Sum:      \( 5 + 2 = 7 \)
   Product: \( 5 \times 2 = 10 \)

2. Improve the previous script to test that the numbers are between 1 and 100, exit with an error if necessary.

3. Improve the previous script to congratulate the user if the sum equals the product.

4. Write a script with a case insensitive case statement, using the shopt nocasematch option. The nocasematch option is reset to the value it had before the scripts started.

5. If time permits (or if you are waiting for other students to finish this practice), take a look at linux system scripts in /etc/init.d and /etc/rc.d and try to understand them. Where does execution of a script start in /etc/init.d/samba? There are also some hidden scripts in ~, we will discuss them later.
22.7. solution : more scripting

1. Write a script that asks for two numbers, and outputs the sum and product (as shown here).

Enter a number: 5  
Enter another number: 2  

Sum: 5 + 2 = 7  
Product: 5 x 2 = 10

#!/bin/bash

echo -n "Enter a number : 
read n1

echo -n "Enter another number : 
read n2

let sum="n1+n2"
let pro="n1*n2"

echo -e "Sum\t: $n1 + $n2 = $sum"
echo -e "Product\t: $n1 * $n2 = $pro"

2. Improve the previous script to test that the numbers are between 1 and 100, exit with an error if necessary.

echo -n "Enter a number between 1 and 100 : 
read n1

if [ $n1 -lt 1 -o $n1 -gt 100 ]
then
    echo Wrong number...
    exit 1
fi

3. Improve the previous script to congratulate the user if the sum equals the product.

if [ $sum -eq $pro ]
then echo Congratulations $sum == $pro
fi

4. Write a script with a case insensitive case statement, using the shopt nocasematch option. The nocasematch option is reset to the value it had before the scripts started.

#!/bin/bash

# Wild Animals Case Insensitive Helpdesk Advice

if shopt -q nocasematch; then
    nocase=yes;
else
    nocase=no;
    shopt -s nocasematch;
fi

echo -n "What animal did you see ? 
read animal
more scripting

case $animal in
  "lion" | "tiger")
    echo "You better start running fast!"
  ;;
  "cat")
    echo "Let that mouse go..."
  ;;
  "dog")
    echo "Don't worry, give it a cookie."
  ;;
  "chicken" | "goose" | "duck" )
    echo "Eggs for breakfast!"
  ;;
  "liger")
    echo "Approach and say 'Ah you big fluffy kitty.'"
  ;;
  "babelfish")
    echo "Did it fall out your ear ?"
  ;;
  *)
    echo "You discovered an unknown animal, name it!"
  ;;
esac

if [ nocase = yes ] ; then
  shopt -s nocasematch;
else
  shopt -u nocasematch;
fi

5. If time permits (or if you are waiting for other students to finish this practice),
take a look at linux system scripts in /etc/init.d and /etc/rc.d and try to understand
them. Where does execution of a script start in /etc/init.d/samba ? There are also some
hidden scripts in ~, we will discuss them later.
Part VII. local user management
Chapter 23. users

Table of Contents

23.1. identify yourself ................................................................. 188
23.2. users ................................................................................. 189
23.3. passwords ......................................................................... 191
23.4. home directories .............................................................. 196
23.5. user shell ................................................................. 197
23.6. switch users with su ........................................................ 198
23.7. run a program as another user ........................................ 199
23.8. practice: users ............................................................... 201
23.9. solution: users ................................................................. 202
23.10. shell environment ...................................................... 204
23.1. identify yourself

whoami

The **whoami** command tells you your username.

```
[root@RHEL5 ~]# whoami
root
[root@RHEL5 ~]# su - paul
[paul@RHEL5 ~]$ whoami
paul
```

who

The **who** command will give you information about who is logged on the system.

```
[paul@RHEL5 ~]$ who
root     tty1         2008-06-24 13:24
sandra   pts/0        2008-06-24 14:05 (192.168.1.34)
paul     pts/1        2008-06-24 16:23 (192.168.1.37)
```

who am i

With **who am i** the who command will display only the line pointing to your current session.

```
[paul@RHEL5 ~]$ who am i
paul     pts/1        2008-06-24 16:23 (192.168.1.34)
```

w

The **w** command shows you who is logged on and what they are doing.

```
$ w
 05:13:36 up 3 min, 4 users, load average: 0.48, 0.72, 0.33
 USER   TTY     FROM       LOGIN@   IDLE    JCPU    PCPU WHAT
root   tty1   -           05:11    2.00s   0.32s   0.27s find / -name shadi
inge   pts/0  192.168.1.33 05:12    0.00s   0.02s   0.02s -ksh
paul   pts/2  192.168.1.34 05:13   25.00s   0.07s   0.04s top
```

id

The **id** command will give you your user id, primary group id, and a list of the groups that you belong to.

```
root@laika:~# id
uid=0(root) gid=0(root) groups=0(root)
root@laika:~# su - brel
brel@laika:~$ id
uid=1001(brel) gid=1001(brel) groups=1001(brel),1008(chanson),11578(wolf)
```
23.2. users

user management

User management on any Unix can be done in three complimentary ways. You can use the graphical tools provided by your distribution. These tools have a look and feel that depends on the distribution. If you are a novice Linux user on your home system, then use the graphical tool that is provided by your distribution. This will make sure that you do not run into problems.

Another option is to use command line tools like useradd, usermod, gpasswd, passwd and others. Server administrators are likely to use these tools, since they are familiar and very similar across many different distributions. This chapter will focus on these command line tools.

A third and rather extremist way is to edit the local configuration files directly using vi (or vipw/vigr). Do not attempt this as a novice on production systems!

/etc/passwd

The local user database on Linux (and on most Unixes) is /etc/passwd.

```
[inge@RHEL5 ~]# tail /etc/passwd
inge:x:518:524:art dealer:/home/inge:/bin/ksh
ann:x:519:525:flute player:/home/ann:/bin/bash
frederik:x:520:526:rubius poet:/home/frederik:/bin/bash
steven:x:521:527:roman emperor:/home/steven:/bin/bash
pascale:x:522:528:artist:/home/pascale:/bin/ksh
geert:x:524:530:kernel developer:/home/geert:/bin/bash
wilm:x:525:531:master damuti:/home/wilm:/bin/bash
sandra:x:526:532:radish stresser:/home/sandra:/bin/bash
annelies:x:527:533:sword fighter:/home/annelies:/bin/bash
laura:x:528:534:art dealer:/home/laura:/bin/ksh
```

As you can see, this file contains seven columns separated by a colon. The columns contain the username, an x, the user id, the primary group id, a description, the name of the home directory, and the login shell.

root

The root user also called the superuser is the most powerful account on your Linux system. This user can do almost anything, including the creation of other users. The root user always has userid 0 (regardless of the name of the account).

```
[root@RHEL5 ~]# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```
users

useradd

You can add users with the **useradd** command. The example below shows how to add a user named yanina (last parameter) and at the same time forcing the creation of the home directory (-m), setting the name of the home directory (-d), and setting a description (-c).

```
[root@RHEL5 ~]# useradd -m -d /home/yanina -c "yanina wickmayer" yanina
```

```
[root@RHEL5 ~]# tail -1 /etc/passwd
yanina:x:529:529:yanina wickmayer:/home/yanina:/bin/bash
```

The user named yanina received userid 529 and **primary group** id 529.

/etc/default/useradd

Both Red Hat Enterprise Linux and Debian/Ubuntu have a file called **/etc/default/useradd** that contains some default user options. Besides using **cat** to display this file, you can also use **useradd -D**.

```
[root@RHEL4 ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

userdel

You can delete the user yanina with **userdel**. The -r option of userdel will also remove the home directory.

```
[root@RHEL5 ~]# userdel -r yanina
```

usermod

You can modify the properties of a user with the **usermod** command. This example uses **usermod** to change the description of the user harry.

```
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:harry potter:/home/harry:/bin/bash
```

```
[root@RHEL4 ~]# usermod -c 'wizard' harry
```

```
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:wizard:/home/harry:/bin/bash
```
23.3. passwords

passwd

Passwords of users can be set with the `passwd` command. Users will have to provide their old password before twice entering the new one.

```
[harry@RHEL4 ~]$ passwd
Changing password for user harry.
Changing password for harry
(current) UNIX password:
New UNIX password:
BAD PASSWORD: it's WAY too short
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[harry@RHEL4 ~]$
```

As you can see, the passwd tool will do some basic verification to prevent users from using too simple passwords. The root user does not have to follow these rules (there will be a warning though). The root user also does not have to provide the old password before entering the new password twice.

/etc/shadow

User passwords are encrypted and kept in `/etc/shadow`. The `/etc/shadow` file is read only and can only be read by root. We will see in the file permissions section how it is possible for users to change their password. For now, you will have to know that users can change their password with the `/usr/bin/passwd` command.

```
[root@RHEL5 ~]# tail /etc/shadow
inge:$1$yWMSImOV$YaYvKqByFVYlKnU3ncd0:14054:0:99999:7:::
ann:!!:14054:0:99999:7:::
frederik:!!:14054:0:99999:7:::
steven:!!:14054:0:99999:7:::
pascale:!!:14054:0:99999:7:::
geert:!!:14054:0:99999:7:::
wim:!!:14054:0:99999:7:::
sandra:!!:14054:0:99999:7:::
annelies:!!:14054:0:99999:7:::
laura:$1$Tvby1Kpa$LWzg0bujUS3LCl1Rmdv1:14054:0:99999:7:::
```

The `/etc/shadow` file contains nine colon separated columns. The nine fields contain (from left to right) the user name, the encrypted password (note that only inge and laura have an encrypted password), the day the password was last changed (day 1 is January 1, 1970), number of days the password must be left unchanged, password expiry day, warning number of days before password expiry, number of days after expiry before disabling the account, and the day the account was disabled (again, since 1970). The last field has no meaning yet.
password encryption

encryption with passwd

Passwords are stored in an encrypted format. This encryption is done by the crypt function. The easiest (and recommended) way to add a user with a password to the system is to add the user with the useradd -m user command, and then set the user's password with passwd.

[root@RHEL4 ~]# useradd -m xavier
[root@RHEL4 ~]# passwd xavier
Changing password for user xavier.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@RHEL4 ~]#

encryption with openssl

Another way to create users with a password is to use the -p option of useradd, but that option requires an encrypted password. You can generate this encrypted password with the openssl passwd command.

[root@RHEL4 ~]# openssl passwd stargate
ZZNX16QZVgUQg
[root@RHEL4 ~]# useradd -m -p ZZNX16QZVgUQg mohamed

encryption with crypt

A third option is to create your own C program using the crypt function, and compile this into a command.

[paul@laika ~]$ cat MyCrypt.c
#include <stdio.h>
#define __USE_XOPEN
#include <unistd.h>

int main(int argc, char** argv)
{
    if(argc==3)
    {
        printf("%s\n", crypt(argv[1],argv[2]));
    }
    else
    {
        printf("Usage: MyCrypt $password $salt\n");
    }
    return 0;
}
This little program can be compiled with **gcc** like this.

[paul@laika ~]$ gcc MyCrypt.c -o MyCrypt -lcrypt

To use it, we need to give two parameters to MyCrypt. The first is the unencrypted password, the second is the salt. The salt is used to perturb the encryption algorithm in one of 4096 different ways. This variation prevents two users with the same password from having the same entry in `/etc/shadow`.

```
paul@laika:~$ ./MyCrypt stargate 12
12L4FoTS3/k9U
paul@laika:~$ ./MyCrypt stargate 01
01Y.yPnlQ6R.Y
paul@laika:~$ ./MyCrypt stargate 33
330asFUbzgVeg
paul@laika:~$ ./MyCrypt stargate 42
42XFxoT4R75gk
```

Did you notice that the first two characters of the password are the **salt**?

The standard output of the crypt function is using the DES algorithm which is old and can be cracked in minutes. A better method is to use **md5** passwords which can be recognized by a salt starting with $\$1$.

```
paul@laika:~$ ./MyCrypt stargate '$1$12'
$1$12$xUIQ4116Us.Q5Osc2Khhm1
paul@laika:~$ ./MyCrypt stargate '$1$01'
$1$01$yNs8brjp4b4TEw.v9/I1J/
paul@laika:~$ ./MyCrypt stargate '$1$33'
$1$33$tLh/Ldy2wskdKAJR.Ph4M0
paul@laika:~$ ./MyCrypt stargate '$1$42'
$1$42$Hb3nvP0KwHSQ7fQm11Y7R.
```

The **md5** salt can be up to eight characters long. The salt is displayed in `/etc/shadow` between the second and third $\$, so never use the password as the salt!

```
paul@laika:~$ ./MyCrypt stargate '$1$stargate'
$1$stargate$qqxoLqiSNVGr5ybMxEVM1
```
password defaults

/etc/login.defs

The /etc/login.defs file contains some default settings for user passwords like password aging and length settings. (You will also find the numerical limits of user ids and group ids and whether or not a home directory should be created by default).

```
[root@RHEL4 ~]# grep -i pass /etc/login.defs
# Password aging controls:
# PASS_MAX_DAYS  Maximum number of days a password may be used.
# PASS_MIN_DAYS  Minimum number of days allowed between password changes.
# PASS_MIN_LEN   Minimum acceptable password length.
# PASS_WARN_AGE  Number of days warning given before a password expires.
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7
```

cchange

The cchange command can be used to set an expiration date for a user account (-E), set a minimum (-m) and maximum (-M) password age, a password expiration date, and set the number of warning days before the password expiration date. Much of this functionality is also available from the passwd command. The -l option of cchange will list these settings for a user.

```
[root@RHEL4 ~]# chage -l harry
Minimum:        0
Maximum:        99999
Warning:        7
Inactive:       -1
Last Change:    Jul 23, 2007
Password Expires: Never
Password Inactive: Never
Account Expires: Never
[root@RHEL4 ~]#
```

disabling a password

Passwords in /etc/shadow cannot begin with an exclamation mark. When the second field in /etc/passwd starts with an exclamation mark, then the password can not be used.

Using this feature is often called locking, disabling, or suspending a user account. Besides vi (or vipw) you can also accomplish this with usermod.

The first line in the next screenshot will disable the password of user harry, making it impossible for harry to authenticate using this password.
The root user (and users with sudo rights on su) still will be able to su to harry (because the password is not needed here). Also note that harry will still be able to login if he has set up passwordless ssh!

You can unlock the account again with usermod -U.

Watch out for tiny differences in the command line options of passwd, usermod, and useradd on different distributions! Verify the local files when using features like "disabling, suspending, or locking" users and passwords!

**editing local files**

If you still want to manually edit the /etc/passwd or /etc/shadow, after knowing these commands for password management, then use vipw instead of vi(m) directly. The vipw tool will do proper locking of the file.

vipw: the password file is busy (/etc/ptmp present)
23.4. home directories

creating home directories

The easiest way to create a home directory is to supply the -m option with `useradd` (it is likely set as a default option on Linux).

A less easy way is to create a home directory manually with `mkdir` which also requires setting the owner and the permissions on the directory with `chmod` and `chown` (both commands are discussed in detail in another chapter).

```
[root@RHEL5 ~]# mkdir /home/laura
[root@RHEL5 ~]# chown laura:laura /home/laura
[root@RHEL5 ~]# chmod 700 /home/laura
[root@RHEL5 ~]# ls -ld /home/laura/
```

/`etc/skel/`

When using `useradd` the -m option, the `/etc/skel/` directory is copied to the newly created home directory. The `/etc/skel/` directory contains some (usually hidden) files that contain profile settings and default values for applications. In this way `/etc/skel/` serves as a default home directory and as a default user profile.

```
[root@RHEL5 ~]# ls -la /etc/skel/
total 48
```

```
drwxr-xr-x 2 root root  4096 Apr  1 00:11 .
drwxr-xr-x 97 root root 12288 Jun 24 15:36 ..
-rw-r--r--  1 root root    24 Jul 12  2006 .bash_logout
-rw-r--r--  1 root root   176 Jul 12  2006 .bash_profile
-rw-r--r--  1 root root   124 Jul 12  2006 .bashrc
```

deleting home directories

The -r option of `userdel` will make sure that the home directory is deleted together with the user account.

```
[root@RHEL5 ~]# ls -ld /home/wim/
```

```
drwx------ 2 wim wim 4096 Jun 24 15:19 /home/wim/
```

```
[root@RHEL5 ~]# userdel -r wim
```

```
[root@RHEL5 ~]# ls -ld /home/wim/
ls: /home/wim/: No such file or directory
```
23.5. user shell

login shell

The /etc/passwd file specifies the login shell for the user. In the screenshot below you can see that user annelies will log in with the /bin/bash shell, and user laura with the /bin/ksh shell.

[root@RHEL5 ~]# tail -2 /etc/passwd
annelies:x:527:533:sword fighter:/home/annelies:/bin/bash
laura:x:528:534:art dealer:/home/laura:/bin/ksh

You can use the usermod command to change the shell for a user.

[root@RHEL5 ~]# usermod -s /bin/bash laura
[root@RHEL5 ~]# tail -1 /etc/passwd
laura:x:528:534:art dealer:/home/laura:/bin/bash

chsh

Users can change their login shell with the chsh command. First, user harry obtains a list of available shells (he could also have done a cat /etc/shells) and then changes his login shell to the Korn shell (/bin/ksh). At the next login, harry will default into ksh instead of bash.

[harry@RHEL4 ~]$ chsh -l
/bin/sh
/bin/bash
/sbin/nologin
/bin/ash
/bin/bash
/bin/ksh
/usr/bin/ksh
/usr/bin/pdksh
/bin/tcsh
/bin/csh
/bin/zsh
[harry@RHEL4 ~]$ chsh -s /bin/ksh
Changing shell for harry.
Password:
Shell changed.
[harry@RHEL4 ~]$
23.6. switch users with su

**su to another user**

The `su` command allows a user to run a shell as another user.

```
[paul@RHEL4b ~]$ su harry
Password:
[harry@RHEL4b paul]$
```

**su to root**

Yes you can also `su` to become `root`, when you know the `root` password.

```
[harry@RHEL4b paul]$ su root
Password:
[root@RHEL4b paul]#
```

**su as root**

Unless you are logged in as `root`, running a shell as another user requires that you know the password of that user. The `root` user can become any user without knowing the user's password.

```
[root@RHEL4b paul]# su serena
[serena@RHEL4b paul]$
```

**su - $username**

By default, the `su` command maintains the same shell environment. To become another user and also get the target user's environment, issue the `su -` command followed by the target username.

```
[paul@RHEL4b ~]$ su - harry
Password:
[harry@RHEL4b ~]$
```

**su -**

When no username is provided to `su` or `su -`, the command will assume `root` is the target.

```
[harry@RHEL4b ~]$ su -
Password:
[root@RHEL4b ~]#
```
23.7. run a program as another user

about sudo

The sudo program allows a user to start a program with the credentials of another user. Before this works, the system administrator has to set up the /etc/sudoers file. This can be useful to delegate administrative tasks to another user (without giving the root password).

The screenshot below shows the usage of sudo. User paul received the right to run useradd with the credentials of root. This allows paul to create new users on the system without becoming root and without knowing the root password.

```
paul@laika:~$ useradd -m inge
useradd: unable to lock password file
paul@laika:~$ sudo useradd -m inge
[sudo] password for paul:
paul@laika:~$
```

setuid on sudo

The sudo binary has the setuid bit set, so any user can run it with the effective userid of root.

```
paul@laika:~$ ls -l `which sudo`
-rwsr-xr-x 2 root root 107872 2008-05-15 02:41 /usr/bin/sudo
paul@laika:~$
```

visudo

Check the man page of visudo before playing with the /etc/sudoers file.

sudo su

On some linux systems like Ubuntu and Kubuntu, the root user does not have a password set. This means that it is not possible to login as root (extra security). To perform tasks as root, the first user is given all sudo rights via the /etc/sudoers. In fact all users that are members of the admin group can use sudo to run all commands as root.

```
root@laika:~# grep admin /etc/sudoers
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```
The end result of this is that the user can type `sudo su -` and become root without having to enter the root password. The `sudo` command does require you to enter your own password. Thus the password prompt in the screenshot below is for `sudo`, not for `su`.

```
paul@laika:~$ sudo su -
Password:
root@laika:~#
```
23.8. practice: users

1. Create the users Serena Williams, Venus Williams and Justine Henin, all of them with password set to stargate, with username (lower case!) as their first name, and their full name in the comment. Verify that the users and their home directory are properly created.

2. Create a user called kornuser, give him the Korn shell (/bin/ksh) as his default shell. Log on with this user (on a command line or in a tty).

3. Create a user named einstime without home directory, give him /bin/date as his default logon shell. What happens when you log on with this user? Can you think of a useful real world example for changing a user's login shell to an application?

4. Try the commands who, whoami, who am i, w, id, echo $USER $UID.

5a. Lock the venus user account with usermod.

5b. Use passwd -d to disable the serena password. Verify the serena line in /etc/shadow before and after disabling.

5c. What is the difference between locking a user account and disabling a user account's password?

6. As root change the password of einstime to stargate.

7. Now try changing the password of serena to serena as serena.

8. Make sure every new user needs to change his password every 10 days.

9. Set the warning number of days to four for the kornuser.

10a. Set the password of two separate users to stargate. Look at the encrypted stargate's in /etc/shadow and explain.

10b. Take a backup as root of /etc/shadow. Use vi to copy an encrypted stargate to another user. Can this other user now log on with stargate as a password?

11. Put a file in the skeleton directory and check whether it is copied to user's home directory. When is the skeleton directory copied?

12. Why use vipw instead of vi? What could be the problem when using vi or vim?

13. Use chsh to list all shells, and compare to cat /etc/shells. Change your login shell to the Korn shell, log out and back in. Now change back to bash.

14. Which useradd option allows you to name a home directory?

15. How can you see whether the password of user harry is locked or unlocked? Give a solution with grep and a solution with passwd.
23.9. solution: users

1. Create the users Serena Williams, Venus Williams and Justine Henin, all of them with password set to stargate, with username (lower case) as their first name, and their full name in the comment. Verify that the users and their home directory are properly created.

```
useradd -m -c "Serena Williams" serena ; passwd serena
useradd -m -c "Venus Williams" venus ; passwd venus
useradd -m -c "Justine Henin" justine ; passwd justine
tail /etc/passwd ; tail /etc/shadow ; ls /home
```

Keep user logon names in lowercase!

2. Create a user called kornuser, give him the Korn shell (/bin/ksh) as his default shell. Log on with this user (on a command line or in a tty).

```
useradd -s /bin/ksh kornuser ; passwd kornuser
```

3. Create a user named einstime without home directory, give him /bin/date as his default logon shell. What happens when you log on with this user? Can you think of a useful real world example for changing a user's login shell to an application?

```
useradd -s /bin/date einstime ; passwd einstime
```

It can be useful when users need to access only one application on the server. Just logging on opens the application for them, and closing the application automatically logs them off.

4. Try the commands who, whoami, who am i, w, id, echo $USER $UID.

```
who ; whoami ; who am i ; w ; id ; echo $USER $UID
```

5a. Lock the venus user account with usermod.

```
usermod -L venus
```

5b. Use passwd -d to disable the serena password. Verify the serena line in /etc/shadow before and after disabling.

```
grep serena /etc/shadow; passwd -d serena ; grep serena /etc/shadow
```

5c. What is the difference between locking a user account and disabling a user account's password?

Locking will prevent the user from logging on to the system with his password (by putting a ! in front of the password in /etc/shadow). Disabling with passwd will erase the password from /etc/shadow.

6. As root change the password of einstime to stargate.

```
Log on as root and type: passwd einstime
```

7. Now try changing the password of serena to serena as serena.

```
log on as serena, then execute: passwd serena... it should fail!
```
8. Make sure every new user needs to change his password every 10 days.

For an existing user: chage -M 10 serena
For all new users: vi /etc/login.defs (and change PASS_MAX_DAYS to 10)

9. Set the warning number of days to four for the kornuser.

chage -W 4 kornuser

10a. Set the password of two separate users to stargate. Look at the encrypted stargate's in /etc/shadow and explain.

If you used passwd, then the salt will be different for the two encrypted passwords.

10b. Take a backup as root of /etc/shadow. Use vi to copy an encrypted stargate to another user. Can this other user now log on with stargate as a password?

Yes.

11. Put a file in the skeleton directory and check whether it is copied to user's home directory. When is the skeleton directory copied?

When you create a user account with a new home directory.

12. Why use vipw instead of vi? What could be the problem when using vi or vim?

vipw will give a warning when someone else is already using that file.

13. Use chsh to list all shells, and compare to cat /etc/shells. Change your login shell to the Korn shell, log out and back in. Now change back to bash.

On Red Hat Enterprise Linux: chsh -l
On Debian/Ubuntu: cat /etc/shells

14. Which useradd option allows you to name a home directory?

-d

15. How can you see whether the password of user harry is locked or unlocked? Give a solution with grep and a solution with passwd.

grep harry /etc/shadow
passwd -S harry
23.10. shell environment

It is nice to have these preset and custom aliases and variables, but where do they all come from? The shell uses a number of startup files that are checked (and executed) whenever the shell is invoked. What follows is an overview of startup scripts.

/etc/profile

Both the bash and the ksh shell will verify the existence of /etc/profile and execute it if it exists.

When reading this script, you might notice (at least on Debian Lenny and on Red Hat Enterprise Linux 5) that it builds the PATH environment variable. The script might also change the PS1 variable, set the HOSTNAME and execute even more scripts like /etc/inputrc

You can use this script to set aliases and variables for every user on the system.

~/.bash_profile

When this file exists in the users home directory, then bash will execute it. On Debian Linux it does not exist by default.

RHEL5 uses a brief ~/.bash_profile where it checks for the existence of ~/.bashrc and then executes it. It also adds $HOME/bin to the $PATH variable.

[serena@rhel53 ~]$ cat .bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
  . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
export PATH

~/.bash_login

When .bash_profile does not exist, then bash will check for ~/.bash_login and execute it.

Neither Debian nor Red Hat have this file by default.
~/.profile

When neither ~/.bash_profile and ~/.bash_login exist, then bash will verify the existence of ~/.profile and execute it. This file does not exist by default on Red Hat.

On Debian this script can execute ~/.bashrc and will add $HOME/bin to the $PATH variable.

```
serena@deb503:~$ tail -12 .profile
# if running bash
if [ -n "$BASH_VERSION" ]; then
  # include .bashrc if it exists
  if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
  fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ]; then
  PATH="$HOME/bin:$PATH"
fi
```

~/.bashrc

As seen in the previous points, the ~/.bashrc script might be executed by other scripts. Let us take a look at what it does by default.

Red Hat uses a very simple ~/.bashrc, checking for /etc/bashrc and executing it. It also leaves room for custom aliases and functions.

```
[serena@rhel53 ~]$ more .bashrc
#.bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
  /etc/bashrc
fi

# User specific aliases and functions
```

On Debian this script is quite a bit longer and configures $PS1, some history variables and a number of active and inactive aliases.

```
serena@deb503:~$ ls -l .bashrc
-rw-r--r-- 1 serena serena 3116 2008-05-12 21:02 .bashrc
```
~/.bash_logout

When exiting bash, it can execute ~/.bash_logout. Debian and Red Hat both use this opportunity to clear the screen.

```
serena@deb503:~$ cat .bash_logout
# ~/.bash_logout: executed by bash(1) when login shell exits.
# when leaving the console clear the screen to increase privacy
if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
```

```
[serena@rhel53 ~]$ cat .bash_logout
# ~/.bash_logout
/usr/bin/clear
```

Debian overview

Below is a table overview of when Debian is running any of these bash startup scripts.

<table>
<thead>
<tr>
<th>script</th>
<th>su</th>
<th>su -</th>
<th>ssh</th>
<th>gdm</th>
</tr>
</thead>
<tbody>
<tr>
<td>~/.bashrc</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>~/.profile</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>/etc/profile</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>/etc/bash.bashrc</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

RHEL5 overview

Below is a table overview of when Red Hat Enterprise Linux 5 is running any of these bash startup scripts.

<table>
<thead>
<tr>
<th>script</th>
<th>su</th>
<th>su -</th>
<th>ssh</th>
<th>gdm</th>
</tr>
</thead>
<tbody>
<tr>
<td>~/.bashrc</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>~/.bash_profile</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>/etc/profile</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>/etc/bashrc</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Chapter 24. groups

Table of Contents

24.1. about groups ................................................................. 208
24.2. groupadd ................................................................. 208
24.3. /etc/group ................................................................. 208
24.4. usermod ................................................................. 209
24.5. groupmod ................................................................. 209
24.6. groupdel ................................................................. 209
24.7. groups ................................................................. 209
24.8. gpasswd ................................................................. 210
24.9. vigr ................................................................. 210
24.10. practice: groups ...................................................... 211
24.11. solution: groups ...................................................... 212
24.1. about groups

Users can be listed in groups. Groups allow you to set permissions on the group level instead of having to set permissions for every individual user. Every Unix or Linux distribution will have a graphical tool to manage groups. Novice users are advised to use this graphical tool. More experienced users can use command line tools to manage users, but be careful: Some distributions do not allow the mixed use of GUI and CLI tools to manage groups (YaST in Novell Suse). Senior administrators can edit the relevant files directly with vi or vigr.

24.2. groupadd

Groups can be created with the groupadd command. The example below shows the creation of five (empty) groups.

    root@laika:~# groupadd tennis
    root@laika:~# groupadd football
    root@laika:~# groupadd snooker
    root@laika:~# groupadd formula1
    root@laika:~# groupadd salsa

24.3. /etc/group

Users can be a member of several groups. Group membership is defined by the /etc/group file.

    root@laika:~# tail -5 /etc/group
    tennis:x:1006:
    football:x:1007:
    snooker:x:1008:
    formula1:x:1009:
    salsa:x:1010:
    root@laika:~#

The first field is the group's name. The second field is the group's (encrypted) password (can be empty). The third field is the group identification or GID. The fourth field is the list of members, these groups have no members.
24.4. usermod

Group membership can be modified with the useradd or usermod command.

```
root@laika:~# usermod -a -G tennis inge
root@laika:~# usermod -a -G tennis katrien
root@laika:~# usermod -a -G salsa katrien
root@laika:~# usermod -a -G snooker sandra
root@laika:~# usermod -a -G formula1 annelies
root@laika:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
snooker:x:1008:sandra
formula1:x:1009:annelies
salsa:x:1010:katrien
root@laika:~#
```

Be careful when using usermod to add users to groups. By default, the usermod command will remove the user from every group of which he is a member if the group is not listed in the command! Using the -a (append) switch prevents this behaviour.

24.5. groupmod

You can change the group name with the groupmod command.

```
root@laika:~# groupmod -n darts snooker
root@laika:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
snooker:x:1008:sandra
formula1:x:1009:annelies
salsa:x:1010:katrien
darts:x:1008:sandra
```

24.6. groupdel

You can permanently remove a group with the groupdel command.

```
root@laika:~# groupdel tennis
root@laika:~#
```

24.7. groups

A user can type the groups command to see a list of groups where the user belongs to.

```
[harry@RHEL4b ~]$ groups
harry sports
[harry@RHEL4b ~]$
```
24.8. gpasswd

You can delegate control of group membership to another user with the gpasswd command. In the example below we delegate permissions to add and remove group members to serena for the sports group. Then we su to serena and add harry to the sports group.

```
[root@RHEL4b ~]# gpasswd -A serena sports
[root@RHEL4b ~]# su - serena
[serena@RHEL4b ~]$ id harry
uid=516(harry) gid=520(harry) groups=520(harry)
[serena@RHEL4b ~]$ gpasswd -a harry sports
Adding user harry to group sports
[serena@RHEL4b ~]$ id harry
uid=516(harry) gid=520(harry) groups=520(harry),522(sports)
[serena@RHEL4b ~]$ tail -1 /etc/group
sports:x:522:serena,venus,harry
[serena@RHEL4b ~]$ 
```

Group administrators do not have to be a member of the group. They can remove themselves from a group, but this does not influence their ability to add or remove members.

```
[serena@RHEL4b ~]$ gpasswd -d serena sports
Removing user serena from group sports
[serena@RHEL4b ~]$ exit
```

Information about group administrators is kept in the /etc/gshadow file.

```
[root@RHEL4b ~]# tail -1 /etc/gshadow
sports::!::serena:venus,harry
[root@RHEL4b ~]#
```

To remove all group administrators from a group, use the gpasswd command to set an empty administrators list.

```
[root@RHEL4b ~]# gpasswd -A "" sports
```

24.9. vigr

Similar to vipw, the vigr command can be used to manually edit the /etc/group file, since it will do proper locking of the file. Only experienced senior administrators should use vi or vigr to manage groups.
24.10. practice: groups

1. Create the groups tennis, football and sports.

2. In one command, make venus a member of tennis and sports.

3. Rename the football group to foot.

4. Use vi to add serena to the tennis group.

5. Use the id command to verify that serena is a member of tennis.

6. Make someone responsible for managing group membership of foot and sports. Test that it works.
24.11. solution: groups

1. Create the groups tennis, football and sports.

   groupadd tennis ; groupadd football ; groupadd sports

2. In one command, make venus a member of tennis and sports.

   usermod -a -G tennis,sports venus

3. Rename the football group to foot.

   groupmod -n foot football

4. Use vi to add serena to the tennis group.

   vi /etc/group

5. Use the id command to verify that serena is a member of tennis.

   id (and after logoff logon serena should be member)

6. Make someone responsible for managing group membership of foot and sports.
   Test that it works.

   gpasswd -A (to make manager)
   gpasswd -a (to add member)
Part VIII. file security
Chapter 25. standard file permissions

Table of Contents

25.1. file ownership ............................................................................................. 215
25.2. list of special files ...................................................................................... 216
25.3. permissions ................................................................................................. 217
25.4. practice: standard file permissions ............................................................. 222
25.5. solution: standard file permissions ............................................................. 223
25.1. file ownership

user owner and group owner

The users and groups of a system can be locally managed in `/etc/passwd` and `/etc/group`, or they can be in a NIS, LDAP, or Samba domain. These users and groups can own files. Actually, every file has a user owner and a group owner, as can be seen in the following screenshot.

```
paul@RHELv4u4:~/test$ ls -l
 total 24
 -rw-rw-r-- 1 paul paul 17 Feb  7 11:53 file1
 -rw-rw-r-- 1 paul paul 106 Feb  5 17:04 file2
 -rw-r-wr-- 1 paul proj 984 Feb  5 15:38 data.odt
 -rw-r--r-- 1 root root 984 Feb  5 15:38 stuff.txt
paul@RHELv4u4:~/test$
```

User paul owns three files, two of those are also owned by the group paul; data.odt is owned by the group proj. The root user owns the file stuff.txt, as does the group root.

chgrp

You can change the group owner of a file using the `chgrp` command.

```
root@laika:/home/paul# touch FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chgrp paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
```

chown

The user owner of a file can be changed with `chown` command.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
```

You can also use `chown` to change both the user owner and the group owner.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown root:project42 FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root project42 0 2008-08-06 14:11 FileForPaul
```
25.2. list of special files

When you use `ls -l`, for each file you can see ten characters before the user and group owner. The first character tells us the type of file. Regular files get a -, directories get a d, symbolic links are shown with an l, pipes get a p, character devices a c, block devices a b, and sockets an s.

**Table 25.1. Unix special files**

<table>
<thead>
<tr>
<th>first character</th>
<th>file type</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>normal file</td>
</tr>
<tr>
<td>d</td>
<td>directory</td>
</tr>
<tr>
<td>l</td>
<td>symbolic link</td>
</tr>
<tr>
<td>p</td>
<td>named pipe</td>
</tr>
<tr>
<td>b</td>
<td>block device</td>
</tr>
<tr>
<td>c</td>
<td>character device</td>
</tr>
<tr>
<td>s</td>
<td>socket</td>
</tr>
</tbody>
</table>

Below a screenshot of a character device (the console) and a block device (the hard disk).

```
paul@debian6lt~$ ls -ld /dev/console /dev/sda
crw-------   1 root root  5, 1 Mar 15 12:45 /dev/console
brw-rw----   1 root disk  8, 0 Mar 15 12:45 /dev/sda
```

And here you can see a directory, a regular file and a symbolic link.

```
paul@debian6lt~$ ls -ld /etc /etc/hosts /etc/motd
drwxr-xr-x 128 root root 12288 Mar 15 18:34 /etc
-rw-r--r-- 1 root root 372 Dec 10 17:36 /etc/hosts
lrwxrwxrwx 1 root root 13 Dec  5 10:36 /etc/motd -> /var/run/motd
```
25.3. permissions

rwx

The nine characters following the file type denote the permissions in three triplets. A permission can be r for read access, w for write access, and x for execute. You need the r permission to list (ls) the contents of a directory. You need the x permission to enter (cd) a directory. You need the w permission to create files in or remove files from a directory.

Table 25.2. standard Unix file permissions

<table>
<thead>
<tr>
<th>permission</th>
<th>on a file</th>
<th>on a directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>r (read)</td>
<td>read file contents (cat)</td>
<td>read directory contents (ls)</td>
</tr>
<tr>
<td>w (write)</td>
<td>change file contents (vi)</td>
<td>create files in (touch)</td>
</tr>
<tr>
<td>x (execute)</td>
<td>execute the file</td>
<td>enter the directory (cd)</td>
</tr>
</tbody>
</table>

three sets of rwx

We already know that the output of `ls -l` starts with ten characters for each file. This screenshot shows a regular file (because the first character is a - ).

```
paul@RHELv4u4:~/test$ ls -l proc42.bash
-rwxr-xr--  1 paul proj 984 Feb  6 12:01 proc42.bash
```

Below is a table describing the function of all ten characters.

Table 25.3. Unix file permissions position

<table>
<thead>
<tr>
<th>position</th>
<th>characters</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>this is a regular file</td>
</tr>
<tr>
<td>2-4</td>
<td>rwx</td>
<td>permissions for the <strong>user owner</strong></td>
</tr>
<tr>
<td>5-7</td>
<td>r-x</td>
<td>permissions for the <strong>group owner</strong></td>
</tr>
<tr>
<td>8-10</td>
<td>r--</td>
<td>permissions for <strong>others</strong></td>
</tr>
</tbody>
</table>

When you are the **user owner** of a file, then the **user owner permissions** apply to you. The rest of the permissions have no influence on your access to the file.

When you belong to the **group** that is the **group owner** of a file, then the **group owner permissions** apply to you. The rest of the permissions have no influence on your access to the file.

When you are not the **user owner** of a file and you do not belong to the **group owner**, then the **others permissions** apply to you. The rest of the permissions have no influence on your access to the file.
permission examples

Some example combinations on files and directories are seen in this screenshot. The name of the file explains the permissions.

```
paul@laika:~/perms$ ls -lh
total 12K
drwxr-xr-x 2 paul paul 4.0K 2007-02-07 22:26 AllEnter_UserCreateDelete
-rwxrwxrw- 1 paul paul 0 2007-02-07 22:21 EveryoneFullControl.txt
-r--r----- 1 paul paul 0 2007-02-07 22:21 OnlyOwnersRead.txt
-rwxrwx--- 1 paul paul 0 2007-02-07 22:21 OwnersAll_RestNothing.txt
dr-xr-x--- 2 paul paul 4.0K 2007-02-07 22:25 UserAndGroupEnter
dr-x------ 2 paul paul 4.0K 2007-02-07 22:25 OnlyUserEnter
paul@laika:~/perms$
```

To summarise, the first `rwx` triplet represents the permissions for the **user owner**. The second triplet corresponds to the **group owner**; it specifies permissions for all members of that group. The third triplet defines permissions for all **other** users that are not the user owner and are not a member of the group owner.

setting permissions (chmod)

Permissions can be changed with `chmod`. The first example gives the user owner execute permissions.

```
paul@laika:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod u+x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the group owners read permission.

```
paul@laika:~/perms$ chmod g-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx---r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the others read permission.

```
paul@laika:~/perms$ chmod o-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx------ 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example gives all of them the write permission.

```
paul@laika:~/perms$ chmod a+w permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```
You don't even have to type the a.

    paul@laika:~/perms$ chmod +x permissions.txt
    paul@laika:~/perms$ ls -l permissions.txt
    -rwx--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt

You can also set explicit permissions.

    paul@laika:~/perms$ chmod u=rw permissions.txt
    paul@laika:~/perms$ ls -l permissions.txt
    -rw--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt

Feel free to make any kind of combination.

    paul@laika:~/perms$ chmod u=rw,g=rx,o=r permissions.txt
    paul@laika:~/perms$ ls -l permissions.txt
    -rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt

Even fishy combinations are accepted by chmod.

    paul@laika:~/perms$ chmod u=rwx,ug+rw,o=r permissions.txt
    paul@laika:~/perms$ ls -l permissions.txt
    -rwxrw=r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
setting octal permissions

Most Unix administrators will use the old school octal system to talk about and set permissions. Look at the triplet bitwise, equating r to 4, w to 2, and x to 1.

Table 25.4. Octal permissions

<table>
<thead>
<tr>
<th>binary</th>
<th>octal</th>
<th>permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>---</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>--x</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>-w-</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>-wx</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>r--</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>r-x</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>rw-</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>rwx</td>
</tr>
</tbody>
</table>

This makes 777 equal to rwxrwxrwx and by the same logic, 654 mean rw-r-xr--. The chmod command will accept these numbers.

```
paul@laika:~/perms$ chmod 777 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 664 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 750 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```
standard file permissions

umask

When creating a file or directory, a set of default permissions are applied. These default permissions are determined by the **umask**. The **umask** specifies permissions that you do not want set on by default. You can display the **umask** with the **umask** command.

```
[Harry@RHEL4b ~]$ umask
0002
[Harry@RHEL4b ~]$ touch test
[Harry@RHEL4b ~]$ ls -l test
-rw-rw-r--  1 Harry Harry 0 Jul 24 06:03 test
[Harry@RHEL4b ~]$
```

As you can also see, the file is also not executable by default. This is a general security feature among Unixes; newly created files are never executable by default. You have to explicitly do a **chmod +x** to make a file executable. This also means that the 1 bit in the **umask** has no meaning—a **umask** of 0022 is the same as 0033.

**mkdir -m**

When creating directories with **mkdir** you can use the **-m** option to set the **mode**. This screenshot explains.

```
paul@debian5~$ mkdir -m 700 MyDir
paul@debian5~$ mkdir -m 777 Public
paul@debian5~$ ls -dl MyDir/ Public/
```

```
``drwx------ 2 paul paul 4096 2011-10-16 19:16 MyDir/
drwxrwxrwx 2 paul paul 4096 2011-10-16 19:16 Public/
```
25.4. practice: standard file permissions

1. As normal user, create a directory ~/permissions. Create a file owned by yourself in there.

2. Copy a file owned by root from /etc/ to your permissions dir, who owns this file now ?

3. As root, create a file in the users ~/permissions directory.

4. As normal user, look at who owns this file created by root.

5. Change the ownership of all files in ~/permissions to yourself.

6. Make sure you have all rights to these files, and others can only read.

7. With chmod, is 770 the same as rwxrwx--- ?

8. With chmod, is 664 the same as r-xr-xr-- ?

9. With chmod, is 400 the same as r-------- ?

10. With chmod, is 734 the same as rwxr-xr-- ?

11a. Display the umask in octal and in symbolic form.

11b. Set the umask to 077, but use the symbolic format to set it. Verify that this works.

12. Create a file as root, give only read to others. Can a normal user read this file ? Test writing to this file with vi.

13a. Create a file as normal user, give only read to others. Can another normal user read this file ? Test writing to this file with vi.

13b. Can root read this file ? Can root write to this file with vi ?

14. Create a directory that belongs to a group, where every member of that group can read and write to files, and create files. Make sure that people can only delete their own files.
25.5. solution: standard file permissions

1. As normal user, create a directory ~/permissions. Create a file owned by yourself in there.
   
   mkdir ~/permissions; touch ~/permissions/myfile.txt

2. Copy a file owned by root from /etc/ to your permissions dir, who owns this file now?
   
   cp /etc/hosts ~/permissions/

   The copy is owned by you.

3. As root, create a file in the users ~/permissions directory.
   
   (become root)# touch /home/username/permissions/rootfile

4. As normal user, look at who owns this file created by root.
   
   ls -l ~/permissions

   The file created by root is owned by root.

5. Change the ownership of all files in ~/permissions to yourself.
   
   chown user ~/permissions/*

   You cannot become owner of the file that belongs to root.

6. Make sure you have all rights to these files, and others can only read.
   
   chmod 644 (on files)
   chmod 755 (on directories)

7. With chmod, is 770 the same as rwxrwx--- ?
   
   yes

8. With chmod, is 664 the same as r-xr-xr-- ?
   
   No

9. With chmod, is 400 the same as r-------- ?
   
   yes

10. With chmod, is 734 the same as rwxr-xr-- ?

    no

11a. Display the umask in octal and in symbolic form.

    umask; umask -S

---

223
11b. Set the umask to 077, but use the symbolic format to set it. Verify that this works.

   umask -S u=rwx,go=

12. Create a file as root, give only read to others. Can a normal user read this file? Test writing to this file with vi.

   (become root)
   # echo hello > /home/username/root.txt
   # chmod 744 /home/username/root.txt
   (become user)
   vi ~/root.txt

13a. Create a file as normal user, give only read to others. Can another normal user read this file? Test writing to this file with vi.

   echo hello > file ; chmod 744 file

   Yes, others can read this file

13b. Can root read this file? Can root write to this file with vi?

   Yes, root can read and write to this file. Permissions do not apply to root.

14. Create a directory that belongs to a group, where every member of that group can read and write to files, and create files. Make sure that people can only delete their own files.

   mkdir /home/project42 ; groupadd project42
   chgrp project42 /home/project42 ; chmod 775 /home/project42

   You can not yet do the last part of this exercise...
Chapter 26. advanced file permissions

Table of Contents

26.1. sticky bit on directory ................................................................. 226
26.2. setgid bit on directory ............................................................... 226
26.3. setgid and setuid on regular files ............................................. 227
26.4. practice: sticky, setuid and setgid bits ...................................... 228
26.5. solution: sticky, setuid and setgid bits ...................................... 229
advanced file permissions

26.1. sticky bit on directory

You can set the **sticky bit** on a directory to prevent users from removing files that they do not own as a user owner. The sticky bit is displayed at the same location as the x permission for others. The sticky bit is represented by a t (meaning x is also there) or a T (when there is no x for others).

```
root@RHELv4u4:~# mkdir /project55
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-x  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~# chmod +t /project55/
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-t  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~#
```

The **sticky bit** can also be set with octal permissions, it is binary 1 in the first of four triplets.

```
root@RHELv4u4:~# chmod 1775 /project55/
root@RHELv4u4:~# ls -ld /project55
drwxrwxr-t  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~#
```

You will typically find the **sticky bit** on the /tmp directory.

```
root@barry:~# ls -ld /tmp
drwxrwxrwt 6 root root 4096 2009-06-04 19:02 /tmp
```

26.2. setgid bit on directory

**setgid** can be used on directories to make sure that all files inside the directory are owned by the group owner of the directory. The **setgid** bit is displayed at the same location as the x permission for group owner. The **setgid** bit is represented by an s (meaning x is also there) or a S (when there is no x for the group owner). As this example shows, even though root does not belong to the group proj55, the files created by root in /project55 will belong to proj55 since the **setgid** is set.

```
root@RHELv4u4:~# groupadd proj55
root@RHELv4u4:~# chown root:proj55 /project55/
root@RHELv4u4:~# chmod 2775 /project55/
root@RHELv4u4:~# touch /project55/fromroot.txt
root@RHELv4u4:~# ls -ld /project55/
```

```
root@RHELv4u4:~# drwxrwsr-x  2 root proj55 4096 Feb  7 17:45 /project55/
```

```
root@RHELv4u4:~# touch /project55/fromroot.txt
root@RHELv4u4:~# ls -1 /project55/
```

```
total 4
-rw-r--r--  1 root proj55  0 Feb  7 17:45 fromroot.txt
```

You can use the **find** command to find all **setgid** directories.

```
paul@laika:~$ find / -type d -perm -2000 2> /dev/null
/var/log/mysql
/var/log/news
/var/local
...
26.3. setgid and setuid on regular files

These two permissions cause an executable file to be executed with the permissions of the file owner instead of the executing owner. This means that if any user executes a program that belongs to the root user, and the setuid bit is set on that program, then the program runs as root. This can be dangerous, but sometimes this is good for security.

Take the example of passwords; they are stored in /etc/shadow which is only readable by root. (The root user never needs permissions anyway.)

```
root@RHELv4u4:~# ls -l /etc/shadow
-r--------  1 root root 1260 Jan 21 07:49 /etc/shadow
```

Changing your password requires an update of this file, so how can normal non-root users do this? Let's take a look at the permissions on the /usr/bin/passwd.

```
root@RHELv4u4:~# ls -l /usr/bin/passwd
-r-s--x--x  1 root root 21200 Jun 17  2005 /usr/bin/passwd
```

When running the passwd program, you are executing it with root credentials.

You can use the find command to find all setuid programs.

```
paul@laika:~$ find /usr/bin -type f -perm -04000
/usr/bin/arping
/usr/bin/kgrantpty
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/sudo
/usr/bin/fping6
/usr/bin/passwd
/usr/bin/gpasswd
...
```

In most cases, setting the setuid bit on executables is sufficient. Setting the setgid bit will result in these programs to run with the credentials of their group owner.
26.4. practice: sticky, setuid and setgid bits

1a. Set up a directory, owned by the group sports.

1b. Members of the sports group should be able to create files in this directory.

1c. All files created in this directory should be group-owned by the sports group.

1d. Users should be able to delete only their own user-owned files.

1e. Test that this works!

2. Verify the permissions on /usr/bin/passwd. Remove the setuid, then try changing your password as a normal user. Reset the permissions back and try again.

3. If time permits (or if you are waiting for other students to finish this practice), read about file attributes in the man page of chattr and lsattr. Try setting the i attribute on a file and test that it works.
26.5. solution: sticky, setuid and setgid bits

1a. Set up a directory, owned by the group sports.

```bash
groupadd sports
mkdir /home/sports
chown root:sports /home/sports
```

1b. Members of the sports group should be able to create files in this directory.

```bash
chmod 770 /home/sports
```

1c. All files created in this directory should be group-owned by the sports group.

```bash
chmod 2770 /home/sports
```

1d. Users should be able to delete only their own user-owned files.

```bash
chmod +t /home/sports
```

1e. Test that this works!

Log in with different users (group members and others and root), create files and watch the permissions. Try changing and deleting files...

2. Verify the permissions on `/usr/bin/passwd`. Remove the setuid, then try changing your password as a normal user. Reset the permissions back and try again.

```bash
root@deb503:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
root@deb503:~# chmod 755 /usr/bin/passwd
root@deb503:~# ls -l /usr/bin/passwd
-rwxr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd

A normal user cannot change password now.

```bash
root@deb503:~# chmod 4755 /usr/bin/passwd
root@deb503:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
```

3. If time permits (or if you are waiting for other students to finish this practice), read about file attributes in the man page of chattr and lsattr. Try setting the i attribute on a file and test that it works.

```bash
paul@laika:~$ sudo su -
[sudo] password for paul:
root@laika:~# mkdir attr
root@laika:~/attr# touch file42
root@laika:~/attr# lsattr
------------------ ./file42
root@laika:~/attr# chattr +i file42
```
root@laika:~/attr# lsattr
----i------------- ./file42
root@laika:~/attr# rm -rf file42
rm: cannot remove `file42': Operation not permitted
root@laika:~/attr# chattr -i file42
root@laika:~/attr# rm -rf file42
root@laika:~/attr#
Chapter 27. access control lists

Table of Contents

27.1. acl in /etc/fstab ................................................................. 232
27.2. getfacl ................................................................. 232
27.3. setfacl ................................................................. 232
27.4. remove an acl entry ......................................................... 233
27.5. remove the complete acl ..................................................... 233
27.6. the acl mask ................................................................. 233
27.7. eiciel ................................................................. 234

Standard Unix permissions might not be enough for some organisations. This chapter introduces access control lists or acl's to further protect files and directories.
27.1. acl in /etc/fstab

File systems that support access control lists, or acls, have to be mounted with the acl option listed in /etc/fstab. In the example below, you can see that the root file system has acl support, whereas /home/data does not.

```
root@laika:~# tail -4 /etc/fstab
/dev/sda1        /              ext3     acl,relatime    0  1
/dev/sdb2        /home/data     auto     noacl,defaults  0  0
pasha:/home/r    /home/pasha    nfs      defaults        0  0
wolf:/srv/data   /home/wolf     nfs      defaults        0  0
```

27.2. getfacl

Reading acls can be done with /usr/bin/getfacl. This screenshot shows how to read the acl of file33 with getfacl.

```
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
mask::rwx
other::r--
```

27.3. setfacl

Writing or changing acls can be done with /usr/bin/setfacl. These screenshots show how to change the acl of file33 with setfacl.

First we add user sandra with octal permission 7 to the acl.

```
paul@laika:~/test$ setfacl -m u:sandra:7 file33
```

Then we add the group tennis with octal permission 6 to the acl of the same file.

```
paul@laika:~/test$ setfacl -m g:tennis:6 file33
```

The result is visible with getfacl.

```
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
user:sandra:rwx
group::r--
group:tennis:rwx
mask::rwx
other::r--
```
27.4. remove an acl entry

The -x option of the `setfacl` command will remove an acl entry from the targeted file.

```
paul@laika:~/test$ setfacl -m u:sandra:7 file33
paul@laika:~/test$ getfacl file33 | grep sandra
user:sandra:rwx
paul@laika:~/test$ setfacl -x sandra file33
paul@laika:~/test$ getfacl file33 | grep sandra
```

Note that omitting the u or g when defining the acl for an account will default it to a user account.

27.5. remove the complete acl

The -b option of the `setfacl` command will remove the acl from the targeted file.

```
paul@laika:~/test$ setfacl -b file33
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
other::r--
```

27.6. the acl mask

The acl mask defines the maximum effective permissions for any entry in the acl. This mask is calculated every time you execute the `setfacl` or `chmod` commands.

You can prevent the calculation by using the --no-mask switch.

```
paul@laika:~/test$ setfacl --no-mask -m u:sandra:7 file33
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
user:sandra:rwx   #effective:rw-
group::r--
mask::rw-
other::r--
```
27.7. eiciel

Desktop users might want to use eiciel to manage acls with a graphical tool.

You will need to install eiciel and nautilus-actions to have an extra tab in nautilus to manage acls.

paul@laika:~$ sudo aptitude install eiciel nautilus-actions
Chapter 28. file links

Table of Contents

28.1. inodes ......................................................................................................... 236
28.2. about directories ......................................................................................... 237
28.3. hard links ................................................................................................... 238
28.4. symbolic links ............................................................................................ 239
28.5. removing links ........................................................................................... 239
28.6. practice : links ............................................................................................ 240
28.7. solution : links ............................................................................................ 241

An average computer using Linux has a file system with many hard links and symbolic links.

To understand links in a file system, you first have to understand what an inode is.
28.1. inodes

inode contents

An inode is a data structure that contains metadata about a file. When the file system stores a new file on the hard disk, it stores not only the contents (data) of the file, but also extra properties like the name of the file, the creation date, its permissions, the owner of the file, and more. All this information (except the name of the file and the contents of the file) is stored in the inode of the file.

The `ls -l` command will display some of the inode contents, as seen in this screenshot.

```
root@rhel53 ~# ls -ld /home/project42/
drwxr-xr-x 4 root pro42 4.0K Mar 27 14:29 /home/project42/
```

inode table

The inode table contains all of the inodes and is created when you create the file system (with `mkfs`). You can use the `df -i` command to see how many inodes are used and free on mounted file systems.

```
root@rhel53 ~# df -i
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/mapper/VolGroup00-LogVol100 4947968 115326 4832642  3% /
/dev/hda1            26104     45 26059     1% /boot
/tmpfs               64417      1 64416     1% /dev/shm
/dev/sda1           262144 2207 260937     1% /home/project42
/dev/sdb1           74400  5519 73841     0% /home/project33
/dev/sdb5            0      0    0   - /home/sales
/dev/sdb6          100744  11 100733     1% /home/research
```

In the `df -i` screenshot above you can see the inode usage for several mounted file systems. You don't see numbers for `/dev/sdb5` because it is a fat file system.

inode number

Each inode has a unique number (the inode number). You can see the inode numbers with the `ls -li` command.

```
paul@RHELv4u4:~/test$ touch file1
paul@RHELv4u4:~/test$ touch file2
paul@RHELv4u4:~/test$ touch file3
paul@RHELv4u4:~/test$ ls -li
```

```
total 12
817266 -rw-rw-r-- 1 paul paul 0 Feb  5 15:38 file1
817267 -rw-rw-r-- 1 paul paul 0 Feb  5 15:38 file2
817268 -rw-rw-r-- 1 paul paul 0 Feb  5 15:38 file3
paul@RHELv4u4:~/test$
```
These three files were created one after the other and got three different **inodes** (the first column). All the information you see with this `ls` command resides in the **inode**, except for the filename (which is contained in the directory).

### Inode and File Contents

Let's put some data in one of the files.

```
paul@RHELv4u4:~/test$ ls -li
total 16
817266 -rw-rw-r--  1 paul paul  0 Feb  5 15:38 file1
817270 -rw-rw-r--  1 paul paul 92 Feb  5 15:42 file2
817268 -rw-rw-r--  1 paul paul  0 Feb  5 15:38 file3
paul@RHELv4u4:~/test$ cat file2
It is winter now and it is very cold.
We do not like the cold, we prefer hot summer nights.
paul@RHELv4u4:~/test$
```

The data that is displayed by the `cat` command is not in the **inode**, but somewhere else on the disk. The **inode** contains a pointer to that data.

### 28.2. About Directories

#### A Directory Is a Table

A **directory** is a special kind of file that contains a table which maps filenames to inodes. Listing our current directory with `ls -ali` will display the contents of the directory file.

```
paul@RHELv4u4:~/test$ ls -ali
total 32
817262 drwxrwxr-x   2 paul paul 4096 Feb  5 15:42 .
800768 drwx------  16 paul paul 4096 Feb  5 15:42 ..
817266 -rw-rw-r--   1 paul paul    0 Feb  5 15:38 file1
817270 -rw-rw-r--   1 paul paul   92 Feb  5 15:42 file2
817268 -rw-rw-r--   1 paul paul    0 Feb  5 15:38 file3
paul@RHELv4u4:~/test$
```

. and ..

You can see five names, and the mapping to their five inodes. The dot . is a mapping to itself, and the dotdot .. is a mapping to the parent directory. The three other names are mappings to different inodes.
28.3. hard links

creating hard links

When we create a hard link to a file with ln, an extra entry is added in the directory. A new file name is mapped to an existing inode.

```
paul@RHELv4u4:$ ln file2 hardlink_to_file2
paul@RHELv4u4:$ ls -li
  total 24
  817266  -rw-rw-r-- 1 paul paul  0 Feb  5 15:38 file1
  817270  -rw-rw-r-- 2 paul paul 92 Feb  5 15:42 file2
  817268  -rw-rw-r-- 1 paul paul  0 Feb  5 15:38 file3
  817270  -rw-rw-r-- 2 paul paul 92 Feb  5 15:42 hardlink_to_file2
paul@RHELv4u4:$
```

Both files have the same inode, so they will always have the same permissions and the same owner. Both files will have the same content. Actually, both files are equal now, meaning you can safely remove the original file, the hardlinked file will remain. The inode contains a counter, counting the number of hard links to itself. When the counter drops to zero, then the inode is emptied.

finding hard links

You can use the find command to look for files with a certain inode. The screenshot below shows how to search for all filenames that point to inode 817270. Remember that an inode number is unique to its partition.

```
paul@RHELv4u4:$ find / -inum 817270 2> /dev/null
/home/paul/test/file2
/home/paul/test/hardlink_to_file2
```
28.4. symbolic links

Symbolic links (sometimes called soft links) do not link to inodes, but create a name to name mapping. Symbolic links are created with `ln -s`. As you can see below, the symbolic link gets an inode of its own.

```
paul@RHELv4u4:~/test$ ln -s file2 symlink_to_file2
paul@RHELv4u4:~/test$ ls -li
  total 32
  817273 -rw-rw-r--  1 paul paul 13 Feb  5 17:06 file1
  817270 -rw-rw-r--  2 paul paul 106 Feb  5 17:04 file2
  817268 -rw-rw-r--  1 paul paul   0 Feb  5 15:38 file3
  817270 -rw-rw-r--  2 paul paul 106 Feb  5 17:04 hardlink_to_file2
  817267 lrwxrwxrwx  1 paul paul   5 Feb  5 16:55 symlink_to_file2 -> file2
paul@RHELv4u4:~/test$
```

Permissions on a symbolic link have no meaning, since the permissions of the target apply. Hard links are limited to their own partition (because they point to an inode), symbolic links can link anywhere (other file systems, even networked).

28.5. removing links

Links can be removed with `rm`.

```
paul@laika:~$ touch data.txt
paul@laika:~$ ln -s data.txt sl_data.txt
paul@laika:~$ ln data.txt hl_data.txt
paul@laika:~$ rm sl_data.txt
paul@laika:~$ rm hl_data.txt
```
28.6. practice : links

1. Create two files named winter.txt and summer.txt, put some text in them.

2. Create a hard link to winter.txt named hlwinter.txt.

3. Display the inode numbers of these three files, the hard links should have the same inode.

4. Use the find command to list the two hardlinked files

5. Everything about a file is in the inode, except two things : name them!

6. Create a symbolic link to summer.txt called ssummer.txt.

7. Find all files with inode number 2. What does this information tell you ?

8. Look at the directories /etc/init.d/ /etc/rc.d/ /etc/rc3.d/ ... do you see the links ?

9. Look in /lib with ls -l...

10. Use find to look in your home directory for regular files that do not(!) have one hard link.
28.7. solution : links

1. Create two files named winter.txt and summer.txt, put some text in them.
   
   ```
   echo cold > winter.txt ; echo hot > summer.txt
   ```

2. Create a hard link to winter.txt named hlwinter.txt.
   
   ```
   ln winter.txt hlwinter.txt
   ```

3. Display the inode numbers of these three files, the hard links should have the same inode.
   
   ```
   ls -li winter.txt summer.txt hlwinter.txt
   ```

4. Use the find command to list the two hardlinked files
   
   ```
   find . -inum xyz
   ```

5. Everything about a file is in the inode, except two things : name them!

   The name of the file is in a directory, and the contents is somewhere on the disk.

6. Create a symbolic link to summer.txt called slsummer.txt.
   
   ```
   ln -s summer.txt slsummer.txt
   ```

7. Find all files with inode number 2. What does this information tell you ?

   It tells you there is more than one inode table (one for every formatted partition + virtual file systems)

8. Look at the directories /etc/init.d/ /etc/rc.d/ /etc/rc3.d/ ... do you see the links?

   ```
   ls -l /etc/init.d
   ls -l /etc/rc.d
   ls -l /etc/rc3.d
   ```

9. Look in /lib with ls -l...

   ```
   ls -l /lib
   ```

10. Use find to look in your home directory for regular files that do not(!) have one hard link.

    ```
    find ~ ! -links 1 -type f
    ```
Part IX. Appendices
Appendix A. certifications

A.1. Certification

LPI: Linux Professional Institute

LPIC Level 1

This is the junior level certification. You need to pass exams 101 and 102 to achieve LPIC 1 certification. To pass level one, you will need Linux command line, user management, backup and restore, installation, networking, and basic system administration skills.

LPIC Level 2

This is the advanced level certification. You need to be LPIC 1 certified and pass exams 201 and 202 to achieve LPIC 2 certification. To pass level two, you will need to be able to administer medium sized Linux networks, including Samba, mail, news, proxy, firewall, web, and ftp servers.

LPIC Level 3

This is the senior level certification. It contains one core exam (301) which tests advanced skills mainly about ldap. To achieve this level you also need LPIC Level 2 and pass a specialty exam (302 or 303). Exam 302 mainly focuses on Samba, and 303 on advanced security. More info on http://www.lpi.org.

Ubuntu

When you are LPIC Level 1 certified, you can take a LPI Ubuntu exam (199) and become Ubuntu certified.

Red Hat Certified Engineer

The big difference with most other certifications is that there are no multiple choice questions for RHCE. Red Hat Certified Engineers have to take a live exam consisting of two parts. First, they have to troubleshoot and maintain an existing but broken setup (scoring at least 80 percent), and second they have to install and configure a machine (scoring at least 70 percent).
MySQL

There are two tracks for MySQL certification; Certified MySQL 5.0 Developer (CMDEV) and Certified MySQL 5.0 DBA (CMDBA). The CMDEV is focused towards database application developers, and the CMDBA towards database administrators. Both tracks require two exams each. The MySQL cluster DBA certification requires CMDBA certification and passing the CMCDBA exam.

Novell CLP/CLE

To become a Novell Certified Linux Professional, you have to take a live practicum. This is a VNC session to a set of real SLES servers. You have to perform several tasks and are free to choose your method (commandline or YaST or ...). No multiple choice involved.

Sun Solaris

Sun uses the classical formula of multiple choice exams for certification. Passing two exams for an operating system gets you the Solaris Certified Administrator for Solaris X title.

Other certifications

There are many other lesser known certifications like EC council's Certified Ethical Hacker, CompTIA's Linux+, and Sair's Linux GNU.
Appendix B. keyboard settings

B.1. about keyboard layout

Many people (like US-Americans) prefer the default US-qwerty keyboard layout. So when you are not from the USA and want a local keyboard layout on your system, then the best practice is to select this keyboard at installation time. Then the keyboard layout will always be correct. Also, whenever you use ssh to remotely manage a linux system, your local keyboard layout will be used, independent of the server keyboard configuration. So you will not find much information on changing keyboard layout on the fly on linux, because not many people need it. Below are some tips to help you.

B.2. X Keyboard Layout

This is the relevant portion in /etc/X11/xorg.conf, first for Belgian azerty, then for US-qwerty.

[paul@RHEL5 ~]$ grep -i xkb /etc/X11/xorg.conf
Option      "XkbModel" "pc105"
Option      "XkbLayout" "be"

[paul@RHEL5 ~]$ grep -i xkb /etc/X11/xorg.conf
Option      "XkbModel" "pc105"
Option      "XkbLayout" "us"

When in Gnome or KDE or any other graphical environment, look in the graphical menu in preferences, there will be a keyboard section to choose your layout. Use the graphical menu instead of editing xorg.conf.

B.3. shell keyboard layout

When in bash, take a look in the /etc/sysconfig/keyboard file. Below a sample US-qwerty configuration, followed by a Belgian azerty configuration.

[paul@RHEL5 ~]$ cat /etc/sysconfig/keyboard
KEYBOARDTYPE="pc"
KEYTABLE="us"

[paul@RHEL5 ~]$ cat /etc/sysconfig/keyboard
KEYBOARDTYPE="pc"
KEYTABLE="be-latin1"

The keymaps themselves can be found in /usr/share/keymaps or /lib/kbd/keymaps.
keyboard settings

[paul@RHEL5 ~]$ ls -l /lib/kbd/keymaps/
total 52
  drwxr-xr-x 2 root root 4096 Apr  1 00:14 amiga
  drwxr-xr-x 2 root root 4096 Apr  1 00:14 atari
  drwxr-xr-x 8 root root 4096 Apr  1 00:14 i386
  drwxr-xr-x 2 root root 4096 Apr  1 00:14 include
  drwxr-xr-x 4 root root 4096 Apr  1 00:14 mac
  lrwxrwxrwx 1 root root    3 Apr  1 00:14 ppc -> mac
  drwxr-xr-x 2 root root 4096 Apr  1 00:14 sun
Appendix C. hardware

C.1. buses

about buses

Hardware components communicate with the Central Processing Unit or cpu over a bus. The most common buses today are usb, pci, agp, pci-express and pcmcia aka pc-card. These are all Plag and Play buses.

Older x86 computers often had isa buses, which can be configured using jumpers or dip switches.

/proc/bus

To list the buses recognised by the Linux kernel on your computer, look at the contents of the /proc/bus/ directory (screenshot from Ubuntu 7.04 and RHEL4u4 below).

root@laika:~# ls /proc/bus/
input pccard pci usb

[root@RHEL4b ~]# ls /proc/bus/
inpu	pci usb

Can you guess which of these two screenshots was taken on a laptop?

/usr/sbin/lsusb

To list all the usb devices connected to your system, you could read the contents of /proc/bus/usb/devices (if it exists) or you could use the more readable output of lsusb, which is executed here on a SPARC system with Ubuntu.

root@shaka:~# lsusb
Bus 001 Device 002: ID 0430:0100 Sun Microsystems, Inc. 3-button Mouse
Bus 001 Device 003: ID 0430:0005 Sun Microsystems, Inc. Type 6 Keyboard
Bus 001 Device 001: ID 04b0:0136 Nikon Corp. Coolpix 7900 (storage)
root@shaka:~#

/var/lib/usbutils/usb.ids

The /var/lib/usbutils/usb.ids file contains a gzipped list of all known usb devices.
/usr/sbin/lspci

To get a list of all pci devices connected, you could take a look at /proc/bus/pci or run lspci (partial output below).

paul@laika:~$ lspci
...  
00:06.0 FireWire (IEEE 1394): Texas Instruments TSB43AB22/A IEEE-139...  
00:08.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-816...  
00:09.0 Multimedia controller: Philips Semiconductors SAA7133/SAA713...  
00:0a.0 Network controller: RaLink RT2500 802.11g Cardbus/mini-PCI  
00:0f.0 RAID bus controller: VIA Technologies, Inc. VIA VT6420 SATA ...  
00:0f.1 IDE interface: VIA Technologies, Inc. VT82C586A/B/VT82C686/A...  
00:10.0 USB Controller: VIA Technologies, Inc. VT82xxxxx UHCI USB 1....  
00:10.1 USB Controller: VIA Technologies, Inc. VT82xxxxx UHCI USB 1....  
...  

C.2. interrupts

about interrupts

An interrupt request or IRQ is a request from a device to the CPU. A device raises an interrupt when it requires the attention of the CPU (could be because the device has data ready to be read by the CPU).

Since the introduction of pci, irq's can be shared among devices.

Interrupt 0 is always reserved for the timer, interrupt 1 for the keyboard. IRQ 2 is used as a channel for IRQ's 8 to 15, and thus is the same as IRQ 9.

/proc/interrupts

You can see a listing of interrupts on your system in /proc/interrupts.

paul@laika:~$ cat /proc/interrupts
hardware

You can also use `dmesg` to find irq's allocated at boot time.

```bash
paul@laika:~$ dmesg | grep "irq 1[45]"
[ 28.930069] ata3: PATA max UDMA/133 cmd 0x1f0 ctl 0x3f6 bmdma 0x2090 irq 14
[ 28.930071] ata4: PATA max UDMA/133 cmd 0x170 ctl 0x376 bmdma 0x2098 irq 15
```

C.3. io ports

about io ports

Communication in the other direction, from CPU to device, happens through IO ports. The CPU writes data or control codes to the IO port of the device. But this is not only a one way communication, the CPU can also use a device's IO port to read status information about the device. Unlike interrupts, ports cannot be shared!

`/proc/ioports`

You can see a listing of your system's IO ports via `/proc/ioports`.

```bash
[root@RHEL4b ~]# cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
0090-0091 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
02f8-02ff : serial
...
C.4. dma

about dma

A device that needs a lot of data, interrupts and ports can pose a heavy load on the cpu. With dma or Direct Memory Access a device can gain (temporary) access to a specific range of the ram memory.

/proc/dma

Looking at /proc/dma might not give you the information that you want, since it only contains currently assigned dma channels for isa devices.

root@laika:~# cat /proc/dma
1: parport0
4: cascade

pci devices that are using dma are not listed in /proc/dma, in this case dmesg can be useful. The screenshot below shows that during boot the parallel port received dma channel 1, and the Infrared port received dma channel 3.

root@laika:~# dmesg | egrep -C 1 'dma 1|dma 3'
[ 20.580000] parport0: PC-style at 0x378 (0x778), irq 7, dma 1...
[ 20.764000] irda_init()

[ 21.204000] pnp: Device 00:0b activated.
[ 21.204000] nsc_ircc_pnp_probe() : From PnP, found firbase 0x2F8...
[ 21.204000] nsc-ircc, chip->init
Index

Symbols
; (shell), 84
!! (shell), 101
! (bash history), 101
! (file globbing), 108
? (file globbing), 107
/ 28, 52
/bin, 53, 76
/bin/bash, 73, 204
/bin/cat, 53
/bin/csh, 73
/bin/date, 53
/bin/ksh, 73, 204
/bin/rm, 77
/bin/sh, 73
/boot, 55
/boot/grub, 55
/boot/grub/grub.cfg, 55
/boot/grub/grub.conf, 55
/dev, 36, 59
/dev/null, 59, 117
/dev/pts/1, 59
/dev/random, 70
/dev/tty1, 59
/dev/urandom, 69, 71
/dev/zero, 70
/etc, 55
/etc/bashrc, 205
/etc/default/useradd, 190
/etc/fstab, 232
/etc/group, 208, 215
/etc/gshadow, 210
/etc/hosts, 71
/etc/inputrc, 55
/etc/inputrc, 204
/etc/login.defs, 194
/etc/passwd, 132, 189, 195, 197, 215
/etc/profile, 204
/etc/resolv.conf, 71
/etc/shadow, 191, 193, 227
/etc/shells, 158, 197
/etc/skel, 56, 196
/etc/sudoers, 199, 199
/etc/sysconfig, 56
/etc/sysconfig/firstboot, 56
/etc/sysconfig/harddisks, 56
/etc/sysconfig/hwconf, 56
/etc/sysconfig/keyboard, 56
/etc/X11/xorg.conf, 55
/export, 57
/home, 57
/lib, 54
/lib/kbd/keymaps/, 56
/lib/modules, 54
/lib32, 54
/lib64, 54
/media, 57
/opt, 54
/proc, 36, 60
/proc/bus, 247
/proc/bus/pci, 248
/proc/bus/usb/devices, 247
/proc/cpuinfo, 61
/proc/dma, 250
/proc/interrupts, 62, 248
/proc/ioports, 249
/proc/kcore, 62
/proc/sys, 61
/root, 57
/run, 67
/sbin, 53, 76
/srv, 57
/sys, 63
/tmp, 58, 226
/usr, 64
/usr/bin, 64
/usr/bin/getfacl, 232
/usr/bin/passwd, 227
/usr/bin/setfacl, 232
/usr/include, 64
/usr/lib, 64
/usr/local, 64
/usr/share, 65
/usr/share/games, 65
/usr/share/man, 65
/usr/src, 65
/var, 66
/var/cache, 66
/var/lib, 67
/var/lib/rpm, 67
/var/lib/usbutils/usb.ids, 247
/var/log, 67
/var/log, 66
Index

/var/log/messages, 66
/var/log/syslog, 66
/var/run, 67
/var/spool, 67
/var/tmp, 67
.
.. (directory), 237
. (directory), 237
. (shell), 159
.bash_history, 102
.bash_login, 204
.bash_logout, 206
.bash_profile, 204
.bashrc, 204, 205
.exrc, 153
.vimrc, 153
` (backtick), 96
~, 27
' (single quote), 96
" (double quotes), 75
(( (shell), 179
-- (shell), 160
[ (file globbing), 107
[ (shell), 164
$? (shell variables), 84
$ (shell variables), 90
$HISTFILE, 102
$HISTFILESIZE, 102
$HISTSIZE, 102
$PATH, 76, 91
$PS1, 28
* (file globbing), 107
\ (backslash), 86
&., 84
&&, 85
#!/bin/bash, 158
#! (shell), 158
# (pound sign), 86
>, 115
>>, 116
|, 116
|, 120
||, 85
1>, 117
2>, 117
2>&1, 117
777, 220

A
access control list, 232
acl, 234
acls, 232
agp, 247
AIX, 3
alias/bash), 77
alias/bash), 77
apropos, 23
arguments/bash), 74

B
backticks, 96
base64, 118
bash, 171
bash history, 101
bash -x, 160
binaries, 53
 Bourne again shell, 73
BSD, 3
bunzip2, 141
bus, 247
bzcat, 141
bzip2, 140, 141, 141
bzmore, 141

C
cal, 139
case, 181
case sensitive, 36
cat, 124
cat(1), 46
cd/bash builtin), 27
cd -(bash builtin), 28
CentOS, 5
chage(1), 194
chgrp(1), 215
chkconfig, 56
chmod, 196, 220
chmod(1), 150, 218
chmod +x, 158, 221
chown, 196
chown(1), 215
chsh(1), 197
CMDBA, 244
CMDEV, 244
comm(1), 129
command line scan, 74
command mode(vi), 147
copyleft, 8
copyright, 7, 7
cp(1), 38, 38
cpu, 247
crypt, 192
csh, 158
Ctrl d, 46
ctrl-r, 102
current directory, 27
cut, 132
cut(1), 126

d
daemon, 23
date, 138
Debian, 5
Dennis Ritchie, 3
devfs, 63
df -i, 236
directory, 237
distribution, 4
distributions, 52
dma, 250
dmesg(1), 249, 250
dumpkeys(1), 56

e
echo, 74
echo(1), 74, 75
echo $-, 95
echo *, 109
Edubuntu, 5
eiciel, 234
ELF, 54
elif, 165
embedding(shell), 96
env(1), 93, 93
environment variable, 90
EOF, 118
escaping (shell), 109
eval, 179
executables, 53
exit (bash), 102
export, 93

F
Fedora, 5
FHS, 52
file(1), 36, 54
file globbing, 106
file ownership, 215
Filesystem Hierarchy Standard, 52
filters, 123
find(1), 137, 226, 227, 238
FireWire, 63
for (bash), 165
FOSS, 7
four freedoms, 8
Free Software, 7
free software, 8
freeware, 7
function (shell), 182

G
gcc(1), 193
getfacl, 232
getopts, 174
GID, 208
glob(7), 107
GNU, 3
gpasswd, 210
GPL, 8
GPLv3, 8
grep(1), 124
grep -i, 124
grep -v, 125
groupadd(1), 208
groupdel(1), 209
groupmod(1), 209
groups, 208
groups(1), 209
gunzip(1), 140
gzip, 140
gzip(1), 140

H
hard link, 238
head(1), 45
here directive, 47
here document, 118
here string, 118
hidden files, 29
HP, 3
HP-UX, 3
http://www.pathname.com/fhs/, 52

I
IBM, 3
id(1), 188
IEEE 1394, 63
if then else (bash), 165
inode, 235, 238
inode table, 236
insert mode(vi), 147
interrupt, 248
IO Ports, 249
IRQ, 248
isa, 247

K
Ken Thompson, 3
kernel, 54
keymaps(5), 56
Korn shell, 103
Korn Shell, 197
ksh, 103, 158
kudzu, 56

L
less(1), 48
let, 180
Linus Torvalds, 3
Linux Mint, 5
ln, 239
ln(1), 238
loadkeys(1), 56
locate(1), 138
logical AND, 85
logical OR, 85
Logiciel Libre, 8
LPIC 1 Certification, 243
LPIC 2 Certification, 243
ls, 217, 236
ls(1), 29, 29, 236, 237
ls -l, 216
lsbci, 248
lsusb, 247

M
magic(5), 36
man(1), 23, 24, 24
mandb(1), 25
man hier, 52
man -k, 23
md5, 193
mkdir, 196
mkdir(1), 31, 221
mkdir -p, 31
mkfs, 236
more(1), 48
mv(1), 39
noclobber, 115
nounset(shell), 94
Novell Certified Linux Professional, 244
octal permissions, 220
od(1), 130
OEL, 5
open source, 8
open source definition, 8
open source software, 7
openssl(1), 192
Oracle Enterprise Linux, 5
owner, 217
parent directory, 27
passwd, 194
passwd(1), 24, 191, 191, 192, 227
passwd(5), 24
path, 28, 29
pc-card, 247
pci, 247
pci-express, 247
pcmcia, 247
pipe, 120
popd, 34
primary group, 190
proprietary, 7
public domain, 7
pushd, 34
pwd(1), 27, 28
random number generator, 70
read, 172
reboot, 102
Red Hat, 5
regular expressions, 103
rename(1), 40
repository, 4
RHCE, 243
Richard Stallman, 3
rm(1), 37, 239
rmdir(1), 31
rmdir -p, 31
rm -rf, 38
root, 53, 189, 198, 199, 199
root directory, 52
rpm, 67

S
salt (encryption), 193
Scientific, 5
sed, 131
set, 95
set(shell), 92
set +x, 78
setfacl, 232
setgid, 226, 226
setuid, 160, 199, 227, 227
set -x, 78
she-bang (shell), 158
shell, 204
shell comment, 86
shell escaping, 86
shell expansion, 74, 74
shell functions, 182
shift, 172
shopt, 175
skeleton, 56
sleep, 139
soft link, 239
Solaris, 3
sort, 132
sort(1), 128
source, 159, 173
stderr, 115
stdin, 115, 120, 124
stdout, 115, 120, 124
sticky bit, 226
strings(1), 48
su, 195, 210
su -, 91
su(1), 198, 198
sudo, 195, 199

sudo(1), 199
sudo su -, 200
Sun, 3
SunOS, 3
superuser, 189
symbolic link, 239
sysfs, 63
System V, 54

T
tab key(bash), 29
tac(1), 47
tail(1), 45
tee(1), 124
test, 164
time, 139
touch(1), 37
tr, 127
tr(1), 126
type(shell), 76

U
Ubuntu, 5
umask(1), 221
unalias(bash), 78
uniq, 132
uniq(1), 129
Unix, 3
unset, 95
unset(shell), 92
unset(shell), 92
until (bash), 166
updatedb(1), 138
usb, 63, 247
useradd, 190, 196
useradd(1), 192, 196
useradd -D, 190
userdel(1), 190
usermod, 209
usermod(1), 190, 194, 195

V
vi, 210
vi(1), 146
vigr(1), 210
vim(1), 146
vimtutor(1), 146
vipw(1), 195
visudo(1), 199
vrije software, 8
Index

W
w(1), 188
wc(1), 127
whatis(1), 23
whereis(1), 24
which(1), 76
while (bash), 166
white space(shell), 74
who, 132
who(1), 188
who am i, 188
whoami(1), 188
wild cards, 108

X
X, 55
X Window System, 55

Z
zcat, 140
zmore, 140