

Linux Fun

Paul Cobbaut

Linux Fun

Paul Cobbaut

It-2.0

Published Sun 24 Feb 2013 01:02:25 CET

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.

More information and free .pdf available at <http://linux-training.be> .

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.be, build scripts; infrastructure setup; minor stuff
- Hendrik De Vloed: hendrik.devloed@ugent.be, buildheader.pl script

We'd also like to thank our reviewers:

- Wouter Verhelst: wouter@grep.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://bdesmet.be>
- Rich Yonts: richyonts@gmail.com,

Copyright 2007-2013 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. process management	242
29. introduction to processes	243
30. process priorities	255
31. background jobs	262
X. disk management	268
32. disk devices	269
33. disk partitions	282
34. file systems	290
35. mounting	298
36. introduction to uid's	307

37. introduction to raid	312
38. logical volume management	320
39. iSCSI devices	344
XI. boot management	353
40. bootloader	354
41. init and runlevels	366
XII. system management	382
42. scheduling	383
43. logging	390
44. memory management	401
45. package management	408
XIII. network management	429
46. general networking	430
47. interface configuration	439
48. network sniffing	454
49. binding and bonding	460
50. ssh client and server	469
51. introduction to nfs	481
52. introduction to networking	485
XIV. kernel management	495
53. the Linux kernel	496
54. library management	513
XV. backup management	516
55. backup	517
XVI. Introduction to Samba	526
56. introduction to samba	527
57. getting started with samba	535
58. a read only file server	547
59. a writable file server	554
60. samba first user account	559
61. samba authentication	564
62. samba securing shares	571
63. samba domain member	579
64. samba domain controller	586
65. a brief look at samba 4	596
XVII. dns server	600
66. introduction to DNS	601
67. advanced DNS	625
XVIII. dhcp server	632
68. introduction to dhcp	633
XIX. dhcp server	641
XX. iptables firewall	642
69. introduction to routers	643
70. Firewall: iptables	649
XXI. apache and squid	657
71. introduction to apache	658
72. introduction to squid	664
XXII. introducing git	668

73. git	669
XXIII. ipv6	671
74. Introduction to ipv6	672
XXIV. mysql database	681
75. introduction to sql using mysql	682
XXV. selinux	696
76. introduction to SELinux(draft)	697
XXVI. Appendices	706
A. certifications	707
B. keyboard settings	709
C. hardware	711
D. installing linux	715
E. disk quotas	718
F. introduction to vnc	719
G. cloning	721
H. License	723
Index	730

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
32.1. ide device naming	272
32.2. scsi device naming	272
33.1. primary, extended and logical partitions	283
33.2. Partition naming	283
38.1. disk partitioning example	321
38.2. LVM Example	321
66.1. the first top level domains	606
66.2. new general purpose tld's	606
69.1. Packet Forwarding Exercise	645
69.2. Packet Forwarding Solution	647

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/Richard_Stallman
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

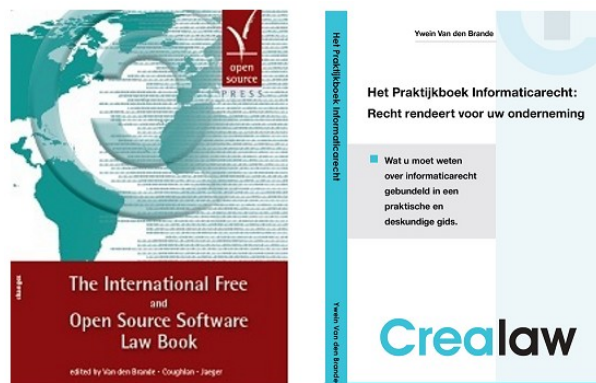
Table of Contents

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 Informatierecht** (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 GPLv. 3 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

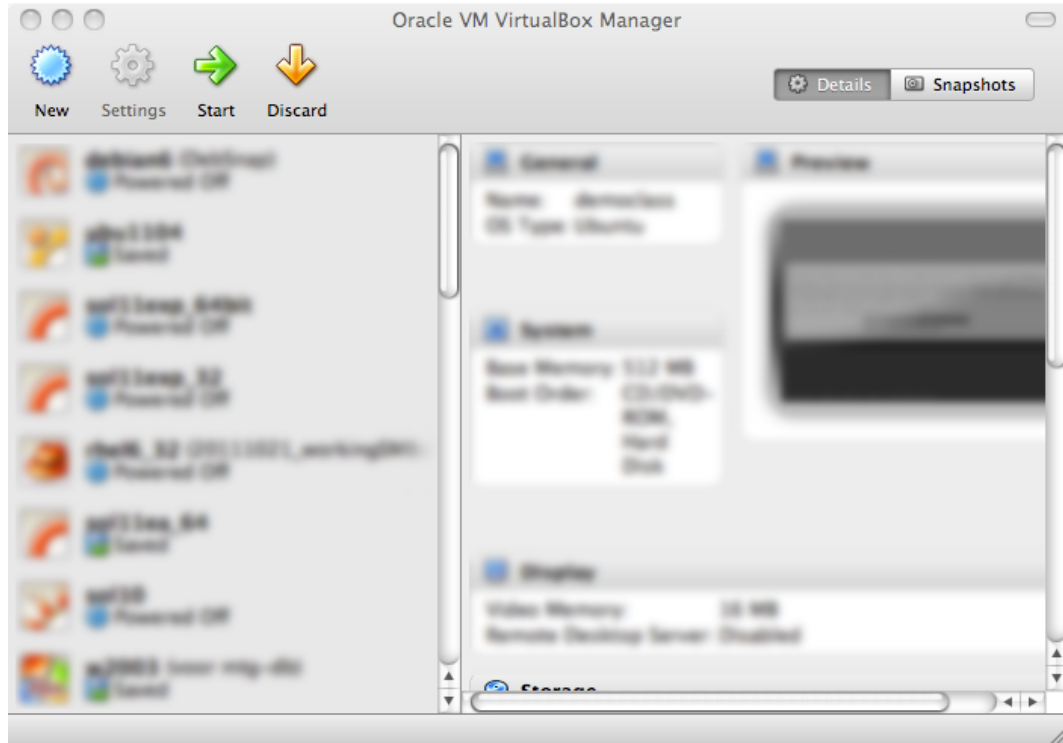
Start by downloading a Linux CD image (an .ISO file) from the distribution of your choice from the Internet. Take care selecting the correct cpu architecture of your computer; choose **i386** if unsure. Choosing the wrong cpu type (like x86_64 when you have an old Pentium) will almost immediately fail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

4.2. download Virtualbox

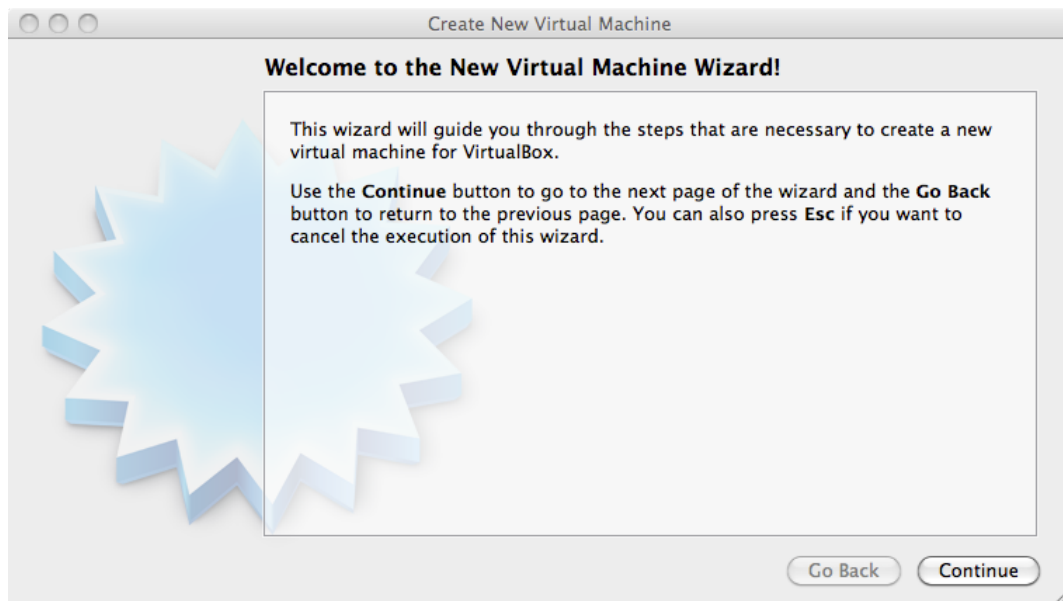
Step two (when the .ISO file has finished downloading) is to download Virtualbox. If you are currently running Microsoft Windows, then download and install Virtualbox for Windows!

4.3. create a virtual machine

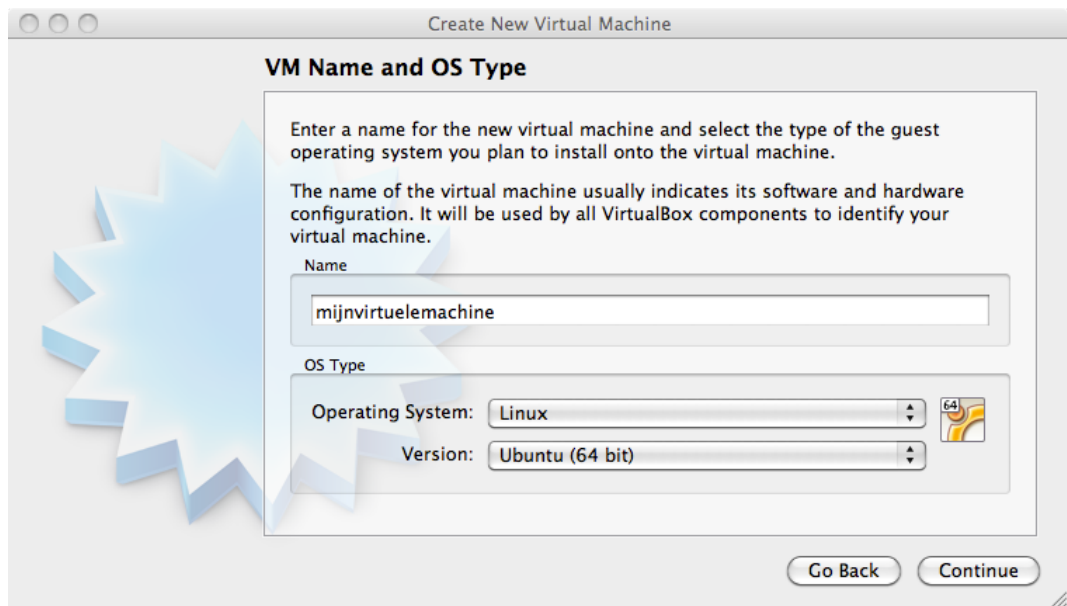
Now start Virtualbox. Contrary to the screenshot below, your left pane should be empty.



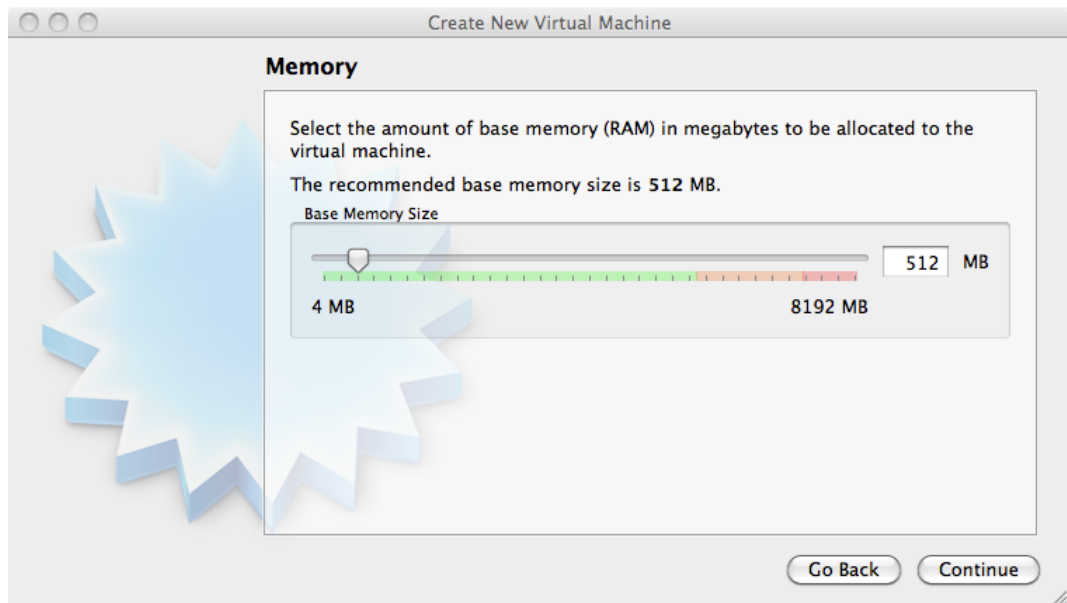
Click **New** to create a new virtual machine. We will walk together through the wizard. The screenshots below are taken on Mac OSX; they will be slightly different if you are running Microsoft Windows.



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).



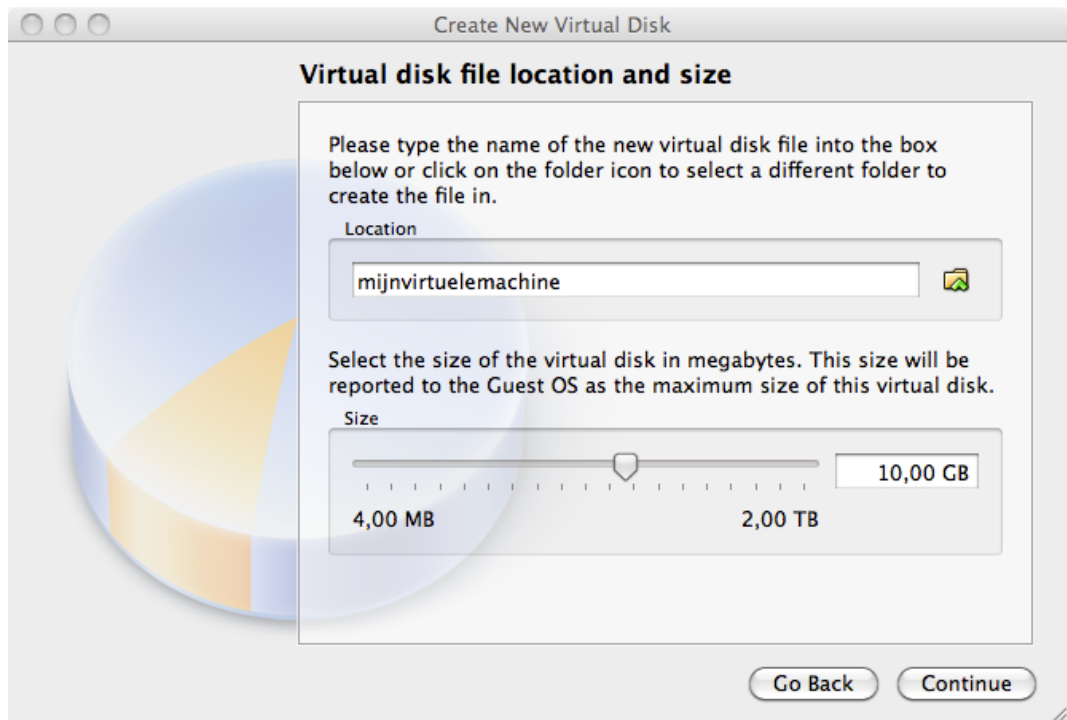
If you get the question below, choose vdi.



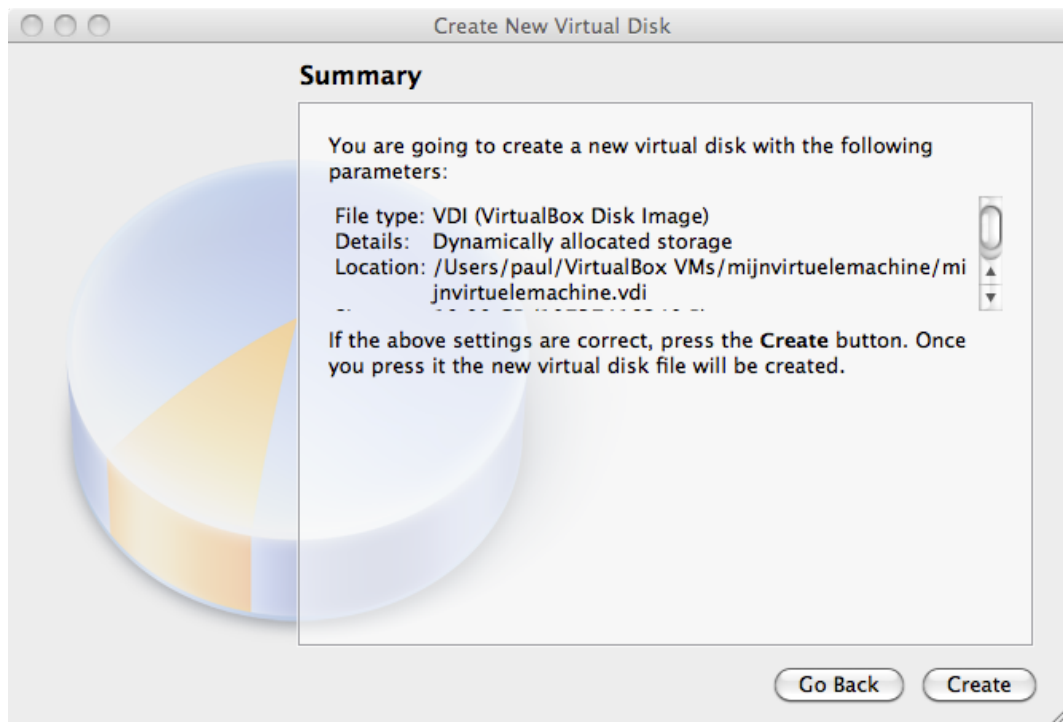
Choose **dynamically allocated** (fixed size is only useful in production or on really old, slow hardware).



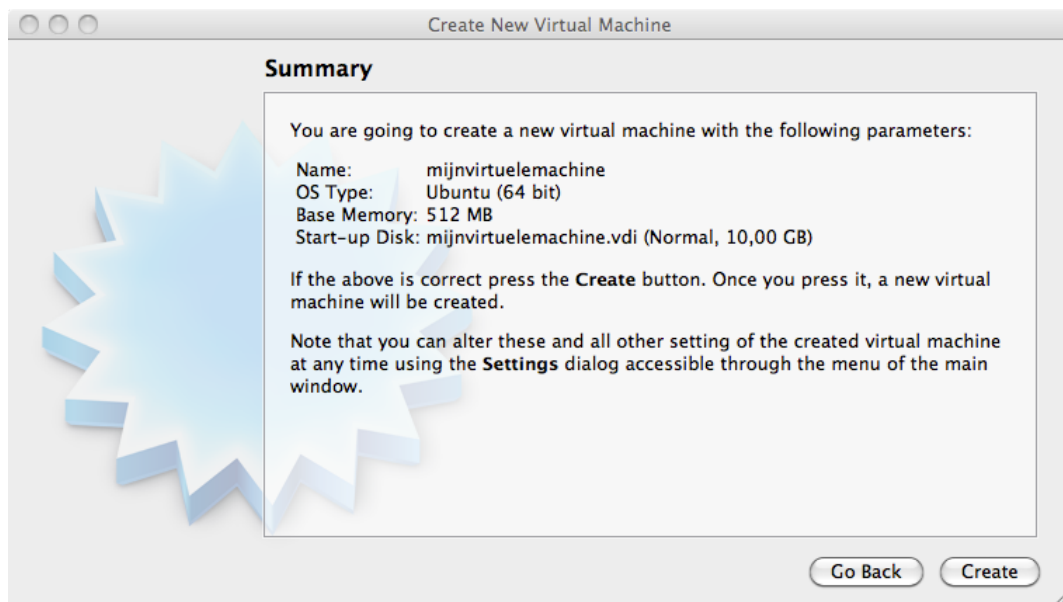
Choose between 10GB and 16GB as the disk size.



Click **create** to create the virtual disk.

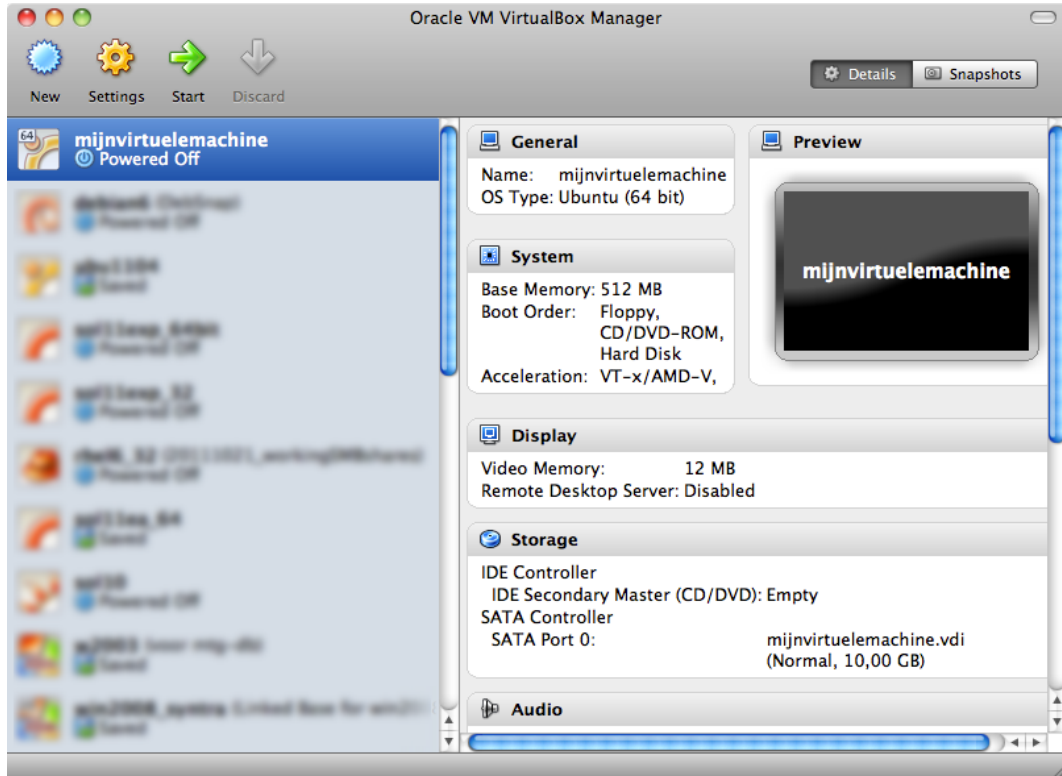


Click **create** to create the virtual machine.

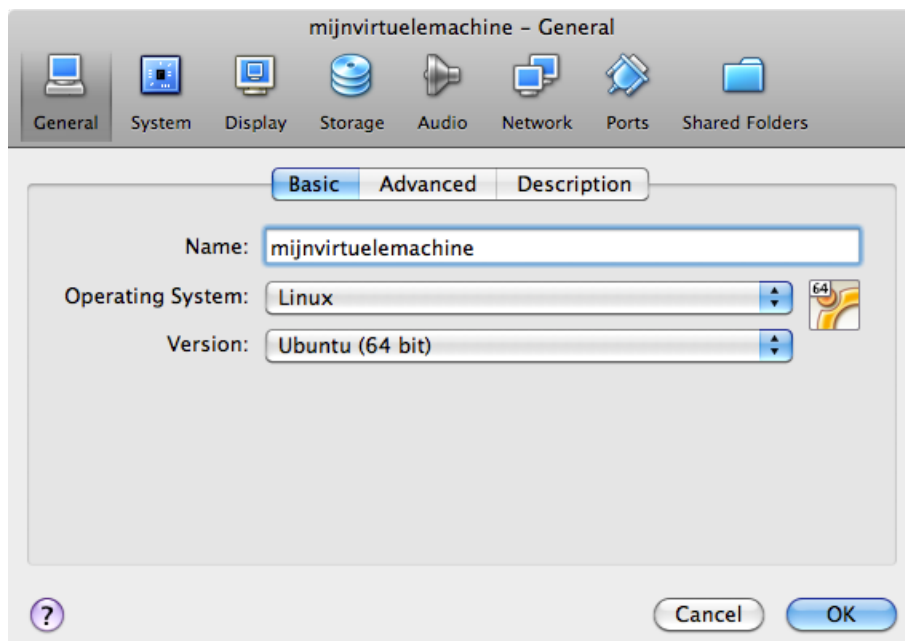


4.4. attach the CD image

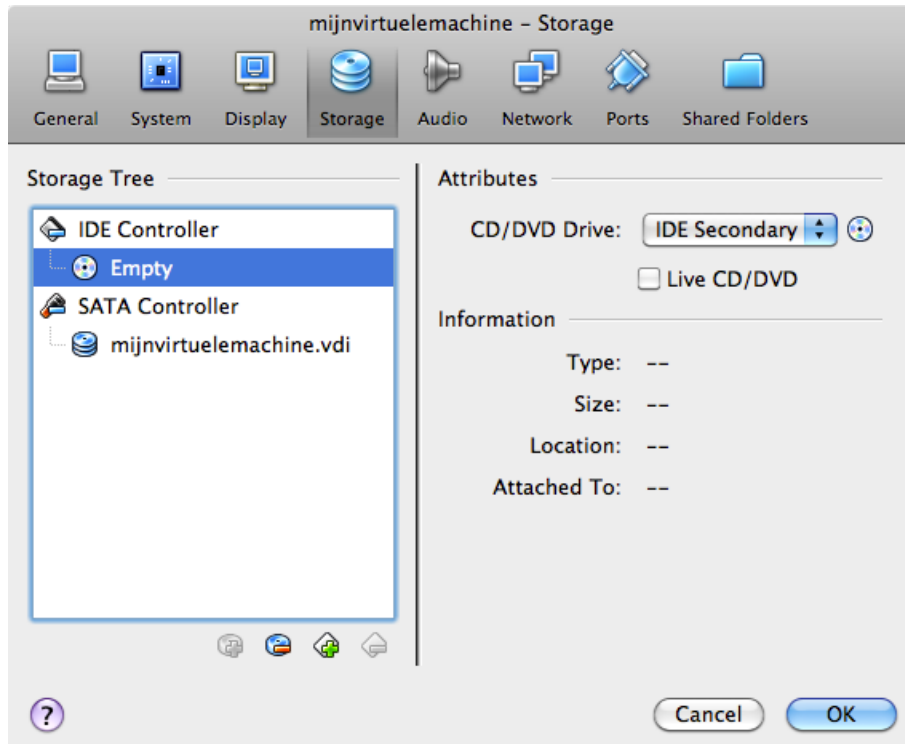
Before we start the virtual computer, let us take a look at some settings (click **Settings**).



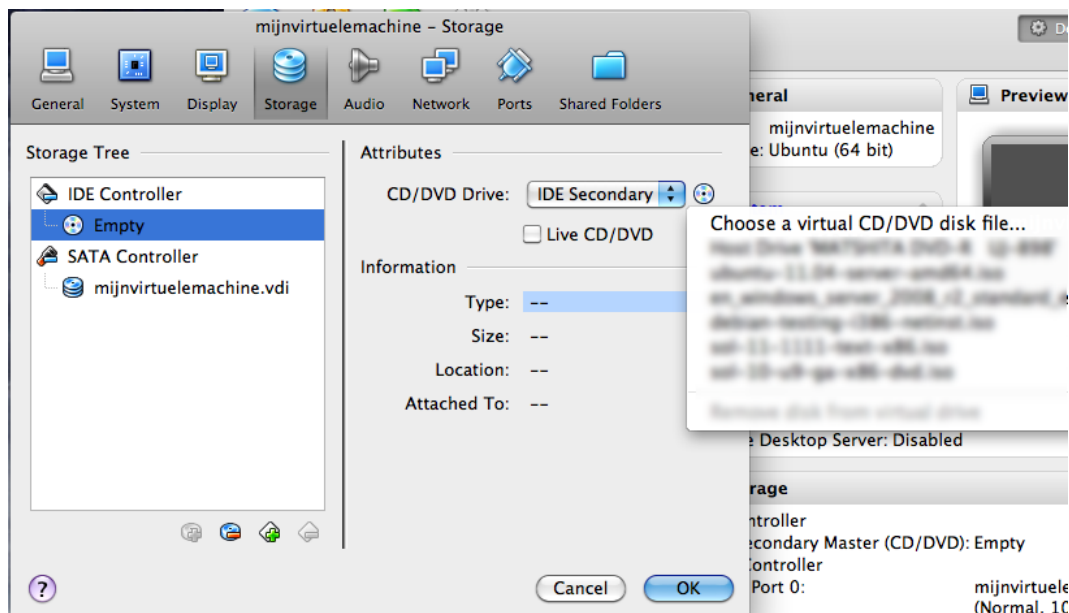
Do not worry if your screen looks different, just find the button named **storage**.



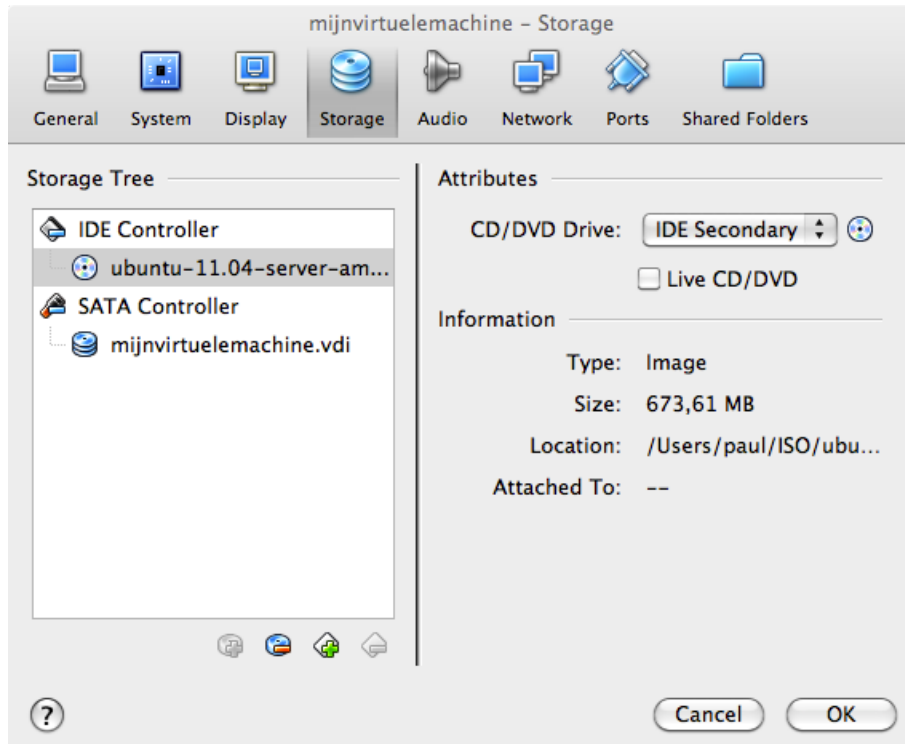
Remember the .ISO file you downloaded? Connect this .ISO file to this virtual machine by clicking on the CD icon next to **Empty**.



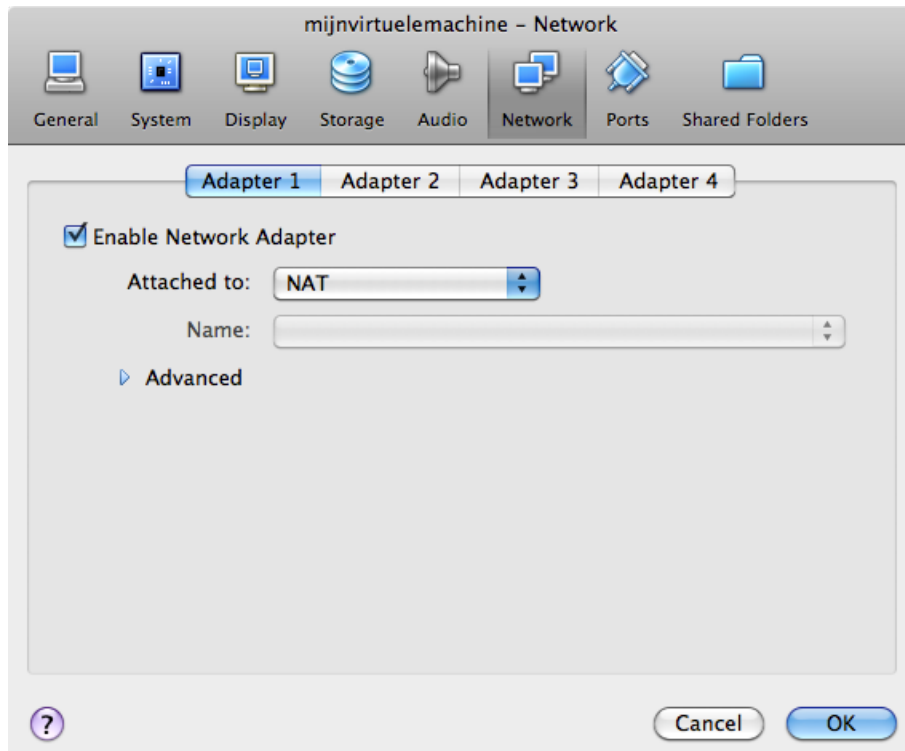
Now click on the other CD icon and attach your ISO file to this virtual CD drive.



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 sysklogd
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 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
```

ls -lh

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  2 paul paul 4096 2007-01-10 21:13 .
drwxr-xr-x 39 paul paul 4096 2007-01-10 21:13 ..
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
rmdir: 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:~$
```

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**.

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 bre1.txt
paul@laika:~/Linux$ rm -i bre1.txt
rm: remove regular empty file `bre1.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
```

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.
- 3a. Download wolf.jpg and LinuxFun.pdf from <http://linux-training.be> (wget <http://linux-training.be/files/studentfiles/wolf.jpg> and wget <http://linux-training.be/files/books/LinuxFun.pdf>)
- 3b. Display the type of file of wolf.jpg and LinuxFun.pdf
- 3c. Rename wolf.jpg to wolf.pdf (use mv).
- 3d. Display the type of file of wolf.pdf and LinuxFun.pdf.
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 into it.
10. Use one command to remove the directory ~/testbackup and all files into it.
11. Create a directory ~/etcbackup and copy all *.conf files from /etc into 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
```

- 3a. Download wolf.jpg and LinuxFun.pdf from <http://linux-training.be> (wget <http://linux-training.be/files/studentfiles/wolf.jpg> and wget <http://linux-training.be/files/books/LinuxFun.pdf>)

```
wget http://linux-training.be/files/studentfiles/wolf.jpg
wget http://linux-training.be/files/studentfiles/wolf.png
wget http://linux-training.be/files/books/LinuxFun.pdf
```

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

```
file wolf.jpg LinuxFun.pdf
```

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

```
mv wolf.jpg wolf.pdf
```

- 3d. Display the type of file of wolf.pdf and LinuxFun.pdf.

```
file wolf.pdf LinuxFun.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 into it.

```
mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/
```

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

```
rm -rf ~/testbackup
```

11. Create a directory ~/etcbackup and copy all *.conf files from /etc into 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
daemon: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:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
paul@laika:~$
```

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
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
```

Head can also display the first n bytes.

```
paul@laika:~$ head -c4 /etc/passwd
rootpaul@laika:~$
```

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
csync2         30865/tcp      # cluster synchronization tool
dircproxy      57000/tcp      # Detachable IRC Proxy
tfido          60177/tcp      # fidonet EMSI over telnet
fido           60179/tcp      # fidonet EMSI over TCP

# Local services
paul@laika:~$
```

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).

```
paul@laika:~$ which ls
/bin/ls
paul@laika:~$ strings /bin/ls
/lib/ld-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 chapter 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
bzcat           fuser          nc             stralign
bzcmp           fusermount     nc.traditional stty
bzdiff          get_mouptions netcat          su
bzegrep         grep           netstat        sync
bzexe           gunzip         ntfs-3g        sysfs
bzfgrep         gzexe          ntfs-3g.probe tailf
bzgrep          gzip           parted_devices tar
bzip2           hostname       parted_server  tempfile
bzip2recover   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@ubul010:~$ ls -l /sbin/ifconfig /sbin/fdisk /sbin/mkfs.ext4
-rwxr-xr-x 1 root root 97172 2011-02-02 09:56 /sbin/fdisk
-rwxr-xr-x 1 root root 65708 2010-07-02 09:27 /sbin/ifconfig
-rwxr-xr-x 5 root root 55140 2010-08-18 18:01 /sbin/mkfs.ext4
```

/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/brltty.conf           /etc/lftp.conf           /etc/sysctl.conf
/etc/ccertificates.conf    /etc/libao.conf          /etc/syslog.conf
/etc/cvs-cron.conf         /etc/logrotate.conf     /etc/ucf.conf
/etc/ddclient.conf        /etc/ltrace.conf        /etc/uniconf.conf
/etc/debconf.conf         /etc/mke2fs.conf        /etc/updatedb.conf
/etc/deluser.conf         /etc/netscsid.conf       /etc/usplash.conf
/etc/fdmount.conf         /etc/nsswitch.conf       /etc/uswsusp.conf
/etc/hdparm.conf          /etc/pam.conf            /etc/vnc.conf
/etc/host.conf            /etc/pnm2ppa.conf        /etc/wodim.conf
/etc/inetd.conf           /etc/povray.conf         /etc/wvdial.conf
/etc/kernel-img.conf      /etc/resolv.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**.

/etc/skel/

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.

/etc/sysconfig/

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   selinux
authconfig    hidd         keyboard     ntpd         spamassassin
autofs        httpd        kudzu        openib.conf  squid
bluetooth     hwconf       lm_sensors   pand         syslog
clock         il8n         mouse        pcmcia       sys-config-sec
console       init         mouse.B      pgsql        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@ubu606:~$ 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@ubu606:~$ 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/.lessht
```

/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
```

/mnt

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.

/tmp

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 ;-).*

/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          slabinfo
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       sysvipc
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  7513      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      loadavg     vmcore
2        45     5380  6430  7189  acpi      locks       vmnet
20845    4542  5412  6450  7191  asound    meminfo     vmstat
221     46     5414  6551  7192  buddyinfo misc         zoneinfo
2338    4704  5416  6568  7199  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 : 000000002cb41780
Cpu1Bogo : 498.68
Cpu1ClkTck : 000000002cb41780
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`.

/proc/interrupts

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

```
paul@RHELv4u4:~$ cat /proc/interrupts
CPU0
 0:   13876877  IO-APIC-edge  timer
 1:         15  IO-APIC-edge  i8042
 8:          1  IO-APIC-edge  rtc
 9:          0  IO-APIC-level acpi
12:         67  IO-APIC-edge  i8042
14:        128  IO-APIC-edge  ide0
15:       124320  IO-APIC-edge  ide1
169:       111993  IO-APIC-level ioc0
177:        2428  IO-APIC-level eth0
NMI:          0
LOC:       13878037
ERR:          0
MIS:          0
```

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

```
paul@laika:~$ cat /proc/interrupts
CPU0          CPU1
 0:   860013    0  IO-APIC-edge  timer
 1:   4533     0  IO-APIC-edge  i8042
 7:    0       0  IO-APIC-edge  parport0
 8:  6588227   0  IO-APIC-edge  rtc
10:   2314    0  IO-APIC-fasteoi acpi
12:   133     0  IO-APIC-edge  i8042
14:    0       0  IO-APIC-edge  libata
15:   72269   0  IO-APIC-edge  libata
18:    1       0  IO-APIC-fasteoi yenta
19:  115036   0  IO-APIC-fasteoi eth0
20:  126871   0  IO-APIC-fasteoi libata, ohci1394
21:   30204   0  IO-APIC-fasteoi ehci_hcd:usb1, uhci_hcd:usb2
22:   1334    0  IO-APIC-fasteoi saa7133[0], saa7133[0]
24:  234739   0  IO-APIC-fasteoi nvidia
NMI:         72    42
LOC:   860000  859994
ERR:          0
```

/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@ubu1010:~$ ls /usr/include/
aalib.h          expat_config.h  math.h          search.h
af_vfs.h        expat_external.h mcheck.h       semaphore.h
aio.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/
```

/usr/share

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@ubul010:~$ ls /usr/share/games/
openttd wesnoth
```

/usr/src

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

```
paul@deb508:~$ ls -l /usr/src/
total 12
drwxr-xr-x 4 root root 4096 2011-02-01 14:43 linux-headers-2.6.26-2-686
drwxr-xr-x 18 root root 4096 2011-02-01 14:43 linux-headers-2.6.26-2-common
drwxr-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
boot.log.1     httpd       messages.3  sa           uucp
boot.log.2     iiim        messages.4  samba        vbox
boot.log.3     iptraf      mysqld.log  scrollkeeper.log vmware-tools-guestd
boot.log.4     lastlog     news        secure       wtmp
canna          mail        pgsql       secure.1     wtmp.1
cron           maillog     ppp         secure.2     Xorg.0.log
cron.1         maillog.1  prelink.log secure.3     Xorg.0.log.old
```

/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 messagebus: messagebus startup succeeded
Jul 30 05:13:57 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@ubul010:~$ ls /var/cache/
apt          dictionaries-common  gdm      man          software-center
binfmts     flashplugin-installer hald     pm-utils
cups        fontconfig           jockey   pppconfig
debconf     fonts                ldconfig samba
```


/var/spool

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).

/var/lib

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

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

/var/...

/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=zeros.txt count=1 bs=100
od zeros.txt
```

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

`/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), **cs**h (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    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 \ta tab"
A line with      a tab
[paul@RHEL4b ~]$ echo -e 'A line with \ta 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...
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.

```
[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.

```
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**.

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

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.

```
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 ?
6. List all current aliases.
- 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:


```
??\
```

Find two solutions with single quotes, two with double quotes and one without quotes (and say thank you to René and Darioush from Google for this extra).

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)
```

6. List all current aliases.

```
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.

```
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:

```
echo
```

```
/bin/echo
```

The **echo** command will be interpreted by the shell as the **built-in echo** command. The **/bin/echo** command will make the shell execute the **echo binary** located in the **/bin** directory.

14. Explain the difference between the following commands:

```
echo Hello
```

```
echo -n Hello
```

The **-n** option of the **echo** command will prevent echo from echoing a trailing newline. **echo Hello** will echo six characters in total, **echo -n hello** only echoes five characters.

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

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

```
echo "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 **\t**.

```
echo -e "4+4\t=8" ; echo -e "10+14\t=24"
```

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

```
??\
echo '??\'
echo -e '??\
echo "??\
echo -e "??\
echo ??\
```

Find two solutions with single quotes, two with double quotes and one without quotes (and say thank you to René and Darioush from Google for this extra).

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

```
echo -e "one \ntwo \nthree"
```

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 **\$?**. Actually **\$?** is a shell parameter and not a variable, since you cannot assign a value to **\$?**.

```
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  
it worked
```

```
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:\w$'
paul@deb503:~$
```


To avoid unrecoverable mistakes, you can set normal user prompts to green and the root prompt to red. Add the following to your **.bashrc** for a green user prompt:

```
# color prompt by paul
RED='\[\033[01;31m\'
WHITE='\[\033[01;00m\'
GREEN='\[\033[01;32m\'
BLUE='\[\033[01;34m\'
export PS1="\${debian_chroot:+($debian_chroot)}$GREEN\u$WHITE@$BLUE\h$WHITE\w\$ "
```

\$PATH

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.

```
[paul@RHEL4b ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:
```

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**.

```
[paul@RHEL4b ~]$ PATH=$PATH:.
[paul@RHEL4b ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:.
[paul@RHEL4b ~]$
```

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.

```
[paul@RHEL3 ~]$ su
Password:
[root@RHEL3 paul]# echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin
[root@RHEL3 paul]# exit
[paul@RHEL3 ~]$ su -
Password:
[root@RHEL3 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
[root@RHEL3 ~]#
```

creating variables

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

[paul@RHEL4b ~]$
```

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.

```
[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).

```
[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.

```
[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
four
```

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

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

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 $(C=sub;echo $C$B$A))
shell subshell subsubshell
```

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 **answer** with a value of **42**.
3. Copy the value of `$LANG` to `$MyLANG`.
4. List all current shell variables.
5. List all exported shell variables.
6. Do the **env** and **set** commands display your variable ?
6. Destroy your **answer** variable.
7. Find the list of shell options in the man page of **bash**. What is the difference between **set -u** and **set -o nounset**?
8. Create two variables, and **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 **echo** and the two variables to echo Dumbledore.
11. Activate **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 **PS1** prompt.
14. Execute **cd /var** and **ls** in an embedded shell.
15. Create the variable `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 **long listing** (`ls -l`) of the **passwd** command using the **which** command inside back ticks.

12.12. solution: shell variables

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

```
echo Hello $USER
```

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

```
answer=42
```

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

```
MyLANG=$LANG
```

4. List all current shell variables.

```
set
```

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

5. List all exported shell variables.

```
env
```

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

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

6. Destroy your **answer** variable.

```
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.

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

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

```
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.

```
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.


```
set -u
OR
set -o nounset
```

Both these lines have the same effect.

12. Deactivate nounset.

```
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.

```
PS1='\t \u@\h \w$ '
```

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

```
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 ?

```
$(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**.

```
[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 ticks.

```
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
paul@debian5:~/test42$
```

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
paul@debian5:~/test42$
```

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@ubul010:~$ 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@ubul010:~$ !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 file*
file1 file2 file3 fileab fileabc
[paul@RHELv4u3 gen]$ ls *ile55
File55
[paul@RHELv4u3 gen]$ ls F*ile55
File55
[paul@RHELv4u3 gen]$ ls F*55
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
```

```
[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
file1 file2 file3 File4 File55 FileA fileab Fileab FileAB fileabc
[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 fileab FileAB fileabc
file2 File4 FileA Fileab fileab2
[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
```

Globbering 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.

```
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)

```
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

```
ls file*
```

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

```
ls File*
```

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

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

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

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

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

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

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

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

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

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

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

```
ls ?????
```

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

```
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.

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

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

```
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 ~]$
```

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 > dirlist 2>&1
```

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

```
ls 2>&1 > dirlist
```

directs only the standard output to file `dirlist`, because the standard error made a copy of the standard output before the standard output was redirected to `dirlist`.

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 <<bro1 > text.txt
> brel
> bro1
[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@ubul1110~$ base64 <<< linux-training.be
bGludXgt dHJhaW5pbm cuYmUK
paul@ubul1110~$ base64 -d <<< bGludXgt dHJhaW5pbm cuYmUK
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 **tailng.txt** file. Then type **tail -f tailng.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**.

```
ls /etc > etc.txt
```

2. Activate the **noclobber** shell option.

```
set -o noclobber
```

3. Verify that **noclobber** is active by repeating your **ls** on **/etc/**.

```
ls /etc > etc.txt (should not work)
```

4. When listing all shell options, which character represents the **noclobber** option ?

```
echo $- (noclobber is visible as C)
```

5. Deactivate the **noclobber** option.

```
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.

```
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.

```
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
16.14. solution: filters	134

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.

```
[paul@RHEL4b pipes]$ grep Bel tennis.txt
Justine Henin, Bel
[paul@RHEL4b pipes]$ grep -i Bel tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

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:
faff: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 CliJstErs, BEL
JustinE HEnin, BEL
SErEna Williams, usa
VENus Williams, USA
```

Here we set all letters to uppercase by defining two ranges.

```
[paul@RHEL4b pipes]$ cat tennis.txt | tr 'a-z' 'A-Z'
AMELIE MAURES MO, 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
```

```
two
three
four
five
[paul@RHEL4b pipes]$ cat count.txt | tr '\n' ' '
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
Amlı Maursmo, Fra
Kim Clijstrs, 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]$
```

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 -l2 list1.txt list2.txt
Abba
Cure
Queen
paul@debian5:~/pipes$ comm -l3 list1.txt list2.txt
Turner
paul@debian5:~/pipes$ comm -l3 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  \n
0000020
```


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.

```
grep bash /etc/passwd | cut -d: -f1 | sort > bashusers.txt
```

2. Put a sorted list of all logged on users in onlineusers.txt.

```
who | cut -d' ' -f1 | sort > onlineusers.txt
```

3. Make a list of all filenames in **/etc** that contain the string samba.

```
ls /etc | grep samba
```

4. Make a sorted list of all files in **/etc** that contain the case insensitive string samba.

```
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.

```
/sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut -d' ' -f3,5
```

6. Write a line that removes all non-letters from a stream.

```
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.

```
paul@deb503:~$ cat text2
it is very cold today without the sun

paul@deb503:~$ cat text2 | tr ' ' '\n'
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/**.)

```
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. bzip2 - bunzip2	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).

```
find . -name "*.conf"
```

Find files of type file (not directory, pipe or etc.) that end in **.conf**.

```
find . -type f -name "*.conf"
```

Find files of type directory that end in **.bak**.

```
find /data -type d -name "*.bak"
```

Find files that are newer than **file42.txt**

```
find . -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/**.

```
find /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.

```
find /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@rhel155 ~$ 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@rhel155 ~$ 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@rhel155 ~$ date +%s
1271501080
```

When will this seconds counter reach two thousand million ?

```
paul@rhel155 ~$ 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
```

This **bzip2** command compresses a file and uses a lot of **cpu time**.

```
paul@rhel155 ~$ 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@rhel155 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
paul@rhel155 ~$ gzip text.txt
paul@rhel155 ~$ 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@rhel155 ~$ gunzip text.txt.gz
paul@rhel155 ~$ 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@rhel155 ~$ head -4 text.txt
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
paul@rhel155 ~$ gzip text.txt
paul@rhel155 ~$ zcat text.txt.gz | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

17.9. bzip2 - bunzip2

Files can also be compressed with **bzip2** which takes a little more time than **gzip**, but compresses better.

```
paul@rhel155 ~$ bzip2 text.txt
paul@rhel155 ~$ 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@rhel155 ~$ bunzip2 text.txt.bz2
paul@rhel155 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
```

17.10. bzipcat - bzipmore

And in the same way **bzipcat** and **bzipmore** can display files compressed with **bzip2**.

```
paul@rhel155 ~$ bzip2 text.txt
paul@rhel155 ~$ bzipcat text.txt.bz2 | head -4
/
/opt
/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 :

```
find . -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 {} {}1 \;
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>

Part V. vi

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 **vimtutor** 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 18.1. getting to command mode

key	action
Esc	set vi(m) in command mode.

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 18.2. switch to insert mode

command	action
a	start typing after the current character
A	start typing at the end of the current line
i	start typing before the current character
I	start typing at the start of the current line
o	start typing on a new line after the current line
O	start typing on a new line before the current line

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

command	action
x	delete the character below the cursor
X	delete the character before the cursor
r	replace the character below the cursor
p	paste after the cursor (here the last deleted character)
xp	switch two characters

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

command	action
u	undo the last action
.	repeat the last action

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

command	action
dd	cut the current line
yy	(yank yank) copy the current line
p	paste after the current line
P	paste before the current line

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 18.6. cut, copy and paste lines

command	action
<code>3dd</code>	cut three lines
<code>4yy</code>	copy four lines

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 18.7. start and end of line

command	action
<code>0</code>	jump to start of current line
<code>^</code>	jump to start of current line
<code>\$</code>	jump to end of current line
<code>d0</code>	delete until start of line
<code>d\$</code>	delete until end of line

18.8. join two lines (J) and more

When in command mode, pressing `J` will append the next line to the current line. With `yy` you duplicate a line and with `ddp` you switch two lines.

Table 18.8. join two lines

command	action
<code>J</code>	join two lines
<code>yy</code>	duplicate a line
<code>ddp</code>	switch two lines

18.9. words (w b)

When in command mode, **w** will jump to the next word and **b** will move to the previous word. **w** and **b** can also be combined with **d** and **y** to copy and cut words (**dw db yw yb**).

Table 18.9. words

command	action
w	forward one word
b	back one word
3w	forward three words
dw	delete one word
yw	yank (copy) one word
5yb	yank five words back
7dw	delete seven words

18.10. save (or not) and exit (:w :q :q!)

Pressing the colon **:** will allow you to give instructions to vi (technically speaking, typing the colon will open the **ex** editor). **:w** will write (save) the file, **:q** will quit an unchanged file without saving, and **:q!** will quit vi discarding any changes. **:wq** will save and quit and is the same as typing **ZZ** in command mode.

Table 18.10. save and exit vi

command	action
:w	save (write)
:w fname	save as fname
:q	quit
:wq	save and quit
ZZ	save and quit
:q!	quit (discarding your changes)
:w!	save (and write to non-writable file!)

The last one is a bit special. With **:w!** vi will try to **chmod** the file to get write permission (this works when you are the owner) and will **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

command	action
/string	forward search for string
?string	backward search for string
n	go to next occurrence of search string
/^string	forward search string at beginning of line
/string\$	forward search string at end of line
/br[aeio]l	search for bral brel bril and brol
^\<he\>	search for the word he (and not for here or the)

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

command	action
:4,8 s/foo/bar/g	replace foo with bar on lines 4 to 8
:1,\$ s/foo/bar/g	replace foo with bar on all lines

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

command	action
:r fname	(read) file fname and paste contents
:r !cmd	execute cmd and paste its output

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

command	action
"add	delete current line and put text in buffer a
"g7yy	copy seven lines into buffer g
"ap	paste from buffer a

18.15. multiple files

You can edit multiple files with vi. Here are some tips.

Table 18.15. multiple files

command	action
vi file1 file2 file3	start editing three files
:args	lists files and marks active file
:n	start editing the next file
:e	toggle with last edited file
:rew	rewind file pointer to first file

18.16. abbreviations

With **:ab** you can put abbreviations in vi. Use **:una** to undo the abbreviation.

Table 18.16. abbreviations

command	action
:ab str long string	abbreviate str to be 'long string'
:una str	un-abbreviate str

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	160
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**.

```
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.

```
#!/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.

```
#!/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.

```
#!/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.

```
#!/bin/bash
#
# Hello World Script
#
echo Hello World
```

19.5. variables

Here is a simple example of a variable inside a script.

```
#!/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

[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.

```
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).

```
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.

```
#!/bin/bash -
or
#!/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.

```
#!/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.

```
#!/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.

```
for i in 1 2 4
do
    echo $i
done
```

An example of a **for loop** combined with an embedded shell.

```
#!/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.

```
#!/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.

```
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**.

```
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.

```
#!/bin/ksh
# endless loop
while :
do
    echo hello
    sleep 1
done
```

20.6. until loop

Below a simple example of an **until loop**.

```
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.

```
#!/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.

```
#!/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.

```
#!/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.

```
#!/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.

```
#!/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**.

```
#!/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 getopt	174
21.6. get shell options with shopt	175
21.7. practice: parameters and options	176
21.8. solution: parameters and options	177

21.1. script parameters

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.

```
#!/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
echo 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 **shift** statement can parse all **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 **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 her 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 **getopts** function allows you to parse options given to a command. The following script allows for any combination of the options a, f and z.

```
kahlan@solexp11$ cat options.ksh
#!/bin/ksh

while getopts ":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 $OPTARG
      ;;
    z)
      echo received -z
      ;;
    :)
      echo "option -$OPTARG needs an argument"
      ;;
    *)
      echo "invalid option -$OPTARG"
      ;;
  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.

```
echo $4 $3 $2 $1
```

2. Write a script that receives two parameters (two filenames) and outputs whether those files exist.

```
#!/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
```

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.

```
[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.

```
[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).

```
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.

```
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 x 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 `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
```

```
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 shad
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`.

```
[root@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
wim:x:525:531:master damuti:/home/wim:/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
```

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$YsYvcVKqByFVYlKnU3ncd0:14054:0:99999:7:::
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:$1$TvbylKpa$L.WzgbujUS3LClIRmdv1:14054:0:99999:7:::
```

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.Q5Osc2Khbm1
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$42$Hb3nvP0KwHSQ7fQmI1Y7R.
```

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$qqxoLqiSVNvGr5ybMxEVM1
```

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
```

chage

The **chage** 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 **chage** 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.


```
[root@RHEL4 ~]# usermod -L harry
[root@RHEL4 ~]# tail -1 /etc/shadow
harry: !\$1\$143TO9IZ\$RLm/FpQkpDrV4/Tkhku5e1:13717:0:99999:7:::
```

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!

```
[root@RHEL4 ~]# su - harry
[harry@RHEL4 ~]$
```

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.

```
[root@RHEL5 ~]# vipw /etc/passwd
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/
drwx----- 2 laura laura 4096 Jun 24 15:17 /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/bsh
/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@rhel153 ~]$ 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 23.1. Debian User Environment

script	su	su -	ssh	gdm
~/.bashrc	no	yes	yes	yes
~/.profile	no	yes	yes	yes
/etc/profile	no	yes	yes	yes
/etc/bash.bashrc	yes	no	no	yes

RHEL5 overview

Below is a table overview of when Red Hat Enterprise Linux 5 is running any of these bash startup scripts.

Table 23.2. Red Hat User Environment

script	su	su -	ssh	gdm
~/.bashrc	yes	yes	yes	yes
~/.bash_profile	no	yes	yes	yes
/etc/profile	no	yes	yes	yes
/etc/bashrc	yes	yes	yes	yes

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 formulal
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:
formulal: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 formulal annelies
root@laika:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
snooker:x:1008:sandra
formulal: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:
formulal: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-rw-r-- 1 paul proj 984 Feb 5 15:38 data.odt
-rw-r--r-- 1 root root 0 Feb 7 16:07 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 root 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

first character	file type
-	normal file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	socket

Below a screenshot of a character device (the console) and a block device (the hard disk).

```
paul@debian61t~$ 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@debian61t~$ 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

permission	on a file	on a directory
r (read)	read file contents (<code>cat</code>)	read directory contents (<code>ls</code>)
w (write)	change file contents (<code>vi</code>)	create files in (<code>touch</code>)
x (execute)	execute the file	enter the directory (<code>cd</code>)

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

position	characters	function
1	-	this is a regular file
2-4	rwX	permissions for the user owner
5-7	r-X	permissions for the group owner
8-10	r--	permissions for others

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
-rwxrwxrwx 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=rw,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

binary	octal	permission
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwX

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
```

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
```

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

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/
drwxrwsr-x  2 root proj55 4096 Feb  7 17:45 /project55/
root@RHELv4u4:~# ls -l /project55/
total 4
-rw-r--r--  1 root proj55 0 Feb  7 17:45 fromroot.txt
root@RHELv4u4:~#
```

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.

```
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.

```
chmod 770 /home/sports
```

1c. All files created in this directory should be group-owned by the sports group.

```
chmod 2770 /home/sports
```

1d. Users should be able to delete only their own user-owned files.

```
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.

```
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.

```
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.

```
paul@laika:~$ sudo su -
[sudo] password for paul:
root@laika:~# mkdir attr
root@laika:~# cd 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:rw-
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:rw-
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

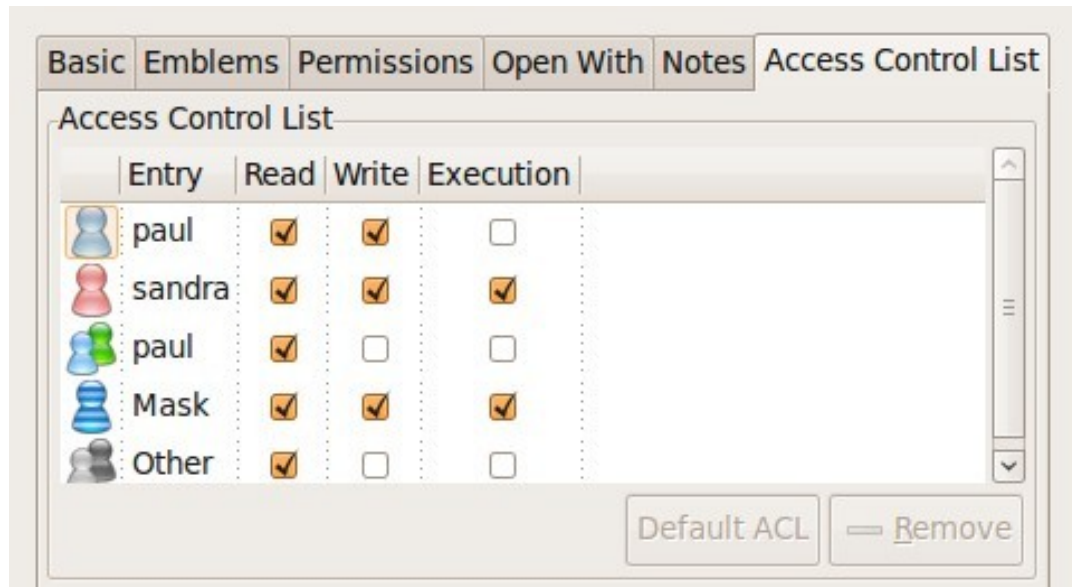
The **acl mask** defines the maximum effective permissions for any entry in the **acl**. This **mask** is calculated every time you execute the **setfacl** or **chmod** commands.

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:rw-   #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 -li** 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   259937    1% /home/project42
/dev/sdb1           74400     5519   68881    8% /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:~/test$ ln file2 hardlink_to_file2
paul@RHELv4u4:~/test$ 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:~/test$
```

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:~/test$ 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 slsummer.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. process management

Chapter 29. introduction to processes

Table of Contents

29.1. terminology	244
29.2. basic process management	245
29.3. signalling processes	249
29.4. practice : basic process management	252
29.5. solution : basic process management	253

29.1. terminology

process

A **process** is compiled source code that is currently running on the system.

PID

All processes have a **process id** or **PID**.

PPID

Every process has a parent process (with a **PPID**). The **child** process is often started by the **parent** process.

init

The **init** process always has process ID 1. The **init** process is started by the **kernel** itself so technically it does not have a parent process. **init** serves as a **foster parent** for **orphaned** processes.

kill

When a process stops running, the process dies, when you want a process to die, you **kill** it.

daemon

Processes that start at system startup and keep running forever are called **daemon** processes or **daemons**. These **daemons** never die.

zombie

When a process is killed, but it still shows up on the system, then the process is referred to as **zombie**. You cannot kill zombies, because they are already dead.

29.2. basic process management

\$\$ and \$PPID

Some shell environment variables contain information about processes. The **\$\$** variable will hold your current **process ID**, and **\$PPID** contains the **parent PID**. Actually **\$\$** is a shell parameter and not a variable, you cannot assign a value to it.

Below we use **echo** to display the values of **\$\$** and **\$PPID**.

```
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
```

pidof

You can find all process id's by name using the **pidof** command.

```
root@rhel53 ~# pidof mingetty
2819 2798 2797 2796 2795 2794
```

parent and child

Processes have a **parent-child** relationship. Every process has a parent process.

When starting a new **bash** you can use **echo** to verify that the **pid** from before is the **ppid** of the new shell. The **child** process from above is now the **parent** process.

```
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $$ $PPID
4812 4224
```

Typing **exit** will end the current process and brings us back to our original values for **\$\$** and **\$PPID**.

```
[paul@RHEL4b ~]$ echo $$ $PPID
4812 4224
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
[paul@RHEL4b ~]$
```

fork and exec

A process starts another process in two phases. First the process creates a **fork** of itself, an identical copy. Then the forked process executes an **exec** to replace the forked process with the target child process.

```
[paul@RHEL4b ~]$ echo $$
4224
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $$ $PPID
5310 4224
[paul@RHEL4b ~]$
```

exec

With the **exec** command, you can execute a process without forking a new process. In the following screenshot a **Korn shell** (ksh) is started and is being replaced with a **bash shell** using the **exec** command. The **pid** of the **bash shell** is the same as the **pid** of the **Korn shell**. Exiting the child **bash shell** will get me back to the parent **bash**, not to the **Korn shell** (which does not exist anymore).

```
[paul@RHEL4b ~]$ echo $$
4224
# PID of bash
[paul@RHEL4b ~]$ ksh
$ echo $$ $PPID
5343 4224
# PID of ksh and bash
$ exec bash
[paul@RHEL4b ~]$ echo $$ $PPID
5343 4224
# PID of bash and bash
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ echo $$
4224
```

ps

One of the most common tools on Linux to look at processes is **ps**. The following screenshot shows the parent child relationship between three bash processes.

```
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $$ $PPID
4866 4224
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $$ $PPID
4884 4866
[paul@RHEL4b ~]$ ps fx
  PID TTY          STAT       TIME COMMAND
  4223 ?            S          0:01 sshd: paul@pts/0
```

```
4224 pts/0    Ss      0:00   \_ -bash
4866 pts/0    S       0:00   \_  bash
4884 pts/0    S       0:00   \_  bash
4902 pts/0    R+      0:00   \_  ps fx
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ ps fx
  PID TTY          STAT       TIME COMMAND
 4223 ?            S          0:01 sshd: paul@pts/0
 4224 pts/0        Ss         0:00   \_ -bash
 4866 pts/0        S          0:00   \_  bash
 4903 pts/0        R+         0:00   \_  ps fx
[paul@RHEL4b ~]$ exit
exit
[paul@RHEL4b ~]$ ps fx
  PID TTY          STAT       TIME COMMAND
 4223 ?            S          0:01 sshd: paul@pts/0
 4224 pts/0        Ss         0:00   \_ -bash
 4904 pts/0        R+         0:00   \_  ps fx
[paul@RHEL4b ~]$
```

On Linux, **ps fax** is often used. On Solaris **ps -ef** (which also works on Linux) is common. Here is a partial output from **ps fax**.

```
[paul@RHEL4a ~]$ ps fax
PID TTY          STAT       TIME COMMAND
 1 ?            S          0:00 init [5]

...

3713 ?            Ss         0:00 /usr/sbin/sshd
5042 ?            Ss         0:00   \_ sshd: paul [priv]
5044 ?            S          0:00   \_ sshd: paul@pts/1
5045 pts/1        Ss         0:00   \_ -bash
5077 pts/1        R+         0:00   \_  ps fax
```

pgrep

Similar to the **ps -C**, you can also use **pgrep** to search for a process by its command name.

```
[paul@RHEL5 ~]$ sleep 1000 &
[1] 32558
[paul@RHEL5 ~]$ pgrep sleep
32558
[paul@RHEL5 ~]$ ps -C sleep
  PID TTY          STAT       TIME CMD
32558 pts/3        S          00:00:00 sleep
```

You can also list the command name of the process with **pgrep**.

```
paul@laika:~$ pgrep -l sleep
9661 sleep
```

top

Another popular tool on Linux is **top**. The **top** tool can order processes according to **cpu usage** or other properties. You can also **kill** processes from within top. Press **h** inside **top** for help.

In case of trouble, top is often the first tool to fire up, since it also provides you memory and swap space information.

29.3. signalling processes

kill

The **kill** command will kill (or stop) a process. The screenshot shows how to use a standard **kill** to stop the process with **pid** 1942.

```
paul@ubuntu910:~$ kill 1942
paul@ubuntu910:~$
```

By using the **kill** we are sending a **signal** to the process.

list signals

Running processes can receive signals from each other or from the users. You can have a list of signals by typing **kill -l**, that is a letter **l**, not the number 1.

```
[paul@RHEL4a ~]$ kill -l
1) SIGHUP          2) SIGINT          3) SIGQUIT        4) SIGILL
5) SIGTRAP        6) SIGABRT        7) SIGBUS         8) SIGFPE
9) SIGKILL       10) SIGUSR1       11) SIGSEGV       12) SIGUSR2
13) SIGPIPE      14) SIGALRM      15) SIGTERM       17) SIGCHLD
18) SIGCONT      19) SIGSTOP      20) SIGTSTP       21) SIGTTIN
22) SIGTTOU      23) SIGURG       24) SIGXCPU       25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF      28) SIGWINCH      29) SIGIO
30) SIGPWR      31) SIGSYS       34) SIGRTMIN      35) SIGRTMIN+1
36) SIGRTMIN+2  37) SIGRTMIN+3   38) SIGRTMIN+4   39) SIGRTMIN+5
40) SIGRTMIN+6  41) SIGRTMIN+7   42) SIGRTMIN+8   43) SIGRTMIN+9
44) SIGRTMIN+10 45) SIGRTMIN+11  46) SIGRTMIN+12  47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15  50) SIGRTMAX-14  51) SIGRTMAX-13
52) SIGRTMAX-12 53) SIGRTMAX-11  54) SIGRTMAX-10  55) SIGRTMAX-9
56) SIGRTMAX-8  57) SIGRTMAX-7   58) SIGRTMAX-6   59) SIGRTMAX-5
60) SIGRTMAX-4  61) SIGRTMAX-3   62) SIGRTMAX-2   63) SIGRTMAX-1
64) SIGRTMAX
[paul@RHEL4a ~]$
```

kill -1 (SIGHUP)

It is common on Linux to use the first signal **SIGHUP** (or HUP or 1) to tell a process that it should re-read its configuration file. Thus, the **kill -1 1** command forces the **init** process (**init** always runs with **pid** 1) to re-read its configuration file.

```
root@deb503:~# kill -1 1
root@deb503:~#
```

It is up to the developer of the process to decide whether the process can do this running, or whether it needs to stop and start. It is up to the user to read the documentation of the program.

kill -15 (SIGTERM)

The **SIGTERM** signal is also called a **standard kill**. Whenever **kill** is executed without specifying the signal, a **kill -15** is assumed.

Both commands in the screenshot below are identical.

```
paul@ubuntu910:~$ kill 1942
paul@ubuntu910:~$ kill -15 1942
```

kill -9 (SIGKILL)

The **SIGKILL** is different from most other signals in that it is not being sent to the process, but to the **Linux kernel**. A **kill -9** is also called a **sure kill**. The **kernel** will shoot down the process. As a developer you have no means to intercept a **kill -9** signal.

```
root@rhel53 ~# kill -9 3342
```

killall

The **killall** command will also default to sending a **signal 15** to the processes.

This command and its SysV counterpart **killall5** can be used when shutting down the system. This screenshot shows how Red Hat Enterprise Linux 5.3 uses **killall5** when halting the system.

```
root@rhel53 ~# grep killall /etc/init.d/halt
action $"Sending all processes the TERM signal..." /sbin/killall5 -15
action $"Sending all processes the KILL signal..." /sbin/killall5 -9
```

pkill

You can use the **pkill** command to kill a process by its command name.

```
[paul@RHEL5 ~]$ sleep 1000 &
[1] 30203
[paul@RHEL5 ~]$ pkill sleep
[1]+  Terminated                  sleep 1000
[paul@RHEL5 ~]$
```

top

Inside **top** the **k** key allows you to select a **signal** and **pid** to kill. Below is a partial screenshot of the line just below the summary in **top** after pressing **k**.


```
PID to kill: 1932
```

```
Kill PID 1932 with signal [15]: 9
```

SIGSTOP and SIGCONT

A running process can be **suspended** when it receives a **SIGSTOP** signal. This is the same as **kill -19** on Linux, but might have a different number in other Unix systems.

A **suspended** process does not use any **cpu cycles**, but it stays in memory and can be re-animated with a **SIGCONT** signal (**kill -18** on Linux).

Both signals will be used in the section about **background** processes.

29.4. practice : basic process management

1. Use **ps** to search for the **init** process by name.
2. What is the **process id** of the **init** process ?
3. Use the **who am i** command to determine your terminal name.
4. Using your terminal name from above, use **ps** to find all processes associated with your terminal.
5. What is the **process id** of your shell ?
6. What is the **parent process id** of your shell ?
7. Start two instances of the **sleep 3342** in background.
8. Locate the **process id** of all **sleep** commands.
9. Display only those two **sleep** processes in **top**. Then quit top.
10. Use a **standard kill** to kill one of the **sleep** processes.
11. Use one command to kill all **sleep** processes.

29.5. solution : basic process management

1. Use **ps** to search for the **init** process by name.

```
root@rhel53 ~# ps -C init
  PID TTY          TIME CMD
    1 ?            00:00:04 init
```

2. What is the **process id** of the **init** process ?

1

3. Use the **who am i** command to determine your terminal name.

```
root@rhel53 ~# who am i
paul      pts/0          2010-04-12 17:44 (192.168.1.38)
```

4. Using your terminal name from above, use **ps** to find all processes associated with your terminal.

```
oot@rhel53 ~# ps fax | grep pts/0
2941 ?          S        0:00      \_ sshd: paul@pts/0
2942 pts/0       Ss       0:00          \_ -bash
2972 pts/0       S        0:00              \_ su -
2973 pts/0       S        0:00                  \_ -bash
3808 pts/0       R+       0:00                      \_ ps fax
3809 pts/0       R+       0:00                          \_ grep pts/0
```

or also

```
root@rhel53 ~# ps -ef | grep pts/0
paul      2941  2939  0 17:44 ?          00:00:00 sshd: paul@pts/0
paul      2942  2941  0 17:44 pts/0      00:00:00 -bash
root      2972  2942  0 17:45 pts/0      00:00:00 su -
root      2973  2972  0 17:45 pts/0      00:00:00 -bash
root      3816  2973  0 21:25 pts/0      00:00:00 ps -ef
root      3817  2973  0 21:25 pts/0      00:00:00 grep pts/0
```

5. What is the **process id** of your shell ?

2973 in the screenshot above, probably different for you

echo \$\$ should display same number as the one you found

6. What is the **parent process id** of your shell ?

2972 in the screenshot above, probably different for you

in this example the PPID is from the **su -** command, but when inside gnome then for example gnome-terminal can be the parent process

7. Start two instances of the **sleep 3342** in background.

```
sleep 3342 &  
sleep 3342 &
```

8. Locate the **process id** of all **sleep** commands.

```
pidof sleep
```

9. Display only those two **sleep** processes in **top**. Then quit top.

```
top -p pidx,pidy (replace pidx pidy with the actual numbers)
```

10. Use a **standard kill** to kill one of the **sleep** processes.

```
kill pidx
```

11. Use one command to kill all **sleep** processes.

```
pkill sleep
```

Chapter 30. process priorities

Table of Contents

30.1. priority and nice values	256
30.2. practice : process priorities	259
30.3. solution : process priorities	260

30.1. priority and nice values

introduction

All processes have a **priority** and a **nice** value. Higher priority processes will get more **cpu time** than lower priority processes. You can influence this with the **nice** and **renice** commands.

pipes (mkfifo)

Processes can communicate with each other via **pipes**. These **pipes** can be created with the **mkfifo** command.

The screenshots shows the creation of four distinct pipes (in a new directory).

```
paul@ubuntu910:~$ mkdir procs
paul@ubuntu910:~$ cd procs/
paul@ubuntu910:~/procs$ mkfifo pipe33a pipe33b pipe42a pipe42b
paul@ubuntu910:~/procs$ ls -l
total 0
prw-r--r-- 1 paul paul 0 2010-04-12 13:21 pipe33a
prw-r--r-- 1 paul paul 0 2010-04-12 13:21 pipe33b
prw-r--r-- 1 paul paul 0 2010-04-12 13:21 pipe42a
prw-r--r-- 1 paul paul 0 2010-04-12 13:21 pipe42b
paul@ubuntu910:~/procs$
```

some fun with cat

To demonstrate the use of the **top** and **renice** commands we will make the **cat** command use the previously created **pipes** to generate a full load on the **cpu**.

The **cat** is copied with a distinct name to the current directory. (This enables us to easily recognize the processes within **top**. You could do the same exercise without copying the **cat** command, but using different users. Or you could just look at the **pid** of each process.)

```
paul@ubuntu910:~/procs$ cp /bin/cat proj33
paul@ubuntu910:~/procs$ cp /bin/cat proj42
paul@ubuntu910:~/procs$ echo -n x | ./proj33 - pipe33a > pipe33b &
[1] 1670
paul@ubuntu910:~/procs$ ./proj33 <pipe33b >pipe33a &
[2] 1671
paul@ubuntu910:~/procs$ echo -n z | ./proj42 - pipe42a > pipe42b &
[3] 1673
paul@ubuntu910:~/procs$ ./proj42 <pipe42b >pipe42a &
[4] 1674
```

The commands you see above will create two **proj33** processes that use **cat** to bounce the **x** character between **pipe33a** and **pipe33b**. And ditto for the **z** character and **proj42**.

top

Just running **top** without options or arguments will display all processes and an overview of innformation. The top of the **top** screen might look something like this.

```
top - 13:59:29 up 48 min,  4 users,  load average: 1.06, 0.25, 0.14
Tasks: 139 total,   3 running, 136 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us, 99.7%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem:    509352k total,  460040k used,  49312k free,   66752k buffers
Swap:   746980k total,    0k used,  746980k free,  247324k cached
```

Notice the **cpu idle time (0.0%id)** is zero. This is because our **cat** processes are consuming the whole **cpu**. Results can vary on systems with four or more **cpu cores**.

top -p

The **top -p 1670,1671,1673,1674** screenshot below shows four processes, all of then using approximately 25 percent of the **cpu**.

```
paul@ubuntu910:~$ top -p 1670,1671,1673,1674
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1674	paul	20	0	2972	616	524	S	26.6	0.1	0:11.92	proj42
1670	paul	20	0	2972	616	524	R	25.0	0.1	0:23.16	proj33
1671	paul	20	0	2972	616	524	S	24.6	0.1	0:23.07	proj33
1673	paul	20	0	2972	620	524	R	23.0	0.1	0:11.48	proj42

All four processes have an equal **priority (PR)**, and are battling for **cpu time**. On some systems the **Linux kernel** might attribute slightly varying **priority values**, but the result will still be four processes fighting for **cpu time**.

renice

Since the processes are already running, we need to use the **renice** command to change their **nice value (NI)**.

The screenshot shows how to use **renice** on both the **proj33** processes.

```
paul@ubuntu910:~$ renice +8 1670
1670: old priority 0, new priority 8
paul@ubuntu910:~$ renice +8 1671
1671: old priority 0, new priority 8
```

Normal users can attribute a **nice value** from zero to 20 to processes they own. Only the **root** user can use negative nice values. Be very careful with negative nice values, since they can make it impossible to use the keyboard or ssh to a system.

impact of nice values

The impact of a nice value on running processes can vary. The screenshot below shows the result of our **renice +8** command. Look at the **%CPU** values.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1674	paul	20	0	2972	616	524	S	46.6	0.1	0:22.37	proj42
1673	paul	20	0	2972	620	524	R	42.6	0.1	0:21.65	proj42
1671	paul	28	8	2972	616	524	S	5.7	0.1	0:29.65	proj33
1670	paul	28	8	2972	616	524	R	4.7	0.1	0:29.82	proj33

Important to remember is to always make less important processes nice to more important processes. Using **negative nice values** can have a serere impact on a system's usability.

nice

The **nice** works identical to the **renice** but it is used when starting a command.

The screenshot shows how to start a script with a **nice** value of five.

```
paul@ubuntu910:~$ nice -5 ./backup.sh
```


30.2. practice : process priorities

1. Create a new directory and create six **pipes** in that directory.
2. Bounce a character between two **pipes**.
3. Use **top** and **ps** to display information (pid, ppid, priority, nice value, ...) about these two cat processes.
4. Bounce another character between two other pipes, but this time start the commands **nice**. Verify that all **cat** processes are battling for the cpu. (Feel free to fire up two more cats with the remaining pipes).
5. Use **ps** to verify that the two new **cat** processes have a **nice** value. Use the -o and -C options of **ps** for this.
6. Use **renice** to increase the nice value from 10 to 15. Notice the difference with the usual commands.

30.3. solution : process priorities

1. Create a new directory and create six **pipes** in that directory.

```
[paul@rhel53 ~]$ mkdir pipes ; cd pipes
[paul@rhel53 pipes]$ mkfifo p1 p2 p3 p4 p5 p6
[paul@rhel53 pipes]$ ls -l
total 0
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p1
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p2
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p3
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p4
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p5
prw-rw-r-- 1 paul paul 0 Apr 12 22:15 p6
```

2. Bounce a character between two **pipes**.

```
[paul@rhel53 pipes]$ echo -n x | cat - p1 > p2 &
[1] 4013
[paul@rhel53 pipes]$ cat <p2 >p1 &
[2] 4016
```

3. Use **top** and **ps** to display information (pid, ppid, priority, nice value, ...) about these two cat processes.

```
top (probably the top two lines)

[paul@rhel53 pipes]$ ps -C cat
  PID TTY          TIME CMD
 4013 pts/0        00:03:38 cat
 4016 pts/0        00:01:07 cat

[paul@rhel53 pipes]$ ps fax | grep cat
 4013 pts/0    R      4:00 |           \_ cat - p1
 4016 pts/0    S      1:13 |           \_ cat
 4044 pts/0    S+    0:00 |           \_ grep cat
```

4. Bounce another character between two other pipes, but this time start the commands **nice**. Verify that all **cat** processes are battling for the cpu. (Feel free to fire up two more cats with the remaining pipes).

```
echo -n y | nice cat - p3 > p4 &
nice cat <p4 >p3 &
```

5. Use **ps** to verify that the two new **cat** processes have a **nice** value. Use the **-o** and **-C** options of **ps** for this.

```
[paul@rhel53 pipes]$ ps -C cat -o pid,ppid,pri,ni,comm
  PID PPID PRI  NI COMMAND
 4013 3947  14   0 cat
 4016 3947  21   0 cat
 4025 3947  13  10 cat
 4026 3947  13  10 cat
```

6. Use **renice** to increase the nice value from 10 to 15. Notice the difference with the usual commands.

```
[paul@rhel53 pipes]$ renice +15 4025
4025: old priority 10, new priority 15
[paul@rhel53 pipes]$ renice +15 4026
```

4026: old priority 10, new priority 15

```
[paul@rhel53 pipes]$ ps -C cat -o pid,ppid,pri,ni,comm
  PID  PPID  PRI  NI  COMMAND
  4013  3947  14   0  cat
  4016  3947  21   0  cat
  4025  3947   9  15  cat
  4026  3947   8  15  cat
```

Chapter 31. background jobs

Table of Contents

31.1. background processes	263
31.2. practice : background processes	265
31.3. solution : background processes	266

31.1. background processes

jobs

Stuff that runs in background of your current shell can be displayed with the **jobs** command. By default you will not have any **jobs** running in background.

```
root@rhel53 ~# jobs
root@rhel53 ~#
```

This **jobs** command will be used several times in this section.

control-Z

Some processes can be **suspended** with the **Ctrl-Z** key combination. This sends a **SIGSTOP** signal to the **Linux kernel**, effectively freezing the operation of the process.

When doing this in **vi(m)**, then **vi(m)** goes to the background. The background **vi(m)** can be seen with the **jobs** command.

```
[paul@RHEL4a ~]$ vi procdemo.txt

[5]+  Stopped                  vim procdemo.txt
[paul@RHEL4a ~]$ jobs
[5]+  Stopped                  vim procdemo.txt
```

& ampersand

Processes that are started in background using the **&** character at the end of the command line are also visible with the **jobs** command.

```
[paul@RHEL4a ~]$ find / > allfiles.txt 2> /dev/null &
[6] 5230
[paul@RHEL4a ~]$ jobs
[5]+  Stopped                  vim procdemo.txt
[6]-  Running                  find / >allfiles.txt 2>/dev/null &
[paul@RHEL4a ~]$
```

jobs -p

An interesting option is **jobs -p** to see the **process id** of background processes.

```
[paul@RHEL4b ~]$ sleep 500 &
```

```
[1] 4902
[paul@RHEL4b ~]$ sleep 400 &
[2] 4903
[paul@RHEL4b ~]$ jobs -p
4902
4903
[paul@RHEL4b ~]$ ps `jobs -p`
  PID TTY          STAT       TIME COMMAND
 4902 pts/0        S           0:00 sleep 500
 4903 pts/0        S           0:00 sleep 400
[paul@RHEL4b ~]$
```

fg

Running the **fg** command will bring a background job to the foreground. The number of the background job to bring forward is the parameter of **fg**.

```
[paul@RHEL5 ~]$ jobs
[1]  Running                  sleep 1000 &
[2]-  Running                  sleep 1000 &
[3]+  Running                  sleep 2000 &
[paul@RHEL5 ~]$ fg 3
sleep 2000
```

bg

Jobs that are **suspended** in background can be started in background with **bg**. The **bg** will send a **SIGCONT** signal.

Below an example of the sleep command (suspended with **Ctrl-Z**) being reactivated in background with **bg**.

```
[paul@RHEL5 ~]$ jobs
[paul@RHEL5 ~]$ sleep 5000 &
[1] 6702
[paul@RHEL5 ~]$ sleep 3000

[2]+  Stopped                  sleep 3000
[paul@RHEL5 ~]$ jobs
[1]-  Running                  sleep 5000 &
[2]+  Stopped                  sleep 3000
[paul@RHEL5 ~]$ bg 2
[2]+  sleep 3000 &
[paul@RHEL5 ~]$ jobs
[1]-  Running                  sleep 5000 &
[2]+  Running                  sleep 3000 &
[paul@RHEL5 ~]$
```

31.2. practice : background processes

1. Use the **jobs** command to verify whether you have any processes running in background.
2. Use **vi** to create a little text file. Suspend **vi** in background.
3. Verify with **jobs** that **vi** is suspended in background.
4. Start **find / > allfiles.txt 2>/dev/null** in foreground. Suspend it in background before it finishes.
5. Start two long **sleep** processes in background.
6. Display all **jobs** in background.
7. Use the **kill** command to suspend the last **sleep** process.
8. Continue the **find** process in background (make sure it runs again).
9. Put one of the **sleep** commands back in foreground.
10. (if time permits, a general review question...) Explain in detail where the numbers come from in the next screenshot. When are the variables replaced by their value ? By which shell ?

```
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
[paul@RHEL4b ~]$ bash -c "echo $$ $PPID"
4224 4223
[paul@RHEL4b ~]$ bash -c 'echo $$ $PPID'
5059 4224
[paul@RHEL4b ~]$ bash -c `echo $$ $PPID`
4223: 4224: command not found
```

31.3. solution : background processes

1. Use the **jobs** command to verify whether you have any processes running in background.

```
jobs (maybe the catfun is still running?)
```

2. Use **vi** to create a little text file. Suspend **vi** in background.

```
vi text.txt  
(inside vi press ctrl-z)
```

3. Verify with **jobs** that **vi** is suspended in background.

```
[paul@rhel53 ~]$ jobs  
[1]+  Stopped                  vim text.txt
```

4. Start **find / > allfiles.txt 2>/dev/null** in foreground. Suspend it in background before it finishes.

```
[paul@rhel53 ~]$ find / > allfiles.txt 2>/dev/null  
(press ctrl-z)  
[2]+  Stopped                  find / > allfiles.txt 2> /dev/null
```

5. Start two long **sleep** processes in background.

```
sleep 4000 & ; sleep 5000 &
```

6. Display all **jobs** in background.

```
[paul@rhel53 ~]$ jobs  
[1]-  Stopped                  vim text.txt  
[2]+  Stopped                  find / > allfiles.txt 2> /dev/null  
[3]   Running                  sleep 4000 &  
[4]   Running                  sleep 5000 &
```

7. Use the **kill** command to suspend the last **sleep** process.

```
[paul@rhel53 ~]$ kill -SIGSTOP 4519  
[paul@rhel53 ~]$ jobs  
[1]   Stopped                  vim text.txt  
[2]-  Stopped                  find / > allfiles.txt 2> /dev/null  
[3]   Running                  sleep 4000 &  
[4]+  Stopped                  sleep 5000
```

8. Continue the **find** process in background (make sure it runs again).

```
bg 2 (verify the job-id in your jobs list)
```

9. Put one of the **sleep** commands back in foreground.

```
fg 3 (again verify your job-id)
```

10. (if time permits, a general review question...) Explain in detail where the numbers come from in the next screenshot. When are the variables replaced by their value ? By which shell ?


```
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
[paul@RHEL4b ~]$ bash -c "echo $$ $PPID"
4224 4223
[paul@RHEL4b ~]$ bash -c 'echo $$ $PPID'
5059 4224
[paul@RHEL4b ~]$ bash -c `echo $$ $PPID`
4223: 4224: command not found
```

The current bash shell will replace the \$\$ and \$PPID while scanning the line, and before executing the echo command.

```
[paul@RHEL4b ~]$ echo $$ $PPID
4224 4223
```

The variables are now double quoted, but the current bash shell will replace \$\$ and \$PPID while scanning the line, and before executing the bash -c command.

```
[paul@RHEL4b ~]$ bash -c "echo $$ $PPID"
4224 4223
```

The variables are now single quoted. The current bash shell will **not** replace the \$\$ and the \$PPID. The bash -c command will be executed before the variables replaced with their value. This latter bash is the one replacing the \$\$ and \$PPID with their value.

```
[paul@RHEL4b ~]$ bash -c 'echo $$ $PPID'
5059 4224
```

With backticks the shell will still replace both variable before the embedded echo is executed. The result of this echo is the two process id's. These are given as commands to bash -c. But two numbers are not commands!

```
[paul@RHEL4b ~]$ bash -c `echo $$ $PPID`
4223: 4224: command not found
```

Part X. disk management

Chapter 32. disk devices

Table of Contents

32.1. terminology	270
32.2. device naming	272
32.3. discovering disk devices	273
32.4. erasing a hard disk	277
32.5. advanced hard disk settings	278
32.6. practice: hard disk devices	279
32.7. solution: hard disk devices	280

This chapter teaches you how to locate and recognise **hard disk devices**. This prepares you for the next chapter, where we put **partitions** on these devices.

32.1. terminology

platter, head, track, cylinder, sector

Data is commonly stored on magnetic or optical **disk platters**. The platters are rotated (at high speeds). Data is read by **heads**, which are very close to the surface of the platter, without touching it! The heads are mounted on an arm (sometimes called a comb or a fork).

Data is written in concentric circles called **tracks**. Track zero is (usually) on the outside. The time it takes to position the head over a certain track is called the **seek time**. Often the platters are stacked on top of each other, hence the set of tracks accessible at a certain position of the comb forms a **cylinder**. Tracks are divided into 512 byte **sectors**, with more unused space (**gap**) between the sectors on the outside of the platter.

When you break down the advertised **access time** of a hard drive, you will notice that most of that time is taken by movement of the heads (about 65%) and **rotational latency** (about 30%).

block device

Random access hard disk devices have an abstraction layer called **block device** to enable formatting in fixed-size (usually 512 bytes) blocks. Blocks can be accessed independent of access to other blocks. A block device has the letter **b** to denote the file type in the output of **ls -l**.

```
[root@RHEL4b ~]# ls -l /dev/sda*
brw-rw---- 1 root disk 8, 0 Aug  4 22:55 /dev/sda
brw-rw---- 1 root disk 8, 1 Aug  4 22:55 /dev/sda1
brw-rw---- 1 root disk 8, 2 Aug  4 22:55 /dev/sda2
[root@RHEL4b ~]#
```

Note that a **character device** is a constant stream of characters, being denoted by a **c** in **ls -l**.

Note also that the **ISO 9660** standard for cdrom uses a **2048 byte** block size.

Old hard disks (and floppy disks) use **cylinder-head-sector** addressing to access a sector on the disk. Most current disks use **LBA (Logical Block Addressing)**.

ide or scsi

Actually, the title should be **ata** or **scsi**, since ide is an ata compatible device. Most desktops use **ata devices**, most servers use **scsi**.

ata

An **ata controller** allows two devices per bus, one **master** and one **slave**. Unless your controller and devices support **cable select**, you have to set this manually with jumpers.

With the introduction of **sata** (serial ata), the original ata was renamed to **parallel ata**. Optical drives often use **atapi**, which is an ATA interface using the SCSI communication protocol.

scsi

A **scsi controller** allows more than two devices. When using **SCSI (small computer system interface)**, each device gets a unique **scsi id**. The **scsi controller** also needs a **scsi id**, do not use this id for a scsi-attached device.

Older 8-bit SCSI is now called **narrow**, whereas 16-bit is **wide**. When the bus speeds was doubled to 10Mhz, this was known as **fast SCSI**. Doubling to 20Mhz made it **ultra SCSI**. Take a look at <http://en.wikipedia.org/wiki/SCSI> for more SCSI standards.

32.2. device naming

ata (ide) device naming

All **ata** drives on your system will start with **/dev/hd** followed by a unit letter. The master hdd on the first **ata controller** is **/dev/hda**, the slave is **/dev/hdb**. For the second controller, the names of the devices are **/dev/hdc** and **/dev/hdd**.

Table 32.1. ide device naming

controller	connection	device name
ide0	master	/dev/hda
	slave	/dev/hdb
ide1	master	/dev/hdc
	slave	/dev/hdd

It is possible to have only **/dev/hda** and **/dev/hdd**. The first one is a single ata hard disk, the second one is the cdrom (by default configured as slave).

scsi device naming

scsi drives follow a similar scheme, but all start with **/dev/sd**. When you run out of letters (after **/dev/sdz**), you can continue with **/dev/sdaa** and **/dev/sdab** and so on. (We will see later on that **lvm** volumes are commonly seen as **/dev/md0**, **/dev/md1** etc.)

Below a **sample** of how scsi devices on a linux can be named. Adding a scsi disk or raid controller with a lower scsi address will change the naming scheme (shifting the higher scsi addresses one letter further in the alphabet).

Table 32.2. scsi device naming

device	scsi id	device name
disk 0	0	/dev/sda
disk 1	1	/dev/sdb
raid controller 0	5	/dev/sdc
raid controller 1	6	/dev/sdd

32.3. discovering disk devices

/sbin/fdisk

You can start by using `/sbin/fdisk` to find out what kind of disks are seen by the kernel. Below the result on Debian, with two **ata-ide disks** present.

```
root@barry:~# fdisk -l | grep Disk
Disk /dev/hda: 60.0 GB, 60022480896 bytes
Disk /dev/hdb: 81.9 GB, 81964302336 bytes
```

And here an example of **sata disks** on a laptop with Ubuntu. Remember that **sata** disks are presented to you with the **scsi** `/dev/sdx` notation.

```
root@laika:~# fdisk -l | grep Disk
Disk /dev/sda: 100.0 GB, 100030242816 bytes
Disk /dev/sdb: 100.0 GB, 100030242816 bytes
```

Here is an overview of disks on a RHEL4u3 server with two real 72GB **scsi disks**. This server is attached to a **NAS** with four **NAS disks** of half a terabyte. On the NAS disks, four LVM (`/dev/mdx`) software RAID devices are configured.

```
[root@tsvtl1 ~]# fdisk -l | grep Disk
Disk /dev/sda: 73.4 GB, 73407488000 bytes
Disk /dev/sdb: 73.4 GB, 73407488000 bytes
Disk /dev/sdc: 499.0 GB, 499036192768 bytes
Disk /dev/sdd: 499.0 GB, 499036192768 bytes
Disk /dev/sde: 499.0 GB, 499036192768 bytes
Disk /dev/sdf: 499.0 GB, 499036192768 bytes
Disk /dev/md0: 271 MB, 271319040 bytes
Disk /dev/md2: 21.4 GB, 21476081664 bytes
Disk /dev/md3: 21.4 GB, 21467889664 bytes
Disk /dev/md1: 21.4 GB, 21476081664 bytes
```

You can also use `fdisk` to obtain information about one specific hard disk device.

```
[root@rhel4 ~]# fdisk -l /dev/sda

Disk /dev/sda: 12.8 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1    *           1          13        104391   83  Linux
/dev/sda2             14         1566       12474472+  8e  Linux LVM
```

Later we will use `fdisk` to do dangerous stuff like creating and deleting partitions.

/bin/dmesg

Kernel boot messages can be seen after boot with **dmesg**. Since hard disk devices are detected by the kernel during boot, you can also use **dmesg** to find information about disk devices.

```
root@barry:~# dmesg | grep "[hs]d[a-z]"
Kernel command line: root=/dev/hda1 ro
    ide0: BM-DMA at 0xfc00-0xfc07, BIOS settings: hda:DMA, hdb:DMA
    ide1: BM-DMA at 0xfc08-0xfc0f, BIOS settings: hdc:DMA, hdd:DMA
hda: ST360021A, ATA DISK drive
hdb: Maxtor 6Y080L0, ATA DISK drive
hdc: SONY DVD RW DRU-510A, ATAPI CD/DVD-ROM drive
hdd: SONY DVD RW DRU-810A, ATAPI CD/DVD-ROM drive
hda: max request size: 128KiB
hda: 117231408 sectors (60022 MB) w/2048KiB Cache, CHS=65535/16/63, UDMA
    hda: hda1 hda2
hdb: max request size: 128KiB
hdb: 160086528 sectors (81964 MB) w/2048KiB Cache, CHS=65535/16/63, UDMA
    hdb: hdb1 hdb2
hdc: ATAPI 32X DVD-ROM DVD-R CD-R/RW drive, 8192kB Cache, UDMA(33)
hdd: ATAPI 40X DVD-ROM DVD-R CD-R/RW drive, 2048kB Cache, UDMA(33)
...
```

Here's another example of **dmesg** (same computer as above, but with extra 200gb disk now).

```
paul@barry:~$ dmesg | grep -i "ata disk"
[ 2.624149] hda: ST360021A, ATA DISK drive
[ 2.904150] hdb: Maxtor 6Y080L0, ATA DISK drive
[ 3.472148] hdd: WDC WD2000BB-98DWA0, ATA DISK drive
```

Third and last example of **dmesg** running on RHEL5.3.

```
root@rhel53 ~# dmesg | grep -i "scsi disk"
sd 0:0:2:0: Attached scsi disk sda
sd 0:0:3:0: Attached scsi disk sdb
sd 0:0:6:0: Attached scsi disk sdc
```

/sbin/lshw

The **lshw** tool will **list hardware**. With the right options **lshw** can show a lot of information about disks (and partitions).

Below a truncated screenshot on Debian 5:

```
root@debian5:~# aptitude search lshw
p lshw - information about hardware configuration
p lshw-gtk - information about hardware configuration
root@debian5:~# aptitude install lshw
...
root@debian5:~# lshw -class volume
*-volume:0
```



```
description: EXT3 volume
vendor: Linux
physical id: 1
bus info: ide@0.0,1
logical name: /dev/hda1
logical name: /
version: 1.0
serial: f327ca8a-8187-48c5-b760-956ec79d414b
size: 19GiB
capacity: 19GiB
capabilities: primary bootable journaled extended_attributes lar\
ge_files huge_files recover ext3 ext2 initialized
configuration: created=2009-10-28 12:02:35 filesystem=ext3 ...
...
```

Below a screenshot of **lshw** running Ubuntu 10.10 on a macbook pro:

```
root@ubul010:~# lshw -class volume
*-volume:0 UNCLAIMED
  description: EFI GPT partition
  physical id: 1
  bus info: scsi@0:0.0.0,1
  capacity: 2047KiB
  capabilities: primary nofs
*-volume:1
  description: EXT4 volume
  vendor: Linux
  physical id: 2
  bus info: scsi@0:0.0.0,2
  logical name: /dev/sda2
  logical name: /
  version: 1.0
  serial: 101eb20f-3e25-4900-b988-4622c0ee4ff5
  size: 142GiB
  capacity: 142GiB
...
```

/sbin/lsscsi

The **/sbin/lsscsi** will give you a nice readable output of all scsi (and scsi emulated devices). This first screenshot shows lsscsi on a SPARC system.

```
root@shaka:~# lsscsi
[0:0:0:0]    disk    Adaptec  RAID5          V1.0  /dev/sda
[1:0:0:0]    disk    SEAGATE  ST336605FSUN36G  0438  /dev/sdb
root@shaka:~#
```

Here is the same command, but run on a laptop with scsi emulated dvd writer and scsi emulated usb.

```
paul@laika:~$ lsscsi
[0:0:0:0]    disk    ATA      HTS721010G9SA00  MCZO  /dev/sda
[1:0:0:0]    disk    ATA      HTS721010G9SA00  MCZO  /dev/sdb
[3:0:0:0]    cd/dvd  _NEC    DVD_RW ND-7551A  1-02  /dev/scd0
[4:0:0:0]    disk    GENERIC  USB Storage-CFC  019A  /dev/sdc
[4:0:0:1]    disk    GENERIC  USB Storage-SDC  019A  /dev/sdd
[4:0:0:2]    disk    GENERIC  USB Storage-SMC  019A  /dev/sde
[4:0:0:3]    disk    GENERIC  USB Storage-MSC  019A  /dev/sdf
```

/proc/scsi/scsi

Another way to locate **scsi** devices is via the **/proc/scsi/scsi** file.

```
root@shaka:~# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: Adaptec Model: RAID5 Rev: V1.0
  Type: Direct-Access ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: SEAGATE Model: ST336605FSUN36G Rev: 0438
  Type: Direct-Access ANSI SCSI revision: 03
root@shaka:~#
```

/sbin/scsi_info and /sbin/scsiinfo

There is also a **scsi_info** command, but this is not always installed by default.

```
root@shaka:~# scsi_info /dev/sdb
SCSI_ID="0,0,0"
HOST="1"
MODEL="SEAGATE ST336605FSUN36G"
FW_REV="0438"
root@shaka:~#
```

Another simple tool is **scsiinfo** which is a part of **scsitools** (also not installed by default).

```
root@debian5:~# scsiinfo -l
/dev/sda /dev/sdb /dev/sdc
```

32.4. erasing a hard disk

Before selling your old hard disk on the internet, it might be a good idea to erase it. By simply repartitioning, by using the Microsoft Windows format utility, or even after an **mkfs** command, some people will still be able to read most of the data on the disk.

Although technically the **/sbin/badblocks** tool is meant to look for bad blocks, you can use it to completely erase all data from a disk. Since this is really writing to every sector of the disk, it can take a long time!

```
root@RHELv4u2:~# badblocks -ws /dev/sdb
Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
```

32.5. advanced hard disk settings

Tweaking of hard disk settings (dma, gap, ...) are not covered in this course. Several tools exist, **hdparm** and **sdparm** are two of them.

/sbin/hdparm can be used to display or set information and parameters about an ATA (or SATA) hard disk device. The **-i** and **-I** options will give you even more information about the physical properties of the device.

```
root@laika:~# hdparm /dev/sdb

/dev/sdb:
IO_support    = 0 (default 16-bit)
readonly     = 0 (off)
readahead    = 256 (on)
geometry     = 12161/255/63, sectors = 195371568, start = 0
```

Below **hdparm** info about a 200GB IDE disk.

```
root@barry:~# hdparm /dev/hdd

/dev/hdd:
multcount    = 0 (off)
IO_support   = 0 (default)
unmaskirq    = 0 (off)
using_dma    = 1 (on)
keepsettings = 0 (off)
readonly     = 0 (off)
readahead    = 256 (on)
geometry     = 24321/255/63, sectors = 390721968, start = 0
```

Here a screenshot of **sdparm** on Ubuntu 10.10.

```
root@ubul010:~# aptitude install sdparm
...
root@ubul010:~# sdparm /dev/sda | head -1
/dev/sda: ATA FUJITSU MJA2160B 0081
root@ubul010:~# man sdparm
```

Use **hdparm** and **sdparm** with care.

32.6. practice: hard disk devices

About this lab: To practice working with hard disks, you will need some hard disks. When there are no physical hard disk available, you can use virtual disks in **vmware** or **VirtualBox**. The teacher will help you in attaching a couple of ATA and/or SCSI disks to a virtual machine. The results of this lab can be used in the next three labs (partitions, file systems, mounting). It is advised to attach at least one **ide** and three equally sized **scsi** disks to the virtual machine.

1. Use **dmesg** to make a list of hard disk devices detected at boot-up.
2. Use **fdisk** to find the total size of all hard disk devices on your system.
3. Stop a virtual machine, add three virtual 1 gigabyte **scsi** hard disk devices and one virtual 400 megabyte **ide** hard disk device. If possible, also add another virtual 400 megabyte **ide** disk.
4. Use **dmesg** to verify that all the new disks are properly detected at boot-up.
5. Verify that you can see the disk devices in **/dev**.
6. Use **fdisk** (with **grep** and **/dev/null**) to display the total size of the new disks.
7. Use **badblocks** to completely erase one of the smaller hard disks.
8. Look at **/proc/scsi/scsi**.
9. If possible, install **lsscsi**, **lshw** and use them to list the disks.

32.7. solution: hard disk devices

1. Use **dmesg** to make a list of hard disk devices detected at boot-up.

Some possible answers...

```
dmesg | grep -i disk
```

```
Looking for ATA disks: dmesg | grep hd[abcd]
```

```
Looking for ATA disks: dmesg | grep -i "ata disk"
```

```
Looking for SCSI disks: dmesg | grep sd[a-f]
```

```
Looking for SCSI disks: dmesg | grep -i "scsi disk"
```

2. Use **fdisk** to find the total size of all hard disk devices on your system.

```
fdisk -l
```

3. Stop a virtual machine, add three virtual 1 gigabyte **scsi** hard disk devices and one virtual 400 megabyte **ide** hard disk device. If possible, also add another virtual 400 megabyte **ide** disk.

This exercise happens in the settings of vmware or VirtualBox.

4. Use **dmesg** to verify that all the new disks are properly detected at boot-up.

See 1.

5. Verify that you can see the disk devices in **/dev**.

```
SCSI+SATA: ls -l /dev/sd*
```

```
ATA: ls -l /dev/hd*
```

6. Use **fdisk** (with **grep** and **/dev/null**) to display the total size of the new disks.

```
root@rhel53 ~# fdisk -l 2>/dev/null | grep [MGT]B
Disk /dev/hda: 21.4 GB, 21474836480 bytes
Disk /dev/hdb: 1073 MB, 1073741824 bytes
Disk /dev/sda: 2147 MB, 2147483648 bytes
Disk /dev/sdb: 2147 MB, 2147483648 bytes
Disk /dev/sdc: 2147 MB, 2147483648 bytes
```

7. Use **badblocks** to completely erase one of the smaller hard disks.

```
#Verify the device (/dev/sdc??) you want to erase before typing this.
#
root@rhel53 ~# badblocks -ws /dev/sdc
Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
```

8. Look at **/proc/scsi/scsi**.

```
root@rhel53 ~# cat /proc/scsi/scsi
```

Attached devices:

```
Host: scsi0 Channel: 00 Id: 02 Lun: 00
  Vendor: VBOX      Model: HARDDISK      Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi0 Channel: 00 Id: 03 Lun: 00
  Vendor: VBOX      Model: HARDDISK      Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
Host: scsi0 Channel: 00 Id: 06 Lun: 00
  Vendor: VBOX      Model: HARDDISK      Rev: 1.0
  Type:   Direct-Access                    ANSI SCSI revision: 05
```

9. If possible, install **lsscsi**, **lshw** and use them to list the disks.

Debian,Ubuntu: aptitude install lsscsi lshw

Fedora: yum install lsscsi lshw

```
root@rhel53 ~# lsscsi
[0:0:2:0]   disk      VBOX      HARDDISK      1.0    /dev/sda
[0:0:3:0]   disk      VBOX      HARDDISK      1.0    /dev/sdb
[0:0:6:0]   disk      VBOX      HARDDISK      1.0    /dev/sdc
```

Chapter 33. disk partitions

Table of Contents

33.1. about partitions	283
33.2. discovering partitions	284
33.3. partitioning new disks	285
33.4. about the partition table	287
33.5. practice: partitions	288
33.6. solution: partitions	289

This chapter continues on the **hard disk devices** from the previous one. Here we will put **partitions** on those devices.

This chapter prepares you for the next chapter, where we put **file systems** on our partitions.

33.1. about partitions

primary, extended and logical

Linux requires you to create one or more **partitions**. The next paragraphs will explain how to create and use partitions.

A partition's **geometry** and size is usually defined by a starting and ending cylinder (sometimes by sector). Partitions can be of type **primary** (maximum four), **extended** (maximum one) or **logical** (contained within the extended partition). Each partition has a **type field** that contains a code. This determines the computers operating system or the partitions file system.

Table 33.1. primary, extended and logical partitions

Partition Type	naming
Primary (max 4)	1-4
Extended (max 1)	1-4
Logical	5-

partition naming

We saw before that hard disk devices are named `/dev/hdx` or `/dev/sdx` with `x` depending on the hardware configuration. Next is the partition number, starting the count at 1. Hence the four (possible) primary partitions are numbered 1 to 4. Logical partition counting always starts at 5. Thus `/dev/hda2` is the second partition on the first ATA hard disk device, and `/dev/hdb5` is the first logical partition on the second ATA hard disk device. Same for SCSI, `/dev/sdb3` is the third partition on the second SCSI disk.

Table 33.2. Partition naming

partition	device
<code>/dev/hda1</code>	first primary partition on <code>/dev/hda</code>
<code>/dev/hda2</code>	second primary or extended partition on <code>/dev/hda</code>
<code>/dev/sda5</code>	first logical drive on <code>/dev/sda</code>
<code>/dev/sdb6</code>	second logical on <code>/dev/sdb</code>

33.2. discovering partitions

fdisk -l

In the **fdisk -l** example below you can see that two partitions exist on **/dev/sdb**. The first partition spans 31 cylinders and contains a Linux swap partition. The second partition is much bigger.

```
root@laika:~# fdisk -l /dev/sdb

Disk /dev/sdb: 100.0 GB, 100030242816 bytes
255 heads, 63 sectors/track, 12161 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            1           31     248976   82  Linux swap / Solaris
/dev/sdb2           32        12161    97434225   83  Linux
root@laika:~#
```

/proc/partitions

The **/proc/partitions** file contains a table with major and minor number of partitioned devices, their number of blocks and the device name in **/dev**. Verify with **/proc/devices** to link the major number to the proper device.

```
paul@RHELv4u4:~$ cat /proc/partitions
major minor #blocks name

   3     0     524288 hda
   3    64     734003 hdb
   8     0    8388608 sda
   8     1     104391 sda1
   8     2     8281507 sda2
   8    16     1048576 sdb
   8    32     1048576 sdc
   8    48     1048576 sdd
  253     0    7176192 dm-0
  253     1    1048576 dm-1
```

The **major** number corresponds to the device type (or driver) and can be found in **/proc/devices**. In this case 3 corresponds to **ide** and 8 to **sd**. The **major** number determines the **device driver** to be used with this device.

The **minor** number is a unique identification of an instance of this device type. The **devices.txt** file in the kernel tree contains a full list of major and minor numbers.

other tools

You might be interested in alternatives to **fdisk** like **parted**, **cfdisk**, **sfdisk** and **gparted**. This course mainly uses **fdisk** to partition hard disks.

33.3. partitioning new disks

In the example below, we bought a new disk for our system. After the new hardware is properly attached, you can use **fdisk** and **parted** to create the necessary partition(s). This example uses **fdisk**, but there is nothing wrong with using **parted**.

recognising the disk

First, we check with **fdisk -l** whether Linux can see the new disk. Yes it does, the new disk is seen as `/dev/sdb`, but it does not have any partitions yet.

```
root@RHELv4u2:~# fdisk -l

Disk /dev/sda: 12.8 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start         End      Blocks   Id  System
/dev/sda1  *           1          13        104391   83  Linux
/dev/sda2                14         1566       12474472+  8e  Linux LVM

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Disk /dev/sdb doesn't contain a valid partition table
```

opening the disk with fdisk

Then we create a partition with **fdisk** on `/dev/sdb`. First we start the **fdisk** tool with `/dev/sdb` as argument. Be very very careful not to partition the wrong disk!!

```
root@RHELv4u2:~# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI...
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected...
```

empty partition table

Inside the **fdisk** tool, we can issue the **p** command to see the current disks partition table.

```
Command (m for help): p

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------

create a new partition

No partitions exist yet, so we issue **n** to create a new partition. We choose **p** for primary, **1** for the partition number, **1** for the start cylinder and **14** for the end cylinder.

```
Command (m for help): n
Command action
e   extended
p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-130, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-130, default 130): 14
```

We can now issue **p** again to verify our changes, but they are not yet written to disk. This means we can still cancel this operation! But it looks good, so we use **w** to write the changes to disk, and then quit the **fdisk** tool.

```
Command (m for help): p

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start          End      Blocks   Id  System
/dev/sdb1        1             14      112423+   83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@RHELv4u2:~#
```

display the new partition

Let's verify again with **fdisk -l** to make sure reality fits our dreams. Indeed, the screenshot below now shows a partition on **/dev/sdb**.

```
root@RHELv4u2:~# fdisk -l

Disk /dev/sda: 12.8 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start          End      Blocks   Id  System
/dev/sda1        *           1           13      104391   83  Linux
/dev/sda2                14          1566     12474472+  8e  Linux LVM

Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot      Start          End      Blocks   Id  System
```

```
/dev/sdb1          1          14          112423+  83  Linux
root@RHELv4u2:~#
```

33.4. about the partition table

master boot record

The **partition table** information (primary and extended partitions) is written in the **master boot record** or **mbr**. You can use **dd** to copy the mbr to a file.

This example copies the master boot record from the first SCSI hard disk.

```
dd if=/dev/sda of=/SCSIidisk.mbr bs=512 count=1
```

The same tool can also be used to wipe out all information about partitions on a disk. This example writes zeroes over the master boot record.

```
dd if=/dev/zero of=/dev/sda bs=512 count=1
```

Or to wipe out the whole partition or disk.

```
dd if=/dev/zero of=/dev/sda
```

partprobe

Don't forget that after restoring a **master boot record** with **dd**, that you need to force the kernel to reread the partition table with **partprobe**. After running **partprobe**, the partitions can be used again.

```
[root@RHEL5 ~]# partprobe
[root@RHEL5 ~]#
```

logical drives

The **partition table** does not contain information about **logical drives**. So the **dd** backup of the **mbr** only works for primary and extended partitions. To backup the partition table including the logical drives, you can use **sfdisk**.

This example shows how to backup all partition and logical drive information to a file.

```
sfdisk -d /dev/sda > parttable.sda.sfdisk
```

The following example copies the **mbr** and all **logical drive** info from /dev/sda to /dev/sdb.

```
sfdisk -d /dev/sda | sfdisk /dev/sdb
```

33.5. practice: partitions

1. Use **fdisk -l** to display existing partitions and sizes.
2. Use **df -h** to display existing partitions and sizes.
3. Compare the output of **fdisk** and **df**.
4. Create a 200MB primary partition on a small disk.
5. Create a 400MB primary partition and two 300MB logical drives on a big disk.
6. Use **df -h** and **fdisk -l** to verify your work.
7. Compare the output again of **fdisk** and **df**. Do both commands display the new partitions ?
8. Create a backup with **dd** of the **mbr** that contains your 200MB primary partition.
9. Take a backup of the **partition table** containing your 400MB primary and 300MB logical drives. Make sure the logical drives are in the backup.
10. (optional) Remove all your partitions with **fdisk**. Then restore your backups.

33.6. solution: partitions

1. Use **fdisk -l** to display existing partitions and sizes.

```
as root: # fdisk -l
```

2. Use **df -h** to display existing partitions and sizes.

```
df -h
```

3. Compare the output of **fdisk** and **df**.

Some partitions will be listed in both outputs (maybe /dev/sda1 or /dev/hda1).

4. Create a 200MB primary partition on a small disk.

Choose one of the disks you added (this example uses /dev/sdc).

```
root@rhel53 ~# fdisk /dev/sdc
```

```
...
```

```
Command (m for help): n
```

```
Command action
```

```
  e   extended
```

```
  p   primary partition (1-4)
```

```
p
```

```
Partition number (1-4): 1
```

```
First cylinder (1-261, default 1): 1
```

```
Last cylinder or +size or +sizeM or +sizeK (1-261, default 261): +200m
```

```
Command (m for help): w
```

```
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
```

```
Syncing disks.
```

5. Create a 400MB primary partition and two 300MB logical drives on a big disk.

Choose one of the disks you added (this example uses /dev/sdb)

```
fdisk /dev/sdb
```

```
inside fdisk : n p 1 +400m enter --- n e 2 enter enter --- n l +300m (twice)
```

6. Use **df -h** and **fdisk -l** to verify your work.

```
fdisk -l ; df -h
```

7. Compare the output again of **fdisk** and **df**. Do both commands display the new partitions ?

The newly created partitions are visible with **fdisk**.

But they are not displayed by **df**.

8. Create a backup with **dd** of the **mbr** that contains your 200MB primary partition.

```
dd if=/dev/sdc of=bootsector.sdc.dd count=1 bs=512
```

9. Take a backup of the **partition table** containing your 400MB primary and 300MB logical drives. Make sure the logical drives are in the backup.

```
sfdisk -d /dev/sdb > parttable.sdb.sfdisk
```

Chapter 34. file systems

Table of Contents

34.1. about file systems	291
34.2. common file systems	291
34.3. putting a file system on a partition	294
34.4. tuning a file system	294
34.5. checking a file system	295
34.6. practice: file systems	296
34.7. solution: file systems	297

When you are finished partitioning the hard disk, you can put a **file system** on each partition.

This chapter builds on the **partitions** from the previous chapter, and prepares you for the next one where we will **mount** the filesystems.

34.1. about file systems

A file system is a way of organizing files on your partition. Besides file-based storage, file systems usually include **directories** and **access control**, and contain meta information about files like access times, modification times and file ownership.

The properties (length, character set, ...) of filenames are determined by the file system you choose. Directories are usually implemented as files, you will have to learn how this is implemented! Access control in file systems is tracked by user ownership (and group owner- and membership) in combination with one or more access control lists.

The manual page about filesystems(5) is usually accessed by typing **man fs**. You can also look at **/proc/filesystems** for currently loaded file system drivers.

```
root@rhel153 ~# cat /proc/filesystems | grep -v nodev
ext2
iso9660
ext3
```

34.2. common file systems

ext2 and ext3

Once the most common Linux file systems is the **ext2** (the second extended) file system. A disadvantage is that file system checks on ext2 can take a long time. You will see that ext2 is being replaced by **ext3** on most Linux machines. They are essentially the same, except for the **journaling** which is only present in ext3.

Journaling means that changes are first written to a journal on the disk. The journal is flushed regularly, writing the changes in the file system. Journaling keeps the file system in a consistent state, so you don't need a file system check after an unclean shutdown or power failure.

You can create these file systems with the **/sbin/mkfs** or **/sbin/mke2fs** commands. Use **mke2fs -j** to create an ext3 file system. You can convert an ext2 to ext3 with **tune2fs -j**. You can mount an ext3 file system as ext2, but then you lose the journaling. Do not forget to run **mkinitrd** if you are booting from this device.

ext4

Since 2008 the newest incarnation of the ext file system is **ext4** is available in the Linux kernel. **ext4** support larger files (up to 16 terabyte) and larger file systems than **ext3** (and many more features).

vfat

The **vfat** file system exists in a couple of forms : **fat12** for floppy disks, **fat16** on **ms-dos**, and **fat32** for larger disks. The Linux **vfat** implementation supports all of these, but vfat lacks a lot of features like security and links. **fat** disks can be read by every operating system, and are used a lot for digital cameras, **usb** sticks and to exchange data between different OS'ses on a home user's computer.

iso 9660

iso 9660 is the standard format for cdroms. Chances are you will encounter this file system also on your hard disk in the form of images of cdroms (often with the .iso extension). The **iso 9660** standard limits filenames to the 8.3 format. The Unix world didn't like this, and thus added the **rock ridge** extensions, which allows for filenames up to 255 characters and Unix-style file-modes, ownership and symbolic links. Another extensions to **iso 9660** is **joliet**, which adds 64 unicode characters to the filename. The **el torito** standard extends **iso 9660** to be able to boot from CD-ROM's.

udf

Most optical media today (including cd's and dvd's) use **udf**, the Universal Disk Format.

swap

All things considered, swap is not a file system. But to use a partition as a **swap partition** it must be formatted and mounted as swap space.

others...

You might encounter **reiserfs** on older Linux systems. Maybe you will see Sun's **zfs** or the open source **btrfs**. This last one requires a chapter on itself.

/proc/filesystems

The **/proc/filesystems** file displays a list of supported file systems. When you mount a file system without explicitly defining one, then mount will first try to probe **/etc/filesystems** and then probe **/proc/filesystems** for all the filesystems without the **nodev** label. If **/etc/filesystems** ends with a line containing only an asterisk (*) then both files are probed.

```
paul@RHELv4u4:~$ cat /proc/filesystems
nodev    sysfs
```

```
nodev rootfs
nodev bdev
nodev proc
nodev sockfs
nodev binfmt_misc
nodev usbfs
nodev usbdevfs
nodev futexfs
nodev tmpfs
nodev pipefs
nodev eventpollfs
nodev devpts
nodev ext2
nodev ramfs
nodev hugetlbfs
nodev iso9660
nodev relayfs
nodev mqueue
nodev selinuxfs
nodev ext3
nodev rpc_pipefs
nodev vmware-hgfs
nodev autofs
paul@RHELv4u4:~$
```

34.3. putting a file system on a partition

We now have a fresh partition. The system binaries to make file systems can be found with `ls`.

```
[root@RHEL4b ~]# ls -lS /sbin/mk*
-rwxr-xr-x 3 root root 34832 Apr 24 2006 /sbin/mke2fs
-rwxr-xr-x 3 root root 34832 Apr 24 2006 /sbin/mkfs.ext2
-rwxr-xr-x 3 root root 34832 Apr 24 2006 /sbin/mkfs.ext3
-rwxr-xr-x 3 root root 28484 Oct 13 2004 /sbin/mkdosfs
-rwxr-xr-x 3 root root 28484 Oct 13 2004 /sbin/mkfs.msdos
-rwxr-xr-x 3 root root 28484 Oct 13 2004 /sbin/mkfs.vfat
-rwxr-xr-x 1 root root 20313 Apr 10 2006 /sbin/mkinitrd
-rwxr-x--- 1 root root 15444 Oct 5 2004 /sbin/mkzonedb
-rwxr-xr-x 1 root root 15300 May 24 2006 /sbin/mkfs.cramfs
-rwxr-xr-x 1 root root 13036 May 24 2006 /sbin/mkswap
-rwxr-xr-x 1 root root 6912 May 24 2006 /sbin/mkfs
-rwxr-xr-x 1 root root 5905 Aug 3 2004 /sbin/mkbootdisk
[root@RHEL4b ~]#
```

It is time for you to read the manual pages of `mkfs` and `mke2fs`. In the example below, you see the creation of an **ext2 file system** on `/dev/sdb1`. In real life, you might want to use options like `-m0` and `-j`.

```
root@RHELv4u2:~# mke2fs /dev/sdb1
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
28112 inodes, 112420 blocks
5621 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
14 block groups
8192 blocks per group, 8192 fragments per group
2008 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

34.4. tuning a file system

You can use `tune2fs` to list and set file system settings. The first screenshot lists the reserved space for root (which is set at five percent).

```
[root@rhel4 ~]# tune2fs -l /dev/sda1 | grep -i "block count"
Block count:          104388
Reserved block count: 5219
[root@rhel4 ~]#
```

This example changes this value to ten percent. You can use `tune2fs` while the file system is active, even if it is the root file system (as in this example).

```
[root@rhel4 ~]# tune2fs -m10 /dev/sda1
tune2fs 1.35 (28-Feb-2004)
Setting reserved blocks percentage to 10 (10430 blocks)
[root@rhel4 ~]# tune2fs -l /dev/sda1 | grep -i "block count"
Block count:          104388
Reserved block count: 10430
[root@rhel4 ~]#
```

34.5. checking a file system

The **fsck** command is a front end tool used to check a file system for errors.

```
[root@RHEL4b ~]# ls /sbin/*fsck*
/sbin/dosfsck  /sbin/fsck          /sbin/fsck.ext2  /sbin/fsck.msdos
/sbin/e2fsck  /sbin/fsck.cramfs  /sbin/fsck.ext3  /sbin/fsck.vfat
[root@RHEL4b ~]#
```

The last column in **/etc/fstab** is used to determine whether a file system should be checked at boot-up.

```
[paul@RHEL4b ~]$ grep ext /etc/fstab
/dev/VolGroup00/LogVol100 /          ext3      defaults    1 1
LABEL=/boot              /boot     ext3      defaults    1 2
[paul@RHEL4b ~]$
```

Manually checking a mounted file system results in a warning from fsck.

```
[root@RHEL4b ~]# fsck /boot
fsck 1.35 (28-Feb-2004)
e2fsck 1.35 (28-Feb-2004)
/dev/sda1 is mounted.
```

```
WARNING!!! Running e2fsck on a mounted filesystem may cause
SEVERE filesystem damage.
```

```
Do you really want to continue (y/n)? no
```

```
check aborted.
```

But after unmounting fsck and **e2fsck** can be used to check an ext2 file system.

```
[root@RHEL4b ~]# fsck /boot
fsck 1.35 (28-Feb-2004)
e2fsck 1.35 (28-Feb-2004)
/boot: clean, 44/26104 files, 17598/104388 blocks
[root@RHEL4b ~]# fsck -p /boot
fsck 1.35 (28-Feb-2004)
/boot: clean, 44/26104 files, 17598/104388 blocks
[root@RHEL4b ~]# e2fsck -p /dev/sda1
/boot: clean, 44/26104 files, 17598/104388 blocks
```

34.6. practice: file systems

1. List the filesystems that are known by your system.
2. Create an **ext2** filesystem on the 200MB partition.
3. Create an **ext3** filesystem on the 400MB partition and one of the 300MB logical drives.
4. Set the reserved space for root on the logical drive to 0 percent.
5. Verify your work with **fdisk** and **df**.

34.7. solution: file systems

1. List the filesystems that are known by your system.

```
man fs
```

```
cat /proc/filesystems
```

```
cat /etc/filesystems (not on all Linux distributions)
```

2. Create an **ext2** filesystem on the 200MB partition.

```
mke2fs /dev/sdc1 (replace sdc1 with the correct partition)
```

3. Create an **ext3** filesystem on the 400MB partition and one of the 300MB logical drives.

```
mke2fs -j /dev/sdb1 (replace sdb1 with the correct partition)
```

```
mke2fs -j /dev/sdb5 (replace sdb5 with the correct partition)
```

4. Set the reserved space for root on the logical drive to 0 percent.

```
tune2fs -m 0 /dev/sdb5
```

5. Verify your work with **fdisk** and **df**.

```
mkfs (mke2fs) makes no difference in the output of these commands
```

```
The big change is in the next topic: mounting
```

Chapter 35. mounting

Table of Contents

35.1. mounting local file systems	299
35.2. displaying mounted file systems	300
35.3. permanent mounts	301
35.4. securing mounts	302
35.5. practice: mounting file systems	304
35.6. solution: mounting file systems	305

Once you've put a file system on a partition, you can **mount** it. Mounting a file system makes it available for use, usually as a directory. We say **mounting a file system** instead of mounting a partition because we will see later that we can also mount file systems that do not exist on partitions.

35.1. mounting local file systems

On all **Unix** systems, every file and every directory is part of one big file tree. To access a file, you need to know the full path starting from the root directory. When adding a **file system** to your computer, you need to make it available somewhere in the file tree. The directory where you make a file system available is called a **mount point**.

/bin/mkdir

This example shows how to create a new **mount point** with **mkdir**.

```
root@RHELv4u2:~# mkdir /home/project55
```

/bin/mount

When the **mount point** is created, and a **file system** is present on the partition, then **mount** can **mount** the **file system** on the **mount point directory**.

```
root@RHELv4u2:~# mount -t ext2 /dev/sdb1 /home/project55/
```

Once mounted, the new file system is accessible to users.

/etc/filesystems

Actually the explicit **-t ext2** option to set the file system is not always necessary. The **mount** command is able to automatically detect a lot of file systems.

When mounting a file system without specifying explicitly the file system, then **mount** will first probe **/etc/filesystems**. Mount will skip lines with the **nodev** directive.

```
paul@RHELv4u4:~$ cat /etc/filesystems
ext3
ext2
nodev proc
nodev devpts
iso9660
vfat
hfs
paul@RHELv4u4:~$
```

/proc/filesystems

When **/etc/filesystems** does not exist, or ends with a single ***** on the last line, then **mount** will read **/proc/filesystems**.

```
[root@RHEL52 ~]# cat /proc/filesystems | grep -v ^nodev
ext2
iso9660
ext3
```

/bin/umount

You can **umount** a mounted file system using the **umount** command.

```
root@pasha:~# umount /home/reet
```

35.2. displaying mounted file systems

To display all mounted file systems, issue the **mount** command. Or look at the files **/proc/mounts** and **/etc/mtab**.

/bin/mount

The simplest and most common way to view all mounts is by issuing the **mount** command without any arguments.

```
root@RHELv4u2:~# mount | grep /dev/sdb
/dev/sdb1 on /home/project55 type ext2 (rw)
```

/proc/mounts

The kernel provides the info in **/proc/mounts** in file form, but **/proc/mounts** does not exist as a file on any hard disk. Looking at **/proc/mounts** is looking at information that comes directly from the kernel.

```
root@RHELv4u2:~# cat /proc/mounts | grep /dev/sdb
/dev/sdb1 /home/project55 ext2 rw 0 0
```

/etc/mtab

The **/etc/mtab** file is not updated by the kernel, but is maintained by the **mount** command. Do not edit **/etc/mtab** manually.

```
root@RHELv4u2:~# cat /etc/mtab | grep /dev/sdb
/dev/sdb1 /home/project55 ext2 rw 0 0
```

/bin/df

A more user friendly way to look at mounted file systems is **df**. The **df (diskfree)** command has the added benefit of showing you the free space on each mounted disk. Like a lot of Linux commands, **df** supports the **-h** switch to make the output more **human readable**.

```
root@RHELv4u2:~# df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/mapper/VolGroup00-LogVol100
11707972 6366996 4746240 58% /
/dev/sda1              101086         9300      86567  10% /boot
none                  127988           0     127988   0% /dev/shm
/dev/sdb1              108865         1550     101694   2% /home/project55
root@RHELv4u2:~# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol100
12G  6.1G  4.6G  58% /
/dev/sda1              99M   9.1M   85M   10% /boot
none                  125M     0   125M   0% /dev/shm
/dev/sdb1              107M   1.6M  100M   2% /home/project55
```

In the **df -h** example below you can see the size, free space, used gigabytes and percentage and mount point of a partition.

```
root@laika:~# df -h | egrep -e "(sdb2|File)"
Filesystem            Size Used Avail Use% Mounted on
/dev/sdb2              92G   83G   8.6G   91% /media/sdb2
root@laika:~#
```

/bin/du

The **du** command can summarize **disk usage** for files and directories. Preventing **du** to go into subdirectories with the **-s** option will give you a total for that directory. This option is often used together with **-h**, so **du -sh** on a mount point gives the total amount used in that partition.

```
root@pasha:~# du -sh /home/reet
881G    /home/reet
```

35.3. permanent mounts

Until now, we performed all mounts manually. This works nice, until the next reboot. Luckily there is a way to tell your computer to automatically mount certain file systems during boot.

/etc/fstab

This is done using the file system table located in the **/etc/fstab** file. Below is a sample **/etc/fstab** file.

```
root@RHELv4u2:~# cat /etc/fstab
/dev/VolGroup00/LogVol100 / ext3 defaults 1 1
LABEL=/boot /boot ext3 defaults 1 2
none /dev/pts devpts gid=5,mode=620 0 0
none /dev/shm tmpfs defaults 0 0
none /proc proc defaults 0 0
none /sys sysfs defaults 0 0
/dev/VolGroup00/LogVol101 swap swap defaults 0 0
```

By adding the following line, we can automate the mounting of a file system.

```
/dev/sdb1 /home/project55 ext2 defaults 0 0
```

mount /mountpoint

Adding an entry to **/etc/fstab** has the added advantage that you can simplify the **mount** command. The command in the screenshot below forces **mount** to look for the partition info in **/etc/fstab**.

```
# mount /home/project55
```

35.4. securing mounts

File systems can be secured with several **mount options**. Here are some examples.

ro

The **ro** option will mount a file system as read only, preventing anyone from writing.

```
root@rhel53 ~# mount -t ext2 -o ro /dev/hdb1 /home/project42
root@rhel53 ~# touch /home/project42/testwrite
touch: cannot touch `/home/project42/testwrite': Read-only file system
```

noexec

The **noexec** option will prevent the execution of binaries and scripts on the mounted file system.

```
root@rhel53 ~# mount -t ext2 -o noexec /dev/hdb1 /home/project42
root@rhel53 ~# cp /bin/cat /home/project42
root@rhel53 ~# /home/project42/cat /etc/hosts
-bash: /home/project42/cat: Permission denied
root@rhel53 ~# echo echo hello > /home/project42/helloscript
root@rhel53 ~# chmod +x /home/project42/helloscript
root@rhel53 ~# /home/project42/helloscript
-bash: /home/project42/helloscript: Permission denied
```

nosuid

The **nosuid** option will ignore **setuid** bit set binaries on the mounted file system.

Note that you can still set the **setuid** bit on files.

```
root@rhel53 ~# mount -o nosuid /dev/hdb1 /home/project42
root@rhel53 ~# cp /bin/sleep /home/project42/
root@rhel53 ~# chmod 4555 /home/project42/sleep
root@rhel53 ~# ls -l /home/project42/sleep
-r-sr-xr-x 1 root root 19564 Jun 24 17:57 /home/project42/sleep
```

But users cannot exploit the **setuid** feature.

```
root@rhel53 ~# su - paul
[paul@rhel53 ~]$ /home/project42/sleep 500 &
[1] 2876
[paul@rhel53 ~]$ ps -f 2876
UID          PID  PPID  C  STIME TTY          STAT      TIME CMD
paul         2876  2853  0  17:58 pts/0      S          0:00 /home/project42/sleep 500
[paul@rhel53 ~]$
```

noacl

To prevent cluttering permissions with **acl**'s, use the **noacl** option.

```
root@rhel53 ~# mount -o noacl /dev/hdb1 /home/project42
```

More **mount options** can be found in the manual page of **mount**.

35.5. practice: mounting file systems

1. Mount the small 200MB partition on `/home/project22`.
2. Mount the big 400MB primary partition on `/mnt`, the copy some files to it (everything in `/etc`). Then unmount, and mount the file system as read only on `/srv/nfs/salesnumbers`. Where are the files you copied ?
3. Verify your work with **fdisk**, **df** and **mount**. Also look in `/etc/mtab` and `/proc/mounts`.
4. Make both mounts permanent, test that it works.
5. What happens when you mount a file system on a directory that contains some files ?
6. What happens when you mount two file systems on the same mount point ?
7. (optional) Describe the difference between these file searching commands: `find`, `locate`, `updatedb`, `whereis`, `apropos` and `which`.
8. (optional) Perform a file system check on the partition mounted at `/srv/nfs/salesnumbers`.

35.6. solution: mounting file systems

1. Mount the small 200MB partition on /home/project22.

```
mkdir /home/project22
mount /dev/sdc1 /home/project22
```

2. Mount the big 400MB primary partition on /mnt, the copy some files to it (everything in /etc). Then unmount, and mount the file system as read only on /srv/nfs/salesnumbers. Where are the files you copied ?

```
mount /dev/sdb1 /mnt
cp -r /etc /mnt
ls -l /mnt
```

```
umount /mnt
ls -l /mnt
```

```
mkdir -p /srv/nfs/salesnumbers
mount /dev/sdb1 /srv/nfs/salesnumbers
```

You see the files in /srv/nfs/salenumbers now...

But physically they are on ext3 on partition /dev/sdb1

3. Verify your work with **fdisk**, **df** and **mount**. Also look in **/etc/mtab** and **/proc/mounts**.

```
fdisk -l
df -h
mount
```

All three the above commands should show your mounted partitions.

```
grep project22 /etc/mtab
grep project22 /proc/mounts
```

4. Make both mounts permanent, test that it works.

add the following lines to /etc/fstab

```
/dev/sdc1 /home/project22 auto defaults 0 0
/dev/sdb1 /srv/nfs/salesnumbers auto defaults 0 0
```

5. What happens when you mount a file system on a directory that contains some files ?

The files are hidden until **umount**.

6. What happens when you mount two file systems on the same mount point ?

Only the last mounted fs is visible.

7. (optional) Describe the difference between these file searching commands: find, locate, updatedb, whereis, apropos and which.

man is your friend

8. (optional) Perform a file system check on the partition mounted at /srv/nfs/salesnumbers.

```
better to unmount first before  
# fsck /dev/sdb1
```

Chapter 36. introduction to uuid's

Table of Contents

36.1. about unique objects	308
36.2. uuid in /etc/fstab	308
36.3. uuid in menu.lst	309
36.4. practice: uuid and filesystems	310
36.5. solution: uuid and filesystems	311

36.1. about unique objects

A **uuid** or **universally unique identifier** is used to uniquely identify objects. This 128bit standard allows anyone to create a unique **uuid**.

/sbin/vol_id

Below we use the **vol_id** utility to display the **uuid** of an **ext3** file system.

```
root@laika:~# vol_id --uuid /dev/sda1
825d4b79-ec40-4390-8a71-9261df8d4c82
```

/lib/udev/vol_id

Red Hat Enterprise Linux 5 puts **vol_id** in **/lib/udev/vol_id**, which is not in the **\$PATH**. The syntax is also a bit different from Debian/Ubuntu.

```
root@rhel53 ~# /lib/udev/vol_id -u /dev/hda1
48a6a316-9ca9-4214-b5c6-e7b33a77e860
```

/sbin/tune2fs

We can also use **tune2fs** to find the **uuid** of a file system.

```
[root@RHEL5 ~]# tune2fs -l /dev/sda1 | grep UUID
Filesystem UUID:          11cfc8bc-07c0-4c3f-9f64-78422ef1dd5c
[root@RHEL5 ~]# /lib/udev/vol_id -u /dev/sda1
11cfc8bc-07c0-4c3f-9f64-78422ef1dd5c
```

36.2. uuid in /etc/fstab

You can use the **uuid** to make sure that a volume is universally uniquely identified in **/etc/fstab**. The device name can change depending on the disk devices that are present at boot time, but a **uuid** never changes.

First we use **tune2fs** to find the **uuid**.

```
[root@RHEL5 ~]# tune2fs -l /dev/sdc1 | grep UUID
Filesystem UUID:          7626d73a-2bb6-4937-90ca-e451025d64e8
```

Then we check that it is properly added to **/etc/fstab**, the **uuid** replaces the variable **devicename** **/dev/sdc1**.

```
[root@RHEL5 ~]# grep UUID /etc/fstab
UUID=7626d73a-2bb6-4937-90ca-e451025d64e8 /home/pro42 ext3 defaults 0 0
```

Now we can mount the volume using the mount point defined in **/etc/fstab**.

```
[root@RHEL5 ~]# mount /home/pro42
[root@RHEL5 ~]# df -h | grep 42
/dev/sdc1          397M   11M   366M   3% /home/pro42
```

The real test now, is to remove **/dev/sdb** from the system, reboot the machine and see what happens. After the reboot, the disk previously known as **/dev/sdc** is now **/dev/sdb**.

```
[root@RHEL5 ~]# tune2fs -l /dev/sdb1 | grep UUID
Filesystem UUID:          7626d73a-2bb6-4937-90ca-e451025d64e8
```

And thanks to the **uuid** in **/etc/fstab**, the mountpoint is mounted on the same disk as before.

```
[root@RHEL5 ~]# df -h | grep sdb
/dev/sdb1          397M   11M   366M   3% /home/pro42
```

36.3. uuid in menu.lst

Recent incarnations of the Ubuntu distribution will use a **uuid** to identify the root file system. This example shows how a **root=/dev/sda1** is replaced with a **uuid**.

```
title Ubuntu 9.10, kernel 2.6.31-19-generic
uuid f001ba5d-9077-422a-9634-8d23d57e782a
kernel /boot/vmlinuz-2.6.31-19-generic \
root=UUID=f001ba5d-9077-422a-9634-8d23d57e782a ro quiet splash
initrd /boot/initrd.img-2.6.31-19-generic
```

The screenshot above contains only four lines. The line starting with **root=** is the continuation of the **kernel** line.

36.4. practice: uuid and filesystems

1. Find the **uuid** of one of your **ext3** partitions with **tune2fs** and **vol_id**.
2. Use this **uuid** in **/etc/fstab** and test that it works with a simple **mount**.
3. (optional) Test it also by removing a disk (so the device name is changed). You can edit settings in vmware/Virtualbox to remove a hard disk.
4. Display the **root=** directive in **/boot/grub/menu.lst**. (We see later in the course how to maintain this file.)
5. (optional) Replace the **/dev/xxx** in **/boot/grub/menu.lst** with a **uuid** (use an extra stanza for this). Test that it works.

36.5. solution: uuid and filesystems

1. Find the **uuid** of one of your **ext3** partitions with **tune2fs** and **vol_id**.

```
root@rhel155:~# /lib/udev/vol_id -u /dev/hda1
60926898-2c78-49b4-a71d-c1d6310c87cc
```

```
root@ubu1004:~# tune2fs -l /dev/sda2 | grep UUID
Filesystem UUID:          3007b743-1dce-2d62-9a59-cf25f85191b7
```

2. Use this **uuid** in **/etc/fstab** and test that it works with a simple **mount**.

```
tail -1 /etc/fstab
UUID=60926898-2c78-49b4-a71d-c1d6310c87cc /home/pro42 ext3 defaults 0 0
```

3. (optional) Test it also by removing a disk (so the device name is changed). You can edit settings in vmware/Virtualbox to remove a hard disk.

4. Display the **root=** directive in **/boot/grub/menu.lst**. (We see later in the course how to maintain this file.)

```
paul@deb503:~$ grep ^[^\#] /boot/grub/menu.lst | grep root=
kernel /boot/vmlinuz-2.6.26-2-686 root=/dev/hda1 ro selinux=1 quiet
kernel /boot/vmlinuz-2.6.26-2-686 root=/dev/hda1 ro selinux=1 single
```

5. (optional) Replace the **/dev/xxx** in **/boot/grub/menu.lst** with a **uuid** (use an extra stanza for this). Test that it works.

Chapter 37. introduction to raid

Table of Contents

37.1. hardware or software	312
37.2. raid levels	313
37.3. building a software raid5 array	315
37.4. practice: raid	318
37.5. solution: raid	319

37.1. hardware or software

Redundant Array of Independent (originally Inexpensive) Disks or **RAID** can be set up using hardware or software. Hardware RAID is more expensive, but offers better performance. Software RAID is cheaper and easier to manage, but it uses your CPU and your memory.

Where ten years ago nobody was arguing about the best choice being hardware RAID, this has changed since technologies like mdadm, lvm and even zfs focus more on managability. The workload on the cpu for software RAID used to be high, but cpu's have gotten a lot faster.

37.2. raid levels

raid 0

raid 0 uses two or more disks, and is often called **striping** (or stripe set, or striped volume). Data is divided in **chunks**, those chunks are evenly spread across every disk in the array. The main advantage of **raid 0** is that you can create **larger drives**. **raid 0** is the only **raid** without redundancy.

jbod

jbod uses two or more disks, and is often called **concatenating** (spanning, spanned set, or spanned volume). Data is written to the first disk, until it is full. Then data is written to the second disk... The main advantage of **jbod** (Just a Bunch of Disks) is that you can create **larger drives**. JBOD offers no redundancy.

raid 1

raid 1 uses exactly two disks, and is often called **mirroring** (or mirror set, or mirrored volume). All data written to the array is written on each disk. The main advantage of raid 1 is **redundancy**. The main disadvantage is that you lose at least half of your available disk space (in other words, you at least double the cost).

raid 2, 3 and 4 ?

raid 2 uses bit level striping, **raid 3** byte level, and **raid 4** is the same as **raid 5**, but with a dedicated parity disk. This is actually slower than **raid 5**, because every write would have to write parity to this one (bottleneck) disk. It is unlikely that you will ever see these **raid** levels in production.

raid 5

raid 5 uses **three** or more disks, each divided into chunks. Every time chunks are written to the array, one of the disks will receive a **parity** chunk. Unlike **raid 4**, the parity chunk will alternate between all disks. The main advantage of this is that **raid 5** will allow for full data recovery in case of **one** hard disk failure.

raid 6

raid 6 is very similar to **raid 5**, but uses two parity chunks. **raid 6** protects against two hard disk failures. Oracle Solaris **zfs** calls this **raidz2** (and also had **raidz3** with triple parity).

raid 0+1

raid 0+1 is a mirror(1) of stripes(0). This means you first create two **raid 0 stripe** sets, and then you set them up as a mirror set. For example, when you have six 100GB disks, then the stripe sets are each 300GB. Combined in a mirror, this makes 300GB total. **raid 0+1** will survive one disk failure. It will only survive the second disk failure if this disk is in the same stripe set as the previous failed disk.

raid 1+0

raid 1+0 is a stripe(0) of mirrors(1). For example, when you have six 100GB disks, then you first create three mirrors of 100GB each. You then stripe them together into a 300GB drive. In this example, as long as not all disks in the same mirror fail, it can survive up to three hard disk failures.

raid 50

raid 5+0 is a stripe(0) of **raid 5** arrays. Suppose you have nine disks of 100GB, then you can create three **raid 5** arrays of 200GB each. You can then combine them into one large stripe set.

many others

There are many other nested **raid** combinations, like **raid 30, 51, 60, 100, 150, ...**

37.3. building a software raid5 array

do we have three disks?

First, you have to attach some disks to your computer. In this scenario, three brand new disks of eight gigabyte each are added. Check with **fdisk -l** that they are connected.

```
[root@rhel6c ~]# fdisk -l 2> /dev/null | grep MB
Disk /dev/sdb: 8589 MB, 8589934592 bytes
Disk /dev/sdc: 8589 MB, 8589934592 bytes
Disk /dev/sdd: 8589 MB, 8589934592 bytes
```

fd partition type

The next step is to create a partition of type **fd** on every disk. The **fd** type is to set the partition as **Linux RAID autodetect**. See this (truncated) screenshot:

```
[root@rhel6c ~]# fdisk /dev/sdd
...
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1044, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-1044, default 1044):
Using default value 1044

Command (m for help): t
Selected partition 1
Hex code (type L to list codes): fd
Changed system type of partition 1 to fd (Linux raid autodetect)

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

verify all three partitions

Now all three disks are ready for **raid 5**, so we have to tell the system what to do with these disks.

```
[root@rhel6c ~]# fdisk -l 2> /dev/null | grep raid
/dev/sdb1      1      1044      8385898+   fd   Linux raid autodetect
/dev/sdc1      1      1044      8385898+   fd   Linux raid autodetect
/dev/sdd1      1      1044      8385898+   fd   Linux raid autodetect
```

create the raid5

The next step used to be *create the raid table in /etc/raidtab*. Nowadays, you can just issue the command **mdadm** with the correct parameters.

The command below is split on two lines to fit this print, but you should type it on one line, without the backslash (\).

```
[root@rhel6c ~]# mdadm --create /dev/md0 --chunk=64 --level=5 --raid-\
devices=3 /dev/sdb1 /dev/sdc1 /dev/sdd1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

Below a partial screenshot how **fdisk -l** sees the **raid 5**.

```
[root@rhel6c ~]# fdisk -l /dev/md0

Disk /dev/md0: 17.2 GB, 17172135936 bytes
2 heads, 4 sectors/track, 4192416 cylinders
Units = cylinders of 8 * 512 = 4096 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 131072 bytes
Disk identifier: 0x00000000
```

```
Disk /dev/md0 doesn't contain a valid partition table
```

We could use this software **raid 5** array in the next topic: **lvm**.

/proc/mdstat

The status of the raid devices can be seen in **/proc/mdstat**. This example shows a **raid 5** in the process of rebuilding.

```
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd1[3] sdc1[1] sdb1[0]
      16769664 blocks super 1.2 level 5, 64k chunk, algorithm 2 [3/2] [UU_]
      [=====>.....] recovery = 62.8% (5266176/8384832) finish=0\
      .3min speed=139200K/sec
```

This example shows an active software **raid 5**.

```
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 sdd1[3] sdc1[1] sdb1[0]
      16769664 blocks super 1.2 level 5, 64k chunk, algorithm 2 [3/3] [UUU]
```

mdadm --detail

Use **mdadm --detail** to get information on a raid device.

```
[root@rhel6c ~]# mdadm --detail /dev/md0
/dev/md0:
  Version : 1.2
  Creation Time : Sun Jul 17 13:48:41 2011
```

```
Raid Level : raid5
Array Size : 16769664 (15.99 GiB 17.17 GB)
Used Dev Size : 8384832 (8.00 GiB 8.59 GB)
Raid Devices : 3
Total Devices : 3
Persistence : Superblock is persistent

Update Time : Sun Jul 17 13:49:43 2011
State : clean
Active Devices : 3
Working Devices : 3
Failed Devices : 0
Spare Devices : 0

Layout : left-symmetric
Chunk Size : 64K

Name : rhel6c:0 (local to host rhel6c)
UUID : c10fd9c3:08f9a25f:be913027:999c8e1f
Events : 18

Number   Major   Minor   RaidDevice State
  0         8       17         0   active sync  /dev/sdb1
  1         8       33         1   active sync  /dev/sdc1
  3         8       49         2   active sync  /dev/sdd1
```

removing a software raid

The software raid is visible in `/proc/mdstat` when active. To remove the raid completely so you can use the disks for other purposes, you stop (de-activate) it with **mdadm**.

```
[root@rhel6c ~]# mdadm --stop /dev/md0
mdadm: stopped /dev/md0
```

The disks can now be repartitioned.

37.4. practice: raid

1. Add three virtual disks of 1GB each to a virtual machine.
2. Create a software **raid 5** on the three disks. (It is not necessary to put a filesystem on it)
3. Verify with **fdisk** and in **/proc** that the **raid 5** exists.
4. (optional) Stop and remove the **raid 5**.
5. (optional) Create a **raid 1** to mirror two disks.

37.5. solution: raid

1. Add three virtual disks of 1GB each to a virtual machine.
2. Create a software **raid 5** on the three disks. (It is not necessary to put a filesystem on it)
3. Verify with **fdisk** and in **/proc** that the **raid 5** exists.
4. (optional) Stop and remove the **raid 5**.
5. (optional) Create a **raid 1** to mirror two disks.

```
[root@rhel6c ~]# mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb1 /dev/sdc1
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
[root@rhel6c ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4] [raid1]
md0 : active raid1 sdc1[1] sdb1[0]
      8384862 blocks super 1.2 [2/2] [UU]
      [====>.....] resync = 20.8% (1745152/8384862) \
finish=0.5min speed=218144K/sec
```

Chapter 38. logical volume management

Table of Contents

38.1. introduction to lvm	321
38.2. lvm terminology	322
38.3. example: using lvm	323
38.4. example: extend a logical volume	325
38.5. example: resize a physical Volume	327
38.6. example: mirror a logical volume	329
38.7. example: snapshot a logical volume	330
38.8. verifying existing physical volumes	331
38.9. verifying existing volume groups	333
38.10. verifying existing logical volumes	335
38.11. manage physical volumes	336
38.12. manage volume groups	338
38.13. manage logical volumes	340
38.14. practice : lvm	343

38.1. introduction to lvm

problems with standard partitions

There are some problems when working with hard disks and standard partitions. Consider a system with a small and a large hard disk device, partitioned like this. The first disk (/dev/sda) is partitioned in two, the second disk (/dev/sdb) has three partitions.

Table 38.1. disk partitioning example

/dev/sda		/dev/sdb			
/dev/sda1	/dev/sda2	/dev/sdb1	/dev/sdb2	/dev/sdb3	unused
/boot	/	/var	/home	/project42	
ext2	ext3	ext2	reiserfs	ext3	

In the example above, consider the options when you want to enlarge the space available for **/project42**. What can you do ? The solution will always force you to unmount the filesystem, take a backup of the data, remove and recreate partitions, and then restore the data and remount the file system.

solution with lvm

Using **lvm** will create a virtual layer between the mounted file systems and the hardware devices. This virtual layer will allow for an administrator to enlarge a mounted file system in use. When **lvm** is properly used, then there is no need to unmount the file system to enlarge it.

Table 38.2. LVM Example

/dev/sda		/dev/sdb			
Volume Group					
/boot	/	/var	/home	/project42	
ext2	ext3	ext2	reiserfs	ext3	

about lvm

Most **lvm** implementations support **physical storage grouping**, **logical volume resizing** and **data migration**.

Physical storage grouping is a fancy name for grouping multiple physical devices (hard disks) into a logical mass storage device. To enlarge this physical group, hard disks or even single partitions can be added at a later time. The size of **lvm volumes** on this **physical group** is independent of the individual size of the components. The total size of the group is the limit.

One of the nicest features of **lvm** is the logical volume resizing. You can increase the size of an **lvm volume**, sometimes even without any downtime. Additionally, you can migrate data away from a failing hard disk device.

38.2. lvm terminology

physical volume (pv)

A **physical volume** is a disk, a partition or a (hardware or software) RAID device. All these devices can become a member of a **Volume Group**.

volume group (vg)

A **Volume Group** is an abstraction layer between **Physical Devices** and **Logical Volumes**.

logical volume (lv)

A **Logical Volume** is created in a **Volume Group**. Logical Volumes that contain a file system can be mounted. The use of logical volumes is similar to the use of partitions (both are standard block devices) and is accomplished with the same standard commands (mkfs, mount, fsck, df, ...).

38.3. example: using lvm

This example shows how you can use a device (in this case `/dev/sdc`, but it could have been `/dev/sdb` or any other disk or partition) with `lvm`, how to create a volume group (`vg`) and how to create and use a logical volume (`vg/lvol0`).

First thing to do, is create physical volumes that can join the volume group with **`pvcreate`**. This command makes a disk or partition available for use in Volume Groups. The screenshot shows how to present the SCSI Disk device to LVM.

```
root@RHEL4:~# pvcreate /dev/sdc
Physical volume "/dev/sdc" successfully created
```

Note for home users: lvm will work fine when using the complete disk, but another operating system on the same computer will not recognize lvm and will mark the disk as being empty! You can avoid this by creating a partition that spans the whole disk, then run `pvcreate` on the partition instead of the disk.

Then **`vgcreate`** creates a volume group using one device. Note that more devices could be added to the volume group.

```
root@RHEL4:~# vgcreate vg /dev/sdc
Volume group "vg" successfully created
```

The last step **`lvcreate`** creates a logical volume.

```
root@RHEL4:~# lvcreate --size 500m vg
Logical volume "lvol0" created
```

The logical volume `/dev/vg/lvol0` can now be formatted with `ext2`, and mounted for normal use.

```
root@RHELv4u2:~# mke2fs -m0 -j /dev/vg/lvol0
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
128016 inodes, 512000 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67633152
63 block groups
8192 blocks per group, 8192 fragments per group
2032 inodes per group
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
root@RHELv4u2:~# mkdir /home/project10
root@RHELv4u2:~# mount /dev/vg/lvol0 /home/project10/
root@RHELv4u2:~# df -h | grep proj
/dev/mapper/vg-lvol0 485M 11M 474M 3% /home/project10
```

A logical volume is very similar to a partition, it can be formatted with a file system, and can be mounted so users can access it.

38.4. example: extend a logical volume

A logical volume can be extended without unmounting the file system. Whether or not a volume can be extended depends on the file system it uses. Volumes that are mounted as vfat or ext2 cannot be extended, so in the example here we use the ext3 file system.

The `fdisk` command shows us newly added scsi-disks that will serve our lvm volume. This volume will then be extended. First, take a look at these disks.

```
[root@RHEL5 ~]# fdisk -l | grep sd[bc]
Disk /dev/sdb doesn't contain a valid partition table
Disk /dev/sdc doesn't contain a valid partition table
Disk /dev/sdb: 1181 MB, 1181115904 bytes
Disk /dev/sdc: 429 MB, 429496320 bytes
```

You already know how to partition a disk, below the first disk is partitioned (in one big primary partition), the second disk is left untouched.

```
[root@RHEL5 ~]# fdisk -l | grep sd[bc]
Disk /dev/sdc doesn't contain a valid partition table
Disk /dev/sdb: 1181 MB, 1181115904 bytes
/dev/sdb1          1          143      1148616    83   Linux
Disk /dev/sdc: 429 MB, 429496320 bytes
```

You also know how to prepare disks for lvm with `pvcreate`, and how to create a volume group with `vgcreate`. This example adds both the partitioned disk and the untouched disk to the volume group named `vg2`.

```
[root@RHEL5 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
[root@RHEL5 ~]# pvcreate /dev/sdc
  Physical volume "/dev/sdc" successfully created
[root@RHEL5 ~]# vgcreate vg2 /dev/sdb1 /dev/sdc
  Volume group "vg2" successfully created
```

You can use `pvdisplay` to verify that both the disk and the partition belong to the volume group.

```
[root@RHEL5 ~]# pvdisplay | grep -B1 vg2
PV Name                /dev/sdb1
VG Name                vg2
--
PV Name                /dev/sdc
VG Name                vg2
```

And you are familiar both with the `lvcreate` command to create a small logical volume and the `mke2fs` command to put ext2 on it.

```
[root@RHEL5 ~]# lvcreate --size 200m vg2
Logical volume "lvol0" created
[root@RHEL5 ~]# mke2fs -m20 -j /dev/vg2/lvol0
...
```

As you see, we end up with a mounted logical volume that according to **df** is almost 200 megabyte in size.

```
[root@RHEL5 ~]# mkdir /home/resizetest
[root@RHEL5 ~]# mount /dev/vg2/lvol0 /home/resizetest/
[root@RHEL5 ~]# df -h | grep resizetest
194M 5.6M 149M 4% /home/resizetest
```

Extending the volume is easy with **lvextend**.

```
[root@RHEL5 ~]# lvextend -L +100 /dev/vg2/lvol0
Extending logical volume lvol0 to 300.00 MB
Logical volume lvol0 successfully resized
```

But as you can see, there is a small problem: it appears that **df** is not able to display the extended volume in its full size. This is because the filesystem is only set for the size of the volume before the extension was added.

```
[root@RHEL5 ~]# df -h | grep resizetest
194M 5.6M 149M 4% /home/resizetest
```

With **lvdisplay** however we can see that the volume is indeed extended.

```
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
LV Size                300.00 MB
```

To finish the extension, you need **resize2fs** to span the filesystem over the full size of the logical volume.

```
[root@RHEL5 ~]# resize2fs /dev/vg2/lvol0
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/vg2/lvol0 is mounted on /home/resizetest; on-line re\
sizing required
Performing an on-line resize of /dev/vg2/lvol0 to 307200 (1k) blocks.
The filesystem on /dev/vg2/lvol0 is now 307200 blocks long.
```

Congratulations, you just successfully expanded a logical volume.

```
[root@RHEL5 ~]# df -h | grep resizetest
291M 6.1M 225M 3% /home/resizetest
[root@RHEL5 ~]#
```

38.5. example: resize a physical Volume

This is a humble demonstration of how to resize a physical Volume with lvm (after you resize it with fdisk). The demonstration starts with a 100MB partition named /dev/sde1. We used fdisk to create it, and to verify the size.

```
[root@RHEL5 ~]# fdisk -l 2>/dev/null | grep sde1
/dev/sde1          1          100          102384    83  Linux
[root@RHEL5 ~]#
```

Now we can use pvcreate to create the Physical Volume, followed by pvs to verify the creation.

```
[root@RHEL5 ~]# pvcreate /dev/sde1
Physical volume "/dev/sde1" successfully created
[root@RHEL5 ~]# pvs | grep sde1
/dev/sde1          lvm2 --          99.98M  99.98M
[root@RHEL5 ~]#
```

The next step is to use fdisk to enlarge the partition (actually deleting it and then recreating /dev/sde1 with more cylinders).

```
[root@RHEL5 ~]# fdisk /dev/sde

Command (m for help): p

Disk /dev/sde: 858 MB, 858993152 bytes
64 heads, 32 sectors/track, 819 cylinders
Units = cylinders of 2048 * 512 = 1048576 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sde1          1           100       102384    83  Linux

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4):
Value out of range.
Partition number (1-4): 1
First cylinder (1-819, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-819, default 819): 200

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
[root@RHEL5 ~]#
```

When we now use `fdisk` and `pvs` to verify the size of the partition and the Physical Volume, then there is a size difference. LVM is still using the old size.

```
[root@RHEL5 ~]# fdisk -l 2>/dev/null | grep sde1
/dev/sde1          1          200          204784    83  Linux
[root@RHEL5 ~]# pvs | grep sde1
/dev/sde1          lvm2 --      99.98M  99.98M
[root@RHEL5 ~]#
```

Executing `pvresize` on the Physical Volume will make `lvm` aware of the size change of the partition. The correct size can be displayed with `pvs`.

```
[root@RHEL5 ~]# pvresize /dev/sde1
Physical volume "/dev/sde1" changed
 1 physical volume(s) resized / 0 physical volume(s) not resized
[root@RHEL5 ~]# pvs | grep sde1
/dev/sde1          lvm2 --      199.98M 199.98M
[root@RHEL5 ~]#
```

38.6. example: mirror a logical volume

We start by creating three physical volumes for lvm. Then we verify the creation and the size with pvs. Three physical disks because lvm uses two disks for the mirror and a third disk for the mirror log!

```
[root@RHEL5 ~]# pvcreate /dev/sdb /dev/sdc /dev/sdd
Physical volume "/dev/sdb" successfully created
Physical volume "/dev/sdc" successfully created
Physical volume "/dev/sdd" successfully created
[root@RHEL5 ~]# pvs
PV          VG          Fmt  Attr  PSize   PFree
/dev/sdb    lvm2  --    409.60M 409.60M
/dev/sdc    lvm2  --    409.60M 409.60M
/dev/sdd    lvm2  --    409.60M 409.60M
```

Then we create the Volume Group and verify again with pvs. Notice how the three physical volumes now belong to vg33, and how the size is rounded down (in steps of the extent size, here 4MB).

```
[root@RHEL5 ~]# vgcreate vg33 /dev/sdb /dev/sdc /dev/sdd
Volume group "vg33" successfully created
[root@RHEL5 ~]# pvs
PV          VG          Fmt  Attr  PSize   PFree
/dev/sda2   VolGroup00 lvm2  a-    15.88G    0
/dev/sdb    vg33        lvm2  a-    408.00M 408.00M
/dev/sdc    vg33        lvm2  a-    408.00M 408.00M
/dev/sdd    vg33        lvm2  a-    408.00M 408.00M
[root@RHEL5 ~]#
```

The last step is to create the Logical Volume with **lvcreate**. Notice the **-m 1** switch to create one mirror. Notice also the change in free space in all three Physical Volumes!

```
[root@RHEL5 ~]# lvcreate --size 300m -n lvmir -m 1 vg33
Logical volume "lvmir" created
[root@RHEL5 ~]# pvs
PV          VG          Fmt  Attr  PSize   PFree
/dev/sda2   VolGroup00 lvm2  a-    15.88G    0
/dev/sdb    vg33        lvm2  a-    408.00M 108.00M
/dev/sdc    vg33        lvm2  a-    408.00M 108.00M
/dev/sdd    vg33        lvm2  a-    408.00M 404.00M
```

You can see the copy status of the mirror with lvs. It currently shows a 100 percent copy.

```
[root@RHEL5 ~]# lvs vg33/lvmir
LV  VG  Attr  LSize  Origin Snap%  Move Log           Copy%
lvmir  vg33 mwi-ao 300.00M                               lvmir_mlog 100.00
```

38.7. example: snapshot a logical volume

A snapshot is a virtual copy of all the data at a point in time on a volume. A snapshot Logical Volume will retain a copy of all changed files of the snapshotted Logical Volume.

The example below creates a snapshot of the bigLV Logical Volume.

```
[root@RHEL5 ~]# lvcreate -L100M -s -n snapLV vg42/bigLV
Logical volume "snapLV" created
[root@RHEL5 ~]#
```

You can see with `lvs` that the snapshot `snapLV` is indeed a snapshot of `bigLV`. Moments after taking the snapshot, there are few changes to `bigLV` (0.02 percent).

```
[root@RHEL5 ~]# lvs
LV      VG      Attr   LSize   Origin Snap%  Move Log Copy%
bigLV   vg42    owi-a- 200.00M
snapLV  vg42    swi-a- 100.00M bigLV   0.02
[root@RHEL5 ~]#
```

But after using `bigLV` for a while, more changes are done. This means the snapshot volume has to keep more original data (10.22 percent).

```
[root@RHEL5 ~]# lvs | grep vg42
bigLV   vg42    owi-ao 200.00M
snapLV  vg42    swi-a- 100.00M bigLV   10.22
[root@RHEL5 ~]#
```

You can now use regular backup tools (`dump`, `tar`, `cpio`, ...) to take a backup of the snapshot Logical Volume. This backup will contain all data as it existed on `bigLV` at the time the snapshot was taken. When the backup is done, you can remove the snapshot.

```
[root@RHEL5 ~]# lvremove vg42/snapLV
Do you really want to remove active logical volume "snapLV"? [y/n]: y
Logical volume "snapLV" successfully removed
[root@RHEL5 ~]#
```


38.8. verifying existing physical volumes

lvmdiskscan

To get a list of block devices that can be used with LVM, use **lvmdiskscan**. The example below uses `grep` to limit the result to SCSI devices.

```
[root@RHEL5 ~]# lvmdiskscan | grep sd
/dev/sda1          [      101.94 MB]
/dev/sda2          [      15.90 GB] LVM physical volume
/dev/sdb           [      409.60 MB]
/dev/sdc           [      409.60 MB]
/dev/sdd           [      409.60 MB] LVM physical volume
/dev/sde1          [       95.98 MB]
/dev/sde5          [      191.98 MB]
/dev/sdf           [      819.20 MB] LVM physical volume
/dev/sdg1          [      818.98 MB]
[root@RHEL5 ~]#
```

pvs

The easiest way to verify whether devices are known to lvm is with the **pvs** command. The screenshot below shows that only `/dev/sda2` is currently known for use with LVM. It shows that `/dev/sda2` is part of `Volgroup00` and is almost 16GB in size. It also shows `/dev/sdc` and `/dev/sdd` as part of `vg33`. The device `/dev/sdb` is known to lvm, but not linked to any Volume Group.

```
[root@RHEL5 ~]# pvs
PV          VG          Fmt Attr PSize  PFree
/dev/sda2   VolGroup00 lvm2 a-   15.88G    0
/dev/sdb    lvm2  --    409.60M 409.60M
/dev/sdc    vg33    lvm2 a-   408.00M 408.00M
/dev/sdd    vg33    lvm2 a-   408.00M 408.00M
[root@RHEL5 ~]#
```

pvscan

The **pvscan** command will scan all disks for existing Physical Volumes. The information is similar to `pvs`, plus you get a line with total sizes.

```
[root@RHEL5 ~]# pvscan
PV /dev/sdc    VG vg33          lvm2 [408.00 MB / 408.00 MB free]
PV /dev/sdd    VG vg33          lvm2 [408.00 MB / 408.00 MB free]
PV /dev/sda2   VG VolGroup00    lvm2 [15.88 GB / 0    free]
PV /dev/sdb    lvm2             lvm2 [409.60 MB]
Total: 4 [17.07 GB] / in use: 3 [16.67 GB] / in no VG: 1 [409.60 MB]
[root@RHEL5 ~]#
```

pvdisplay

Use **pvdisplay** to get more information about physical volumes. You can also use **pvdisplay** without an argument to display information about all physical (lvm) volumes.

```
[root@RHEL5 ~]# pvdisplay /dev/sda2
--- Physical volume ---
PV Name                /dev/sda2
VG Name                VolGroup00
PV Size                15.90 GB / not usable 20.79 MB
Allocatable           yes (but full)
PE Size (KByte)       32768
Total PE              508
Free PE                0
Allocated PE          508
PV UUID                TobYfp-Ggg0-Rf8r-xtLd-5XgN-RSPc-8vkTHD

[root@RHEL5 ~]#
```

38.9. verifying existing volume groups

vgs

Similar to **pvs** is the use of **vgs** to display a quick overview of all volume groups. There is only one volume group in the screenshot below, it is named VolGroup00 and is almost 16GB in size.

```
[root@RHEL5 ~]# vgs
  VG          #PV #LV #SN Attr   VSize  VFree
  VolGroup00   1   2   0 wz--n- 15.88G   0
[root@RHEL5 ~]#
```

vgscan

The **vgscan** command will scan all disks for existing Volume Groups. It will also update the **/etc/lvm/.cache** file. This file contains a list of all current lvm devices.

```
[root@RHEL5 ~]# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "VolGroup00" using metadata type lvm2
[root@RHEL5 ~]#
```

LVM will run the **vgscan** automatically at boot-up, so if you add hot swap devices, then you will need to run **vgscan** to update **/etc/lvm/.cache** with the new devices.

vgdisplay

The **vgdisplay** command will give you more detailed information about a volume group (or about all volume groups if you omit the argument).

```
[root@RHEL5 ~]# vgdisplay VolGroup00
  --- Volume group ---
  VG Name                VolGroup00
  System ID
  Format                  lvm2
  Metadata Areas         1
  Metadata Sequence No   3
  VG Access               read/write
  VG Status               resizable
  MAX LV                  0
  Cur LV                  2
  Open LV                  2
  Max PV                  0
  Cur PV                  1
  Act PV                  1
  VG Size                 15.88 GB
  PE Size                 32.00 MB
  Total PE                508
```

logical volume management

```
Alloc PE / Size      508 / 15.88 GB
Free  PE / Size      0 / 0
VG UUID              qsXvJb-71qV-917U-ishX-FobM-qpTE-VXmKIg
```

```
[root@RHEL5 ~]#
```

38.10. verifying existing logical volumes

lvs

Use **lvs** for a quick look at all existing logical volumes. Below you can see two logical volumes named LogVol00 and LogVol01.

```
[root@RHEL5 ~]# lvs
  LV      VG          Attr      LSize  Origin Snap%  Move Log Copy%
  LogVol00 VolGroup00 -wi-ao 14.88G
  LogVol01 VolGroup00 -wi-ao  1.00G
[root@RHEL5 ~]#
```

lvscan

The **lvscan** command will scan all disks for existing Logical Volumes.

```
[root@RHEL5 ~]# lvscan
ACTIVE          '/dev/VolGroup00/LogVol100' [14.88 GB] inherit
ACTIVE          '/dev/VolGroup00/LogVol101' [1.00 GB] inherit
[root@RHEL5 ~]#
```

lvdisplay

More detailed information about logical volumes is available through the **lvdisplay(1)** command.

```
[root@RHEL5 ~]# lvdisplay VolGroup00/LogVol01
--- Logical volume ---
LV Name                /dev/VolGroup00/LogVol01
VG Name                VolGroup00
LV UUID                RnTGK6-xWsi-t530-ksJx-7cax-co5c-A1K1Dp
LV Write Access        read/write
LV Status              available
# open                 1
LV Size                1.00 GB
Current LE             32
Segments               1
Allocation              inherit
Read ahead sectors     0
Block device           253:1

[root@RHEL5 ~]#
```

38.11. manage physical volumes

pvcreate

Use the **pvcreate** command to add devices to lvm. This example shows how to add a disk (or hardware RAID device) to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sdb
Physical volume "/dev/sdb" successfully created
[root@RHEL5 ~]#
```

This example shows how to add a partition to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sdc1
Physical volume "/dev/sdc1" successfully created
[root@RHEL5 ~]#
```

You can also add multiple disks or partitions as target to pvcreate. This example adds three disks to lvm.

```
[root@RHEL5 ~]# pvcreate /dev/sde /dev/sdf /dev/sdg
Physical volume "/dev/sde" successfully created
Physical volume "/dev/sdf" successfully created
Physical volume "/dev/sdg" successfully created
[root@RHEL5 ~]#
```

pvremove

Use the **pvremove** command to remove physical volumes from lvm. The devices may not be in use.

```
[root@RHEL5 ~]# pvremove /dev/sde /dev/sdf /dev/sdg
Labels on physical volume "/dev/sde" successfully wiped
Labels on physical volume "/dev/sdf" successfully wiped
Labels on physical volume "/dev/sdg" successfully wiped
[root@RHEL5 ~]#
```

pvresize

When you used fdisk to resize a partition on a disk, then you must use **pvresize** to make lvm recognize the new size of the physical volume that represents this partition.

```
[root@RHEL5 ~]# pvresize /dev/sde1
Physical volume "/dev/sde1" changed
1 physical volume(s) resized / 0 physical volume(s) not resized
```

pvchange

With **pvchange** you can prevent the allocation of a Physical Volume in a new Volume Group or Logical Volume. This can be useful if you plan to remove a Physical Volume.

```
[root@RHEL5 ~]# pvchange -xn /dev/sdd
Physical volume "/dev/sdd" changed
1 physical volume changed / 0 physical volumes not changed
[root@RHEL5 ~]#
```

To revert your previous decision, this example shows you how to re-enable the Physical Volume to allow allocation.

```
[root@RHEL5 ~]# pvchange -xy /dev/sdd
Physical volume "/dev/sdd" changed
1 physical volume changed / 0 physical volumes not changed
[root@RHEL5 ~]#
```

pvmove

With **pvmove** you can move Logical Volumes from within a Volume Group to another Physical Volume. This must be done before removing a Physical Volume.

```
[root@RHEL5 ~]# pvs | grep vg1
/dev/sdf vg1 lvm2 a- 816.00M 0
/dev/sdg vg1 lvm2 a- 816.00M 816.00M
[root@RHEL5 ~]# pvmove /dev/sdf
/dev/sdf: Moved: 70.1%
/dev/sdf: Moved: 100.0%
[root@RHEL5 ~]# pvs | grep vg1
/dev/sdf vg1 lvm2 a- 816.00M 816.00M
/dev/sdg vg1 lvm2 a- 816.00M 0
```

38.12. manage volume groups

vgcreate

Use the **vgcreate** command to create a volume group. You can immediately name all the physical volumes that span the volume group.

```
[root@RHEL5 ~]# vgcreate vg42 /dev/sde /dev/sdf
Volume group "vg42" successfully created
[root@RHEL5 ~]#
```

vgextend

Use the **vgextend** command to extend an existing volume group with a physical volume.

```
[root@RHEL5 ~]# vgextend vg42 /dev/sdg
Volume group "vg42" successfully extended
[root@RHEL5 ~]#
```

vgremove

Use the **vgremove** command to remove volume groups from lvm. The volume groups may not be in use.

```
[root@RHEL5 ~]# vgremove vg42
Volume group "vg42" successfully removed
[root@RHEL5 ~]#
```

vgreduce

Use the **vgreduce** command to remove a Physical Volume from the Volume Group.

The following example adds Physical Volume /dev/sdg to the vg1 Volume Group using vgextend. And then removes it again using vgreduce.

```
[root@RHEL5 ~]# pvs | grep sdg
/dev/sdg          lvm2 --  819.20M 819.20M
[root@RHEL5 ~]# vgextend vg1 /dev/sdg
Volume group "vg1" successfully extended
[root@RHEL5 ~]# pvs | grep sdg
/dev/sdg          vg1      lvm2 a-  816.00M 816.00M
[root@RHEL5 ~]# vgreduce vg1 /dev/sdg
Removed "/dev/sdg" from volume group "vg1"
[root@RHEL5 ~]# pvs | grep sdg
```



```
/dev/sdg          lvm2 --      819.20M 819.20M
```

vgchange

Use the **vgchange** command to change parameters of a Volume Group.

This example shows how to prevent Physical Volumes from being added or removed to the Volume Group `vg1`.

```
[root@RHEL5 ~]# vgchange -xn vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgextend vg1 /dev/sdg
  Volume group vg1 is not resizable.
```

You can also use `vgchange` to change most other properties of a Volume Group. This example changes the maximum number of Logical Volumes and maximum number of Physical Volumes that `vg1` can serve.

```
[root@RHEL5 ~]# vgdisplay vg1 | grep -i max
  MAX LV          0
  Max PV          0
[root@RHEL5 ~]# vgchange -l16 vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgchange -p8 vg1
  Volume group "vg1" successfully changed
[root@RHEL5 ~]# vgdisplay vg1 | grep -i max
  MAX LV          16
  Max PV          8
```

vgmerge

Merging two Volume Groups into one is done with **vgmerge**. The following example merges `vg2` into `vg1`, keeping all the properties of `vg1`.

```
[root@RHEL5 ~]# vgmerge vg1 vg2
  Volume group "vg2" successfully merged into "vg1"
[root@RHEL5 ~]#
```

38.13. manage logical volumes

lvcreate

Use the **lvcreate** command to create Logical Volumes in a Volume Group. This example creates an 8GB Logical Volume in Volume Group `vg42`.

```
[root@RHEL5 ~]# lvcreate -L5G vg42
Logical volume "lvol0" created
[root@RHEL5 ~]#
```

As you can see, `lvm` automatically names the Logical Volume **lvol0**. The next example creates a 200MB Logical Volume named `MyLV` in Volume Group `vg42`.

```
[root@RHEL5 ~]# lvcreate -L200M -nMyLV vg42
Logical volume "MyLV" created
[root@RHEL5 ~]#
```

The next example does the same thing, but with different syntax.

```
[root@RHEL5 ~]# lvcreate --size 200M -n MyLV vg42
Logical volume "MyLV" created
[root@RHEL5 ~]#
```

This example creates a Logical Volume that occupies 10 percent of the Volume Group.

```
[root@RHEL5 ~]# lvcreate -l 10%VG -n MyLV2 vg42
Logical volume "MyLV2" created
[root@RHEL5 ~]#
```

This example creates a Logical Volume that occupies 30 percent of the remaining free space in the Volume Group.

```
[root@RHEL5 ~]# lvcreate -l 30%FREE -n MyLV3 vg42
Logical volume "MyLV3" created
[root@RHEL5 ~]#
```

lvremove

Use the **lvremove** command to remove Logical Volumes from a Volume Group. Removing a Logical Volume requires the name of the Volume Group.

```
[root@RHEL5 ~]# lvremove vg42/MyLV
```

```
Do you really want to remove active logical volume "MyLV"? [y/n]: y
Logical volume "MyLV" successfully removed
[root@RHEL5 ~]#
```

Removing multiple Logical Volumes will request confirmation for each individual volume.

```
[root@RHEL5 ~]# lvremove vg42/MyLV vg42/MyLV2 vg42/MyLV3
Do you really want to remove active logical volume "MyLV"? [y/n]: y
Logical volume "MyLV" successfully removed
Do you really want to remove active logical volume "MyLV2"? [y/n]: y
Logical volume "MyLV2" successfully removed
Do you really want to remove active logical volume "MyLV3"? [y/n]: y
Logical volume "MyLV3" successfully removed
[root@RHEL5 ~]#
```

lvextend

Extending the volume is easy with **lvextend**. This example extends a 200MB Logical Volume with 100 MB.

```
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
LV Size                200.00 MB
[root@RHEL5 ~]# lvextend -L +100 /dev/vg2/lvol0
Extending logical volume lvol0 to 300.00 MB
Logical volume lvol0 successfully resized
[root@RHEL5 ~]# lvdisplay /dev/vg2/lvol0 | grep Size
LV Size                300.00 MB
```

The next example creates a 100MB Logical Volume, and then extends it to 500MB.

```
[root@RHEL5 ~]# lvcreate --size 100M -n extLV vg42
Logical volume "extLV" created
[root@RHEL5 ~]# lvextend -L 500M vg42/extLV
Extending logical volume extLV to 500.00 MB
Logical volume extLV successfully resized
[root@RHEL5 ~]#
```

This example doubles the size of a Logical Volume.

```
[root@RHEL5 ~]# lvextend -l+100%LV vg42/extLV
Extending logical volume extLV to 1000.00 MB
Logical volume extLV successfully resized
[root@RHEL5 ~]#
```

lvrename

Renaming a Logical Volume is done with **lvrename**. This example renames extLV to bigLV in the vg42 Volume Group.

```
[root@RHEL5 ~]# lvrename vg42/extLV vg42/bigLV
Renamed "extLV" to "bigLV" in volume group "vg42"
[root@RHEL5 ~]#
```

38.14. practice : lvm

1. Create a volume group that contains a complete disk and a partition on another disk.
2. Create two logical volumes (a small one and a bigger one) in this volume group. Format them with ext3, mount them and copy some files to them.
3. Verify usage with fdisk, mount, pvs, vgs, lvs, pvdisplay, vgdisplay, lvdisplay and df. Does fdisk give you any information about lvm?
4. Enlarge the small logical volume by 50 percent, and verify your work!
5. Take a look at other commands that start with vg* , pv* or lv*.
6. Create a mirror and a striped Logical Volume.
7. Convert a linear logical volume to a mirror.
8. Convert a mirror logical volume to a linear.
9. Create a snapshot of a Logical Volume, take a backup of the snapshot. Then delete some files on the Logical Volume, then restore your backup.
10. Move your volume group to another disk (keep the Logical Volumes mounted).
11. If time permits, split a Volume Group with vgsplit, then merge it again with vgmerge.

Chapter 39. iSCSI devices

Table of Contents

39.1. iSCSI terminology	345
39.2. iSCSI target installation	345
39.3. iSCSI target setup	346
39.4. iSCSI client initiator setup	348
39.5. using iSCSI devices	350
39.6. practice: iSCSI devices	351
39.7. solution: iSCSI devices	352

This chapter teaches you how to setup an **iSCSI target server** and an **iSCSI initiator client**.

39.1. iSCSI terminology

iSCSI is a protocol that enables SCSI over IP. This means that you can have local SCSI devices (like /dev/sdb) without having the storage hardware in the local computer.

The computer holding the physical storage hardware is called the **iSCSI Target**. Each individual addressable iSCSI device on the target server will get a **LUN number**.

The iSCSI client computer that is connecting to the Target server is called an **Initiator**. An initiator will send SCSI commands over IP instead of directly to the hardware. The Initiator will connect to the Target.

39.2. iSCSI target installation

Installing the software for the target server requires **iscsitarget** on Ubuntu and Debian, and an extra **iscsitarget-dkms** for the kernel modules only on Debian.

```
root@debby6:~# aptitude install iscsitarget
The following NEW packages will be installed:
  iscsitarget
0 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 69.4 kB of archives. After unpacking 262 kB will be used.
Get:1 http://ftp.belnet.be/debian/ squeeze/main iscsitarget i386 1.4.20.2-1 [69.4 kB]
Fetched 69.4 kB in 0s (415 kB/s)
Selecting previously deselected package iscsitarget.
(Reading database ... 36441 files and directories currently installed.)
Unpacking iscsitarget (from ../iscsitarget_1.4.20.2-1_i386.deb) ...
Processing triggers for man-db ...
Setting up iscsitarget (1.4.20.2-1) ...
iscsitarget not enabled in "/etc/default/iscsitarget", not starting... .. (warning).
```

On Debian 6 you will also need **aptitude install iscsitarget-dkms** for the kernel modules, on Debian 5 this is **aptitude install iscsitarget-modules-`uname -a`**. Ubuntu includes the kernel modules in the main package.

The iSCSI target server is disabled by default, so we enable it.

```
root@debby6:~# cat /etc/default/iscsitarget
ISCSITARGET_ENABLE=false
root@debby6:~# vi /etc/default/iscsitarget
root@debby6:~# cat /etc/default/iscsitarget
ISCSITARGET_ENABLE=true
```

39.3. iSCSI target setup

You can use LVM volumes (`/dev/md0/lvol0`), physical partitions (`/dev/sda`), raid devices (`/dev/md0`) or just plain files for storage. In this demo, we use files created with `dd`.

This screenshot shows how to create three small files (100MB, 200MB and 300MB).

```
root@debby6:~# mkdir /iscsi
root@debby6:~# dd if=/dev/zero of=/iscsi/lun1.img bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (105 MB) copied, 0.315825 s, 332 MB/s
root@debby6:~# dd if=/dev/zero of=/iscsi/lun2.img bs=1M count=200
200+0 records in
200+0 records out
209715200 bytes (210 MB) copied, 1.08342 s, 194 MB/s
root@debby6:~# dd if=/dev/zero of=/iscsi/lun3.img bs=1M count=300
300+0 records in
300+0 records out
314572800 bytes (315 MB) copied, 1.36209 s, 231 MB/s
```

We need to declare these three files as iSCSI targets in `/etc/iet/ietd.conf` (used to be `/etc/ietd.conf`).

```
root@debby6:/etc/iet# cp ietd.conf ietd.conf.original
root@debby6:/etc/iet# > ietd.conf
root@debby6:/etc/iet# vi ietd.conf
root@debby6:/etc/iet# cat ietd.conf
Target iqn.2010-02.be.linux-training:storage.lun1
  IncomingUser isuser hunter2
  OutgoingUser
  Lun 0 Path=/iscsi/lun1.img,Type=fileio
  Alias LUN1

Target iqn.2010-02.be.linux-training:storage.lun2
  IncomingUser isuser hunter2
  OutgoingUser
  Lun 0 Path=/iscsi/lun2.img,Type=fileio
  Alias LUN2

Target iqn.2010-02.be.linux-training:storage.lun3
  IncomingUser isuser hunter2
  OutgoingUser
  Lun 0 Path=/iscsi/lun3.img,Type=fileio
  Alias LUN3
```

We also need to add our devices to the `/etc/initiators.allow` file.

```
root@debby6:/etc/iet# cp initiators.allow initiators.allow.original
root@debby6:/etc/iet# > initiators.allow
root@debby6:/etc/iet# vi initiators.allow
root@debby6:/etc/iet# cat initiators.allow
iqn.2010-02.be.linux-training:storage.lun1
iqn.2010-02.be.linux-training:storage.lun2
iqn.2010-02.be.linux-training:storage.lun3
```

Time to start the server now:

```
root@debby6:/etc/iet# /etc/init.d/iscsitarget start
Starting iSCSI enterprise target service:.
```



```
.  
root@debby6:/etc/iet#
```

Verify activation of the storage devices in **/proc/net/iet**:

```
root@debby6:/etc/iet# cat /proc/net/iet/volume  
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3  
  lun:0 state:0 iotype:fileio iomode:wt blocks:614400 blocksize:\  
512 path:/iscsi/lun3.img  
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2  
  lun:0 state:0 iotype:fileio iomode:wt blocks:409600 blocksize:\  
512 path:/iscsi/lun2.img  
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1  
  lun:0 state:0 iotype:fileio iomode:wt blocks:204800 blocksize:\  
512 path:/iscsi/lun1.img  
root@debby6:/etc/iet# cat /proc/net/iet/session  
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3  
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2  
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1
```

39.4. iSCSI client initiator setup

First we install the iSCSI client software (on another computer than the target).

```
root@ubull104:~# aptitude install open-iscsi
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
The following NEW packages will be installed:
  open-iscsi open-iscsi-utils{a}
```

Then we set the iSCSI client to start automatically.

```
root@ubull104:/etc/iscsi# cp iscsid.conf iscsid.conf.original
root@ubull104:/etc/iscsi# vi iscsid.conf
root@ubull104:/etc/iscsi# grep ^node.startup iscsid.conf
node.startup = automatic
```

Or you could start it manually.

```
root@ubull104:/etc/iscsi/nodes# /etc/init.d/open-iscsi start
* Starting iSCSI initiator service iscsid
* Setting up iSCSI targets
root@ubull104:/etc/iscsi/nodes#
```

Now we can connect to the Target server and use **iscsiadm** to discover the devices it offers:

```
root@ubull104:/etc/iscsi# iscsiadm -m discovery -t st -p 192.168.1.31
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun2
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun1
192.168.1.31:3260,1 iqn.2010-02.be.linux-training:storage.lun3
```

We can use the same **iscsiadm** to edit the files in **/etc/iscsi/nodes/**.

```
root@ubull104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.authmethod --value=CHAP
root@ubull104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.username --value=isuser
root@ubull104:/etc/iscsi# iscsiadm -m node --targetname "iqn.2010-02.be.linu\
x-training:storage.lun1" --portal "192.168.1.31:3260" --op=update --name no\
de.session.auth.password --value=hunter2
```

Repeat the above for the other two devices.

Restart the initiator service to log in to the target.

```
root@ubull104:/etc/iscsi/nodes# /etc/init.d/open-iscsi restart
* Disconnecting iSCSI targets [ OK ]
* Stopping iSCSI initiator service [ OK ]
* Starting iSCSI initiator service iscsid [ OK ]
* Setting up iSCSI targets
```

Use **fdisk -l** to enjoy three new iSCSI devices.

```
root@ubull104:/etc/iscsi/nodes# fdisk -l 2> /dev/null | grep Disk
Disk /dev/sda: 17.2 GB, 17179869184 bytes
```

```
Disk identifier: 0x0001983f
Disk /dev/sdb: 209 MB, 209715200 bytes
Disk identifier: 0x00000000
Disk /dev/sdd: 314 MB, 314572800 bytes
Disk identifier: 0x00000000
Disk /dev/sdc: 104 MB, 104857600 bytes
Disk identifier: 0x00000000
```

The Target (the server) now shows active sessions.

```
root@debby6:/etc/iet# cat /proc/net/iet/session
tid:3 name:iqn.2010-02.be.linux-training:storage.lun3
  sid:5348024611832320 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
tid:2 name:iqn.2010-02.be.linux-training:storage.lun2
  sid:4785074624856576 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
tid:1 name:iqn.2010-02.be.linux-training:storage.lun1
  sid:5066549618344448 initiator:iqn.1993-08.org.debian:01:8983ed2d770
  cid:0 ip:192.168.1.35 state:active hd:none dd:none
root@debby6:/etc/iet#
```

39.5. using iSCSI devices

There is no difference between using SCSI or iSCSI devices once they are connected : partition, make filesystem, mount.

```
root@ubull104:/etc/iscsi/nodes# history | tail -13
 94 fdisk /dev/sdc
 95 fdisk /dev/sdd
 96 fdisk /dev/sdb
 97 mke2fs /dev/sdb1
 98 mke2fs -j /dev/sdc1
 99 mkfs.ext4 /dev/sdd1
100 mkdir /mnt/is1
101 mkdir /mnt/is2
102 mkdir /mnt/is3
103 mount /dev/sdb1 /mnt/is1
104 mount /dev/sdc1 /mnt/is2
105 mount /dev/sdd1 /mnt/is3
106 history | tail -13
root@ubull104:/etc/iscsi/nodes# mount | grep is
/dev/sdb1 on /mnt/is1 type ext2 (rw)
/dev/sdc1 on /mnt/is2 type ext3 (rw)
/dev/sdd1 on /mnt/is3 type ext4 (rw)
```

39.6. practice: iSCSI devices

1. Set up a target (using an LVM and a SCSI device) and an initiator that connects to both.

39.7. solution: iSCSI devices

1. Set up a target (using an LVM and a SCSI device) and an initiator that connects to both.

Decide (with a partner) on a computer to be the Target and another computer to be the Initiator.

On the Target computer:

First install `iscsitarget` using the standard tools for installing software in your distribution. Then use your knowledge from the previous chapter to setup a logical volume (`/dev/vg/lvol0`) and use the RAID chapter to setup `/dev/md0`. Then perform the following step:

```
vi /etc/default/iscsitarget (set enable to true)
```

Add your devices to `/etc/iet/ietf.conf`

```
root@debby6:/etc/iet# cat ietd.conf
Target iqn.2010-02.be.linux-training:storage.lun1
  IncomingUser isuser hunter2
  OutgoingUser
  Lun 0 Path=/dev/vg/lvol0,Type=fileio
  Alias LUN1
Target iqn.2010-02.be.linux-training:storage.lun2
  IncomingUser isuser hunter2
  OutgoingUser
  Lun 0 Path=/dev/md0,Type=fileio
  Alias LUN2
```

Add both devices to `/etc/iet/initiators.allow`

```
root@debby6:/etc/iet# cat initiators.allow
iqn.2010-02.be.linux-training:storage.lun1
iqn.2010-02.be.linux-training:storage.lun2
```

Now start the `iscsitarget` daemon and move over to the Initiator.

On the Initiator computer:

Install `open-iscsi` and start the daemon.

Then use `iscsiadm -m discovery -t st -p 'target-ip'` to see the `iscsi` devices on the Target.

Edit the files `/etc/iscsi/nodes/` as shown in the book. Then restart the `iSCSI` daemon and run `fdisk -l` to see the `iSCSI` devices.

Part XI. boot management

Chapter 40. bootloader

Table of Contents

40.1. boot terminology	355
40.2. grub	357
40.3. lilo	362
40.4. practice : bootloader	364
40.5. solution : bootloader	365

40.1. boot terminology

The exact order of things that happen when starting a computer system, depends on the hardware architecture (**Intel x86** is different from **Sun Sparc** etc), on the boot loader (**grub** is different from **lilo**) and on the operating system (**Linux, Solaris, BSD** etc). Most of this chapter is focused on booting **Linux** on **Intel x86** with **grub**.

post

A computer starts booting the moment you turn on the power (no kidding). This first process is called **post** or **power on self test**. If all goes well then this leads to the **bios**. If all goes not so well, then you might hear nothing, or hear beeping, or see an error message on the screen, or maybe see smoke coming out of the computer (burning hardware smells bad!).

bios

All **Intel x86** computers will have a **basic input/output system** or **bios** to detect, identify and initialize hardware. The **bios** then goes looking for a **boot device**. This can be a floppy, hard disk, cdrom, network card or usb drive.

During the **bios** you can see a message on the screen telling you which key (often **Del** or **F2**) to press to enter the **bios** setup.

PhoenixBIOS Setup Utility							
Main	Advanced	Security	Power	Boot	Exit		
System Time:					[18]:06:55]	Item Specific Help	
System Date:					[06/01/2009]	<Tab>, <Shift-Tab>, or <Enter> selects field.	
Legacy Diskette A:					[1.44/1.25 MB 3½"]		
Legacy Diskette B:					[Disabled]		
▶ Primary Master					[None]		
▶ Primary Slave					[None]		
▶ Secondary Master					[UMware Virtual ID]		
▶ Secondary Slave					[None]		
▶ Keyboard Features							
System Memory:					640 KB		
Extended Memory:					261120 KB		
Boot-time Diagnostic Screen:					[Disabled]		
F1	Help	↑↓	Select Item	-/+	Change Values	F9	Setup Defaults
Esc	Exit	↔	Select Menu	Enter	Select ▶ Sub-Menu	F10	Save and Exit

openboot

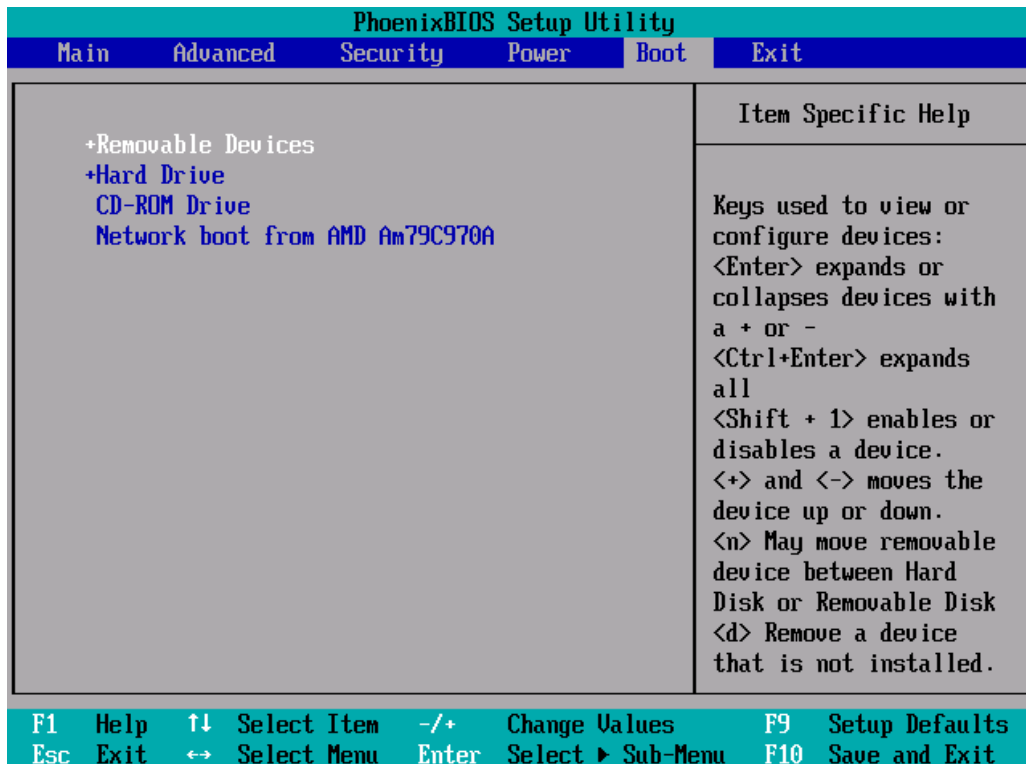
Sun **sparc** systems start with **openboot** to test the hardware and to boot the operating system. **Bill Callkins** explains **openboot** in his Solaris System Administration books. The details of **openboot** are not the focus of this course.

boot password

The **bios** allows you to set a password. Do not forget this password, or you will have to open up the hardware to reset it. You can sometimes set a password to boot the system, and another password to protect the **bios** from being modified.

boot device

The **bios** will look for a **boot device** in the order configured in the bios setup. Usually an operating system on a production server boots of a hard disk.



master boot record

The **master boot record** or **mbr** is the first sector of a hard disk. The partitioning of a disk in **primary** partitions, and the active partition are defined in the **mbr**.

The **mbr** is 512 bytes long and can be copied with **dd**.

```
dd if=/dev/sda of=bootsect.mbr count=1 bs=512
```

bootloader

The **mbr** is executed by the **bios** and contains either (a small) **bootloader** or code to load a **bootloader**.

Looking at the **mbr** with **od** can reveal information about the **bootloader**.

```
paul@laika:~$ sudo dd if=/dev/sda count=1 bs=16 skip=24 2>/dev/null|od -c
0000000 376  G  R  U  B      \0  G  e  o  m  \0  H  a  r  d
0000020
```

There are a variety of bootloaders available, most common on **Intel** architecture is **grub**, which is replacing **lilo** in many places. When installing **Linux** on **sparc** architecture, you can choose **siloboot**, **Itanium** systems can use **elilo**, **IBM S/390** and **zSeries** use **z/IPL**, **Alpha** uses **milo** and **PowerPC** architectures use **yaboot** (yet another boot loader).

Bootable cd's and dvd's often use **syslinux**.

kernel

The goal of all this is to load an operating system, or rather the **kernel** of an operating system. A typical bootloader like **grub** will copy a kernel from hard disk to memory, and will then hand control of the computer to the kernel (execute the kernel).

Once the Linux kernel is loaded, the bootloader turns control over to it. From that moment on, the kernel is in control of the system. After discussing bootloaders, we continue with the init system that starts all the daemons.

40.2. grub

about grub

The most common bootloader on linux systems today is **grub**. On almost all Intel based systems grub is replacing **lilo** (the **Linux loader**). Even **Solaris** switched to **grub** on **x86** architecture.

One of the big advantages of **grub** over **lilo** is the capability to change the configuration during boot (by pressing **e** to edit the boot command line).

/boot/grub/menu.lst

grub's configuration file is called **menu.lst** and is located in **/boot/grub**. The screenshot below show the location and size of **menu.lst** on Debian.

```
root@barry:~# ls -l /boot/grub/menu.lst
-rw-r--r-- 1 root root 5155 2009-03-31 18:20 /boot/grub/menu.lst
```

/boot/grub/grub.conf

Some distributions like Red Hat Enterprise Linux 5 use **grub.conf** and provide a symbolic link to **menu.lst**. This is the same file, only the name changed from **grub.conf** to **menu.lst**. Notice also in this screenshot that this file is a lot smaller on Red Hat.

```
[root@RHEL52 grub]# ls -l grub.conf menu.lst
-rw----- 1 root root 1346 Jan 21 04:20 grub.conf
lrwxrwxrwx 1 root root  11 Oct 11  2008 menu.lst -> ./grub.conf
```

menu commands

The **menu commands** always have to be at the top of **grub**'s configuration file.

default

The **default** command sets a default **entry** to start. The first **entry** has number 0.

```
default 0
```

fallback

In case the **default** does not boot, use the **fallback** entry instead.

```
fallback 1
```

timeout

The **timeout** will wait a number of seconds before booting the **default** entry.

```
timeout 5
```

hiddenmenu

The **hiddenmenu** will hide the **grub** menu unless the user presses **Esc** before the **timeout** expires.

```
hiddenmenu
```

title

With **title** we can start a new **entry** or **stanza**.

```
title Debian Lenny
```

password

You can add a **password** to prevent interactive selection of a boot environment while **grub** is running.

```
password --md5 $1$Ec.id/$T2C2ahI/EG3WRRsmmu/HN/
```

Use the **grub** interactive shell to create the password hash.

```
grub> md5crypt
```

```
Password: *****  
Encrypted: $1$Ec.id/$T2C2ahI/EG3WRRsmmu/HN/
```

stanza commands

Every **operating system** or **kernel** that you want to boot with **grub** will have a **stanza** aka an **entry** of a couple of lines. Listed here are some of the common **stanza** commands.

boot

Technically the **boot** command is only mandatory when running the **grub command line**. This command does not have any parameters and can only be set as the last command of a stanza.

```
boot
```

kernel

The **kernel** command points to the location of the kernel. To boot Linux this means booting a **gzip** compressed **zImage** or **bzip2** compressed **bzImage**.

This screenshot shows a typical **kernel** command used to load a Debian kernel.

```
kernel /boot/vmlinuz-2.6.17-2-686 root=/dev/hda1 ro
```

And this is how Red Hat uses the **kernel** command.

```
kernel /vmlinuz-2.6.18-128.el5 ro root=/dev/VolGroup00/LogVol100 rhgb quiet
```

initrd

Many **Linux** installations will need an **initial ramdisk** at boot time. This can be set in **grub** with the **initrd** command.

Here a screenshot of Debian 4.0

```
initrd /boot/initrd.img-2.6.17-2-686
```

And the same for Red Hat Enterprise Linux 5.3

```
initrd /initrd-2.6.18-128.el5.img
```

root

The **root** command accepts the root device as a parameter.

The **root** command will point to the hard disk and partition to use, with **hd0** as the first hard disk device and **hd1** as the second hard disk device. The same numbering is used for partitions, so **hd0,0** is the first partition on the first disk and **hd0,1** is the second partition on that disk.

```
root (hd0,0)
```

savedefault

The **savedefault** command can be used together with **default saved** as a menu command. This combination will set the currently booted stanza as the next default stanza to boot.

```
default saved
timeout 10

title Linux
root (hd0,0)
kernel /boot/vmlinuz
savedefault
```

```
title DOS
root (hd0,1)
makeactive
chainloader +1
savedefault
```

chainloading

With **grub** booting, there are two choices: loading an operating system or **chainloading** another bootloader. The **chainloading** feature of grub loads the bootsector of a partition (that contains an operating system).

Some older operating systems require a **primary partition** that is set as **active**. Only one partition can be set **active** so **grub** can do this on the fly just before **chainloading**.

This screenshot shows how to set the first primary partition **active** with **grub**.

```
root (hd0,0)
makeactive
```

Chainloading refers to grub loading another operating system's bootloader. The **chainloader** switch receives one option: the number of sectors to read and boot. For **DOS** and **OS/2** one sector is enough. Note that **DOS** requires the boot/root partition to be active!

Here is a complete example to **chainload** an old operating system.

```
title MS-DOS 6.22
root (hd0,1)
makeactive
chainloader +1
```

stanza examples

This is a screenshot of a typical **Debian 4.0** stanza.

```
title Debian GNU/Linux, kernel 2.6.17-2-686
root (hd0,0)
kernel /boot/vmlinuz-2.6.17-2-686 root=/dev/hda1 ro
initrd /boot/initrd.img-2.6.17-2-686
```

Here a screenshot of a typical **Red Hat Enterprise Linux** stanza.

```
title Red Hat Enterprise Linux Server (2.6.18-128.el5)
root (hd0,0)
kernel /vmlinuz-2.6.18-98.el5 ro root=/dev/VolGroup00/LogVol100 rhgb quiet
initrd /initrd-2.6.18-98.el5.img
```

editing grub at boot time

At boot time, when the **grub** menu is displayed, you can type **e** to edit the current stanza. This enables you to add parameters to the kernel.

One such parameter, useful when you lost the root password, is **single**. This will boot the kernel in single user mode (although some distributions will still require you to type the root password).

```
kernel /boot/vmlinuz-2.6.17-2-686 root=/dev/hda1 ro single
```

Another option to reset a root password is to use an **init=/bin/bash** parameter.

```
kernel /boot/vmlinuz-2.6.17-2-686 root=/dev/hda1 ro init=/bin/bash
```

installing grub

Run the **grub-install** command to install **grub**. The command requires a destination for overwriting the **boot sector** or **mbr**.

```
# grub-install /dev/hda
```

40.3. lilo

Linux loader

lilo used to be the most used Linux bootloader, but is steadily being replaced in **x86** with **grub**.

lilo.conf

Here is an example of a typical **lilo.conf** file. The **delay** switch receives a number in tenths of a second. So the delay below is three seconds, not thirty!

```
boot = /dev/hda
delay = 30

image = /boot/vmlinuz
  root = /dev/hda1
  label = Red Hat 5.2

image = /boot/vmlinuz
  root = /dev/hda2
  label = S.U.S.E. 8.0
```



```
other = /dev/hda4  
table = /dev/hda  
label = MS-DOS 6.22
```

The configuration file shows three example stanzas. The first one boots Red Hat from the first partition on the first disk (hda1). The second stanza boots Suse 8.0 from the next partition. The last one loads MS-DOS.

40.4. practice : bootloader

1. Make a copy of the kernel, initrd and System.map files in /boot. Put the copies also in /boot but replace 2.6.x with 3.0 (just imagine that Linux 3.0 is out.).
2. Add a stanza in grub for the 3.0 files. Make sure the title is different.
3. Set the boot menu timeout to 30 seconds.
4. Reboot and test the new stanza.

40.5. solution : bootloader

1. Make a copy of the kernel, initrd and System.map files in /boot. Put the copies also in /boot but replace 2.6.x with 3.0 (just imagine that Linux 3.0 is out.).

```
cd /boot
cp vmlinuz-2.6.18-8.e15 vmlinuz-3.0
cp initrd-2.6.18-8.e15.img initrd-3.0.img
cp System.map-2.6.18-8.e15 System.map-3.0
```

Do not forget the initrd file ends in .img .

2. Add a stanza in grub for the 3.0 files. Make sure the title is different.

```
[root@RHEL5 ~]# grep 3.0 /boot/grub/menu.lst
title Red Hat Enterprise Linux Server (3.0)
 kernel /vmlinuz-3.0 ro root=/dev/VolGroup00/LogVol100 rhgb quiet
 initrd /initrd-3.0.img
```

3. Set the boot menu timeout to 30 seconds.

```
[root@RHEL5 ~]# grep time /boot/grub/menu.lst
timeout=30
```

4. Reboot and test the new stanza.

Chapter 41. init and runlevels

Table of Contents

41.1. about sysv init	367
41.2. system init(ialization)	367
41.3. daemon or demon ?	371
41.4. starting and stopping daemons	372
41.5. chkconfig	372
41.6. update-rc.d	374
41.7. bum	375
41.8. runlevels	376
41.9. practice: init	379
41.10. solution : init	380

41.1. about sysv init

Many Linux distributions use **init** scripts to start daemons in the same way that **Unix System V** did. This chapter will explain in detail how that works.

Init starts daemons by using scripts, where each script starts one daemon, and where each script waits for the previous script to finish. This serial process of starting daemons is slow, and although slow booting is not a problem on servers where uptime is measured in years, the recent uptake of Linux on the desktop results in user complaints.

To improve Linux startup speed, **Canonical** has developed **upstart**, which was first used in Ubuntu. Solaris also used init up to Solaris 9, for Solaris 10 **Sun** has developed **Service Management Facility**. Both systems start daemons in parallel and can replace the SysV init scripts. There is also an ongoing effort to create **initng** (init next generation).

41.2. system initialization

process id 1

The kernel receives system control from the bootloader. After a while the kernel starts the **init daemon**. The **init** daemon (**/sbin/init**) is the first daemon that is started and receives **process id 1** (PID 1). **Init** never dies.

configuration in /etc/inittab

When **/sbin/init** is started, it will first read its configuration file **/etc/inittab**. In that file, it will look for the value of **initdefault** (3 in the screenshot below).

```
[paul@rhel4 ~]$ grep ^id /etc/inittab
id:3:initdefault:
```

initdefault

The value found in **initdefault** indicates the default **runlevel**. Some Linux distributions have a brief description of runlevels in **/etc/inittab**, like here on Red Hat Enterprise Linux 4.

```
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you don't have network)
```

```
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
```

Runlevel 0 means the system is shutting down. **Runlevel 1** is used for troubleshooting, only the root user can log on, and only at the console. **Runlevel 3** is typical for servers, whereas **runlevel 5** is typical for desktops (graphical logon). Besides runlevels 0, 1 and 6, the use may vary depending on the distribution. Debian and derived Linux systems have full network and GUI logon on runlevels 2 to 5. So always verify the proper meaning of runlevels on your system.

sysinit script

/etc/rc.d/rc.sysinit

The next line in **/etc/inittab** in Red Hat and derivatives is the following.

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

This means that independent of the selected runlevel, **init** will run the **/etc/rc.d/rc.sysinit** script. This script initializes hardware, sets some basic environment, populates **/etc/mtab** while mounting file systems, starts swap and more.

```
[paul@rhel ~]$ egrep -e"^# Ini" -e"^# Sta" -e"^# Che" /etc/rc.d/rc.sysinit
# Check SELinux status
# Initialize hardware
# Start the graphical boot, if necessary; /usr may not be mounted yet...
# Initialiaze ACPI bits
# Check filesystems
# Start the graphical boot, if necessary and not done yet.
# Check to see if SELinux requires a relabel
# Initialize pseudo-random number generator
# Start up swapping.
# Initialize the serial ports.
```

*That **egrep** command could also have been written with **grep** like this :*

```
grep "^# \(Ini\|Sta\|Che\)".
```

/etc/init.d/rcS

Debian has the following line after **initdefault**.

```
si::sysinit:/etc/init.d/rcS
```

The **/etc/init.d/rcS** script will always run on Debian (independent of the selected runlevel). The script is actually running all scripts in the **/etc/rcS.d/** directory in alphabetical order.

```
root@barry:~# cat /etc/init.d/rcS
#!/bin/sh
#
# rcS
#
# Call all S??* scripts in /etc/rcS.d/ in numerical/alphabetical order
#

exec /etc/init.d/rc S
```

rc scripts

Init will continue to read **/etc/inittab** and meets this section on Debian Linux.

```
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
```

On Red Hat Enterprise Linux it is identical except **init.d** is **rc.d**.

```
10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
13:3:wait:/etc/rc.d/rc 3
14:4:wait:/etc/rc.d/rc 4
15:5:wait:/etc/rc.d/rc 5
16:6:wait:/etc/rc.d/rc 6
```

In both cases, this means that **init** will start the rc script with the runlevel as the only parameter. Actually **/etc/inittab** has fields separated by colons. The second field determines the runlevel in which this line should be executed. So in both cases, only one line of the seven will be executed, depending on the runlevel set by **initdefault**.

rc directories

When you take a look any of the **/etc/rcX.d/** directories, then you will see a lot of (links to) scripts whose name starts with either uppercase K or uppercase S.

```
[root@RHEL52 rc3.d]# ls -l | tail -4
lrwxrwxrwx 1 root root 19 Oct 11 2008 S98haldaemon -> ../init.d/haldaemon
lrwxrwxrwx 1 root root 19 Oct 11 2008 S99firstboot -> ../init.d/firstboot
```

```
lrwxrwxrwx 1 root root 11 Jan 21 04:16 S99local -> ../rc.local
lrwxrwxrwx 1 root root 16 Jan 21 04:17 S99smartd -> ../init.d/smartd
```

The **/etc/rcX.d/** directories only contain links to scripts in **/etc/init.d/**. Links allow for the script to have a different name. When entering a runlevel, all scripts that start with uppercase K or uppercase S will be started in alphabetical order. Those that start with K will be started first, with **stop** as the only parameter. The remaining scripts with S will be started with **start** as the only parameter.

All this is done by the **/etc/rc.d/rc** script on Red Hat and by the **/etc/init.d/rc** script on Debian.

mingetty

mingetty in /etc/inittab

Almost at the end of **/etc/inittab** there is a section to start and **respawn** several **mingetty** daemons.

```
[root@RHEL4b ~]# grep getty /etc/inittab
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

mingetty and /bin/login

This **/sbin/mingetty** will display a message on a virtual console and allow you to type a userid. Then it executes the **/bin/login** command with that userid. The **/bin/login** program will verify whether that user exists in **/etc/passwd** and prompt for (and verify) a password. If the password is correct, **/bin/login** passes control to the shell listed in **/etc/passwd**.

respawning mingetty

The **mingetty** daemons are started by **init** and watched until they die (user exits the shell and is logged out). When this happens, the **init** daemon will **respawn** a new **mingetty**. So even if you **kill** a **mingetty** daemon, it will be restarted automatically.

This example shows that **init** respawns **mingetty** daemons. Look at the PID's of the last two **mingetty** processes.


```
[root@RHEL52 ~]# ps -C mingetty
  PID TTY          TIME CMD
 2407 tty1        00:00:00 mingetty
 2408 tty2        00:00:00 mingetty
 2409 tty3        00:00:00 mingetty
 2410 tty4        00:00:00 mingetty
 2411 tty5        00:00:00 mingetty
 2412 tty6        00:00:00 mingetty
```

When we **kill** the last two mingettys, then **init** will notice this and start them again (with a different PID).

```
[root@RHEL52 ~]# kill 2411 2412
[root@RHEL52 ~]# ps -C mingetty
  PID TTY          TIME CMD
 2407 tty1        00:00:00 mingetty
 2408 tty2        00:00:00 mingetty
 2409 tty3        00:00:00 mingetty
 2410 tty4        00:00:00 mingetty
 2821 tty5        00:00:00 mingetty
 2824 tty6        00:00:00 mingetty
```

disabling a mingetty

You can disable a mingetty for a certain tty by removing the runlevel from the second field in its line in `/etc/inittab`. Don't forget to tell `init` about the change of its configuration file with **kill -1 1**.

The example below shows how to disable mingetty on tty3 to tty6 in runlevels 4 and 5.

```
[root@RHEL52 ~]# grep getty /etc/inittab
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:23:respawn:/sbin/mingetty tty3
4:23:respawn:/sbin/mingetty tty4
5:23:respawn:/sbin/mingetty tty5
6:23:respawn:/sbin/mingetty tty6
```

41.3. daemon or demon ?

A **daemon** is a process that runs in background, without a link to a GUI or terminal. Daemons are usually started at system boot, and stay alive until the system shuts down. In more recent technical writings, daemons are often referred to as **services**.

Unix **daemons** are not to be confused with demons. **Evi Nemeth**, co-author of the UNIX System Administration Handbook has the following to say about daemons:

Many people equate the word "daemon" with the word "demon", implying some kind of satanic connection between UNIX and the underworld. This is an egregious

misunderstanding. "Daemon" is actually a much older form of "demon"; daemons have no particular bias towards good or evil, but rather serve to help define a person's character or personality. The ancient Greeks' concept of a "personal daemon" was similar to the modern concept of a "guardian angel"

41.4. starting and stopping daemons

The K and S scripts are links to the real scripts in `/etc/init.d/`. These can also be used when the system is running to start and stop daemons (or services). Most of them accept the following parameters: start, stop, restart, status.

For example in this screenshot we restart the samba daemon.

```
root@laika:~# /etc/init.d/samba restart
* Stopping Samba daemons... [ OK ]
* Starting Samba daemons... [ OK ]
```

You can achieve the same result on RHEL/Fedora with the **service** command.

```
[root@RHEL4b ~]# service smb restart
Shutting down SMB services: [ OK ]
Shutting down NMB services: [ OK ]
Starting SMB services: [ OK ]
Starting NMB services: [ OK ]
```

You might also want to take a look at **chkconfig**, **update-rc.d**.

41.5. chkconfig

The purpose of **chkconfig** is to relieve system administrators of manually managing all the links and scripts in `/etc/init.d` and `/etc/rcX.d/`.

chkconfig --list

Here we use **chkconfig** to list the status of a service in the different runlevels. You can see that the **crond** daemon (or service) is only activated in runlevels 2 to 5.

```
[root@RHEL52 ~]# chkconfig --list crond
crond          0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

When you compare the screenshot above with the one below, you can see that **off** equals to a K link to the script, whereas **on** equals to an S link.

```
[root@RHEL52 etc]# find ./rc?.d/ -name \*cron* -exec ls -l {} \;|cut -b40-
./rc0.d/K60cron* -> ../init.d/cron*
./rc1.d/K60cron* -> ../init.d/cron*
./rc2.d/S90cron* -> ../init.d/cron*
./rc3.d/S90cron* -> ../init.d/cron*
./rc4.d/S90cron* -> ../init.d/cron*
./rc5.d/S90cron* -> ../init.d/cron*
./rc6.d/K60cron* -> ../init.d/cron*
```

runlevel configuration

Here you see how to use `chkconfig` to disable (or enable) a service in a certain runlevel.

This screenshot shows how to disable **crond** in runlevel 3.

```
[root@RHEL52 ~]# chkconfig --level 3 crond off
[root@RHEL52 ~]# chkconfig --list crond
crond          0:off 1:off 2:on 3:off 4:on 5:on 6:off
```

This screenshot shows how to enable **crond** in runlevels 3 and 4.

```
[root@RHEL52 ~]# chkconfig --level 34 crond on
[root@RHEL52 ~]# chkconfig --list crond
crond          0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

chkconfig configuration

Every script in `/etc/init.d/` can have (comment) lines to tell `chkconfig` what to do with the service. The line with **# chkconfig:** contains the runlevels in which the service should be started (2345), followed by the priority for start (90) and stop (60).

```
[root@RHEL52 ~]# head -9 /etc/init.d/crond | tail -5
# chkconfig: 2345 90 60
# description: cron is a standard UNIX program that runs user-specified
#                programs at periodic scheduled times. vixie cron adds a
#                number of features to the basic UNIX cron, including better
#                security and more powerful configuration options.
```

enable and disable services

Services can be enabled or disabled in all runlevels with one command. Runlevels 0, 1 and 6 are always stopping services (or calling the scripts with **stop**) even when their name starts with uppercase S.

```
[root@RHEL52 ~]# chkconfig crond off
```

```
[root@RHEL52 ~]# chkconfig --list crond
crond          0:off  1:off  2:off  3:off  4:off  5:off  6:off
[root@RHEL52 ~]# chkconfig crond on
[root@RHEL52 ~]# chkconfig --list crond
crond          0:off  1:off  2:on   3:on   4:on   5:on   6:off
```

41.6. update-rc.d

about update-rc.d

The Debian equivalent of **chkconfig** is called **update-rc.d**. This tool is designed for use in scripts, if you prefer a graphical tool then look at **bum**.

When there are existing links in **/etc/rcX.d/** then **update-rc.d** does not do anything. This is to avoid that post installation scripts using **update-rc.d** are overwriting changes made by a system administrator.

```
root@barry:~# update-rc.d cron remove
update-rc.d: /etc/init.d/cron exists during rc.d purge (use -f to force)
```

As you can see in the next screenshot, nothing changed for the cron daemon.

```
root@barry:~# find /etc/rc?.d/ -name '*cron' -exec ls -l {} \;|cut -b44-
/etc/rc0.d/K11cron -> ../init.d/cron
/etc/rc1.d/K11cron -> ../init.d/cron
/etc/rc2.d/S89cron -> ../init.d/cron
/etc/rc3.d/S89cron -> ../init.d/cron
/etc/rc4.d/S89cron -> ../init.d/cron
/etc/rc5.d/S89cron -> ../init.d/cron
/etc/rc6.d/K11cron -> ../init.d/cron
```

removing a service

Here we remove **cron** from all runlevels. Remember that the proper way to disable a service is to put **K** scripts oin all runlevels!

```
root@barry:~# update-rc.d -f cron remove
Removing any system startup links for /etc/init.d/cron ...
  /etc/rc0.d/K11cron
  /etc/rc1.d/K11cron
  /etc/rc2.d/S89cron
  /etc/rc3.d/S89cron
  /etc/rc4.d/S89cron
  /etc/rc5.d/S89cron
  /etc/rc6.d/K11cron
root@barry:~# find /etc/rc?.d/ -name '*cron' -exec ls -l {} \;|cut -b44-
root@barry:~#
```

enable a service

This screenshot shows how to use **update-rc.d** to enable a service in runlevels 2, 3, 4 and 5 and disable the service in runlevels 0, 1 and 6.

```
root@barry:~# update-rc.d cron defaults
Adding system startup for /etc/init.d/cron ...
/etc/rc0.d/K20cron -> ../init.d/cron
/etc/rc1.d/K20cron -> ../init.d/cron
/etc/rc6.d/K20cron -> ../init.d/cron
/etc/rc2.d/S20cron -> ../init.d/cron
/etc/rc3.d/S20cron -> ../init.d/cron
/etc/rc4.d/S20cron -> ../init.d/cron
/etc/rc5.d/S20cron -> ../init.d/cron
```

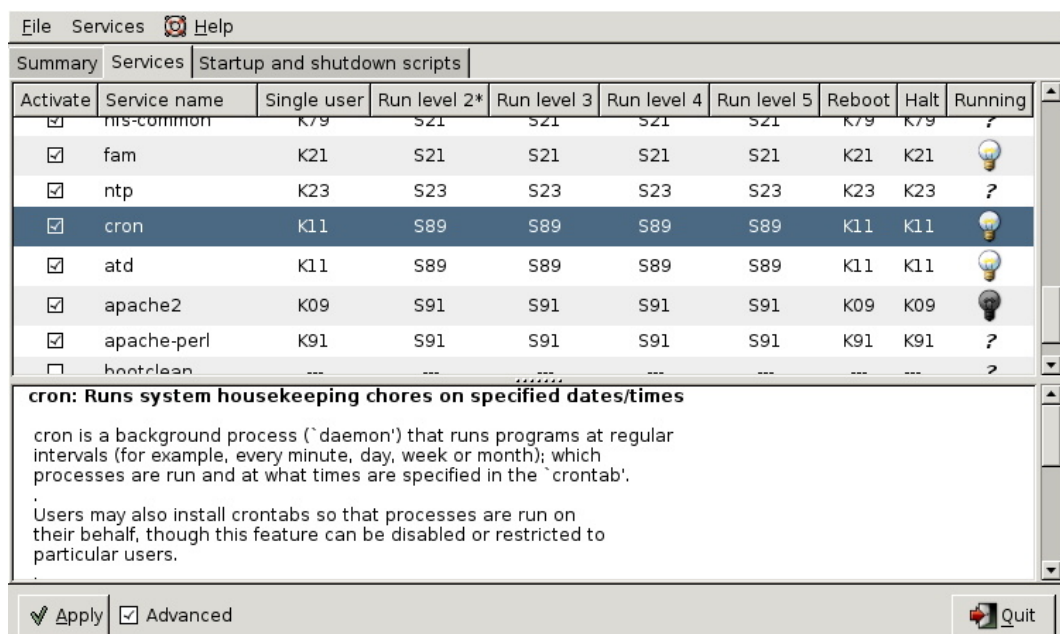
customize a service

And here is an example on how to set your custom configuration for the cron daemon.

```
root@barry:~# update-rc.d -n cron start 11 2 3 4 5 . stop 89 0 1 6 .
Adding system startup for /etc/init.d/cron ...
/etc/rc0.d/K89cron -> ../init.d/cron
/etc/rc1.d/K89cron -> ../init.d/cron
/etc/rc6.d/K89cron -> ../init.d/cron
/etc/rc2.d/S11cron -> ../init.d/cron
/etc/rc3.d/S11cron -> ../init.d/cron
/etc/rc4.d/S11cron -> ../init.d/cron
/etc/rc5.d/S11cron -> ../init.d/cron
```

41.7. bum

This screenshot shows **bum** in advanced mode.



41.8. runlevels

display the runlevel

You can see your current runlevel with the **runlevel** or **who -r** commands.

The **runlevel** command is typical Linux and will output the previous and the current runlevel. If there was no previous runlevel, then it will mark it with the letter N.

```
[root@RHEL4b ~]# runlevel
N 3
```

The history of **who -r** dates back to Seventies Unix, it still works on Linux.

```
[root@RHEL4b ~]# who -r
run-level 3 Jul 28 09:15 last=S
```

changing the runlevel

You can switch to another runlevel with the **telinit** command. On Linux **/sbin/telinit** is usually a (hard) link to **/sbin/init**.

This screenshot shows how to switch from runlevel 2 to runlevel 3 without reboot.

```
root@barry:~# runlevel
N 2
root@barry:~# init 3
root@barry:~# runlevel
2 3
```

/sbin/shutdown

The **shutdown** command is used to properly shut down a system.

Common switches used with **shutdown** are **-a**, **-t**, **-h** and **-r**.

The **-a** switch forces **/sbin/shutdown** to use **/etc/shutdown.allow**. The **-t** switch is used to define the number of seconds between the sending of the **TERM** signal and the **KILL** signal. The **-h** switch halts the system instead of changing to runlevel 1. The **-r** switch tells **/sbin/shutdown** to reboot after shutting down.

This screenshot shows how to use **shutdown** with five seconds between **TERM** and **KILL** signals.

```
root@barry:~# shutdown -t5 -h now
```

The **now** is the time argument. This can be **+m** for the number of minutes to wait before shutting down (with **now** as an alias for **+0**). The command will also accept **hh:mm** instead of **+m**.

halt, reboot and poweroff

The binary **/sbin/reboot** is the same as **/sbin/halt** and **/sbin/poweroff**. Depending on the name we use to call the command, it can behave differently.

When in runlevel 0 or 6 **halt**, **reboot** and **poweroff** will tell the kernel to **halt**, **reboot** or **poweroff** the system.

When not in runlevel 0 or 6, typing **reboot** as root actually calls the **shutdown** command with the **-r** switch and typing **poweroff** will switch off the power when halting the system.

/var/log/wtmp

halt, **reboot** and **poweroff** all write to **/var/log/wtmp**. To look at **/var/log/wtmp**, we need to use **th last**.

```
[root@RHEL52 ~]# last | grep reboot
reboot    system boot    2.6.18-128.el5    Fri May 29 11:44    (192+05:01)
reboot    system boot    2.6.18-128.el5    Wed May 27 12:10    (06:49)
reboot    system boot    2.6.18-128.el5    Mon May 25 19:34    (1+15:59)
reboot    system boot    2.6.18-128.el5    Mon Feb  9 13:20    (106+21:13)
```

Ctrl-Alt-Del

When **rc** is finished starting all those scripts, **init** will continue to read **/etc/inittab**. The next line is about what to do when the user hits **Ctrl-Alt-Delete** on the keyboard.

Here is what Debian 4.0 does.

```
root@barry:~# grep -i ctrl /etc/inittab
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

Which is very similar to the default Red Hat Enterprise Linux 5.2 action.

```
[root@RHEL52 ~]# grep -i ctrl /etc/inittab
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

One noticeable difference is that Debian forces shutdown to use `/etc/shutdown.allow`, where Red Hat allows everyone to invoke `shutdown` pressing **Ctrl-Alt-Delete**.

UPS and loss of power

```
[root@RHEL52 ~]# grep ^p /etc/inittab
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
```

It will read commands on what to execute in case of **powerfailure**, **powerok** and **Ctrl-Alt-Delete**. The init process never stops keeping an eye on power failures and that triple key combo.

```
root@barry:~# grep ^p /etc/inittab
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop
```


41.9. practice: init

1. Change **/etc/inittab** so that only two **mingetty**'s are respawned. Kill the other **mingetty**'s and verify that they don't come back.
2. Use the Red Hat Enterprise Linux virtual machine. Go to runlevel 5, display the current and previous runlevel, then go back to runlevel 3.
3. Is the **sysinit** script on your computers setting or changing the **PATH** environment variable ?
4. List all **init.d** scripts that are started in runlevel 2.
5. Write a script that acts like a daemon script in **/etc/init.d/**. It should have a case statement to act on start/stop/restart and status. Test the script!
6. Use **chkconfig** to setup your script to start in runlevels 3,4 and 5, and to stop in any other runlevel.

41.10. solution : init

1. Change **/etc/inittab** so that only two mingetty's are respawned. Kill the other **mingetty**'s and verify that they don't come back.

Killing the mingetty's will result in init respawning them. You can edit **/etc/inittab** so it looks like the screenshot below. Don't forget to also run **kill -1 1**.

```
[root@RHEL5 ~]# grep tty /etc/inittab
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2:respawn:/sbin/mingetty tty3
4:2:respawn:/sbin/mingetty tty4
5:2:respawn:/sbin/mingetty tty5
6:2:respawn:/sbin/mingetty tty6
[root@RHEL5 ~]#
```

2. Use the Red Hat Enterprise Linux virtual machine. Go to runlevel 5, display the current and previous runlevel, then go back to runlevel 3.

```
init 5 (watch the console for the change taking place)
runlevel
init 3 (again you can follow this on the console)
```

3. Is the sysinit script on your computers setting or changing the PATH environment variable ?

On Red Hat, grep for PATH in **/etc/rc.sysinit**, on Debian/Ubuntu check **/etc/rc.local** and **/etc/init.d/rc.local**. The answer is probably no, but on RHEL5 the **rc.sysinit** script does set the HOSTNAME variable.

```
[root@RHEL5 etc]# grep HOSTNAME rc.sysinit
```

4. List all init.d scripts that are started in runlevel 2.

```
root@RHEL5 ~# chkconfig --list | grep '2:on'
```

5. Write a script that acts like a daemon script in **/etc/init.d/**. It should have a case statement to act on start/stop/restart and status. Test the script!

The script could look something like this.

```
#!/bin/bash
#
# chkconfig: 345 99 01
# description: pold demo script
#
# /etc/init.d/pold
```

```
#
case "$1" in
start)
    echo -n "Starting pold..."
    sleep 1;
    touch /var/lock/subsys/pold
    echo "done."
    echo pold started >> /var/log/messages
    ;;
stop)
    echo -n "Stopping pold..."
    sleep 1;
    rm -rf /var/lock/subsys/pold
    echo "done."
    echo pold stopped >> /var/log/messages
    ;;
*)
    echo "Usage: /etc/init.d/pold {start|stop}"
    exit 1
    ;;
esac
exit 0
```

The **touch /var/lock/subsys/pold** is mandatory and must be the same filename as the script name, if you want the stop sequence (the K01pold link) to be run.

6. Use **chkconfig** to setup your script to start in runlevels 3,4 and 5, and to stop in any other runlevel.

```
chkconfig --add pold
```

The command above will only work when the **# chkconfig:** and **# description:** lines in the pold script are there.

Part XII. system management

Chapter 42. scheduling

Table of Contents

42.1. one time jobs with at	384
42.2. cron	386
42.3. practice : scheduling	388
42.4. solution : scheduling	389

Linux administrators use the **at** to schedule one time jobs. Recurring jobs are better scheduled with **cron**. The next two sections will discuss both tools.

42.1. one time jobs with at

at

Simple scheduling can be done with the **at** command. This screenshot shows the scheduling of the date command at 22:01 and the sleep command at 22:03.

```
root@laika:~# at 22:01
at> date
at> <EOT>
job 1 at Wed Aug 1 22:01:00 2007
root@laika:~# at 22:03
at> sleep 10
at> <EOT>
job 2 at Wed Aug 1 22:03:00 2007
root@laika:~#
```

In real life you will hopefully be scheduling more useful commands ;-)

atq

It is easy to check when jobs are scheduled with the **atq** or **at -l** commands.

```
root@laika:~# atq
1      Wed Aug 1 22:01:00 2007 a root
2      Wed Aug 1 22:03:00 2007 a root
root@laika:~# at -l
1      Wed Aug 1 22:01:00 2007 a root
2      Wed Aug 1 22:03:00 2007 a root
root@laika:~#
```

The **at** command understands English words like tomorrow and teatime to schedule commands the next day and at four in the afternoon.

```
root@laika:~# at 10:05 tomorrow
at> sleep 100
at> <EOT>
job 5 at Thu Aug 2 10:05:00 2007
root@laika:~# at teatime tomorrow
at> tea
at> <EOT>
job 6 at Thu Aug 2 16:00:00 2007
root@laika:~# atq
6      Thu Aug 2 16:00:00 2007 a root
5      Thu Aug 2 10:05:00 2007 a root
root@laika:~#
```

atrm

Jobs in the at queue can be removed with **atrm**.

```
root@laika:~# atq
6          Thu Aug  2 16:00:00 2007 a root
5          Thu Aug  2 10:05:00 2007 a root
root@laika:~# atrm 5
root@laika:~# atq
6          Thu Aug  2 16:00:00 2007 a root
root@laika:~#
```

at.allow and at.deny

You can also use the **/etc/at.allow** and **/etc/at.deny** files to manage who can schedule jobs with **at**.

The **/etc/at.allow** file can contain a list of users that are allowed to schedule **at** jobs. When **/etc/at.allow** does not exist, then everyone can use **at** unless their username is listed in **/etc/at.deny**.

If none of these files exist, then everyone can use **at**.

42.2. cron

crontab file

The **crontab(1)** command can be used to maintain the **crontab(5)** file. Each user can have their own crontab file to schedule jobs at a specific time. This time can be specified with five fields in this order: minute, hour, day of the month, month and day of the week. If a field contains an asterisk (*), then this means all values of that field.

The following example means : run script42 eight minutes after two, every day of the month, every month and every day of the week.

```
8 14 * * * script42
```

Run script8472 every month on the first of the month at 25 past midnight.

```
25 0 1 * * script8472
```

Run this script33 every two minutes on Sunday (both 0 and 7 refer to Sunday).

```
*/2 * * * 0
```

Instead of these five fields, you can also type one of these: @reboot, @yearly or @annually, @monthly, @weekly, @daily or @midnight, and @hourly.

crontab command

Users should not edit the crontab file directly, instead they should type **crontab -e** which will use the editor defined in the EDITOR or VISUAL environment variable. Users can display their cron table with **crontab -l**.

cron.allow and cron.deny

The **cron daemon crond** is reading the cron tables, taking into account the **/etc/cron.allow** and **/etc/cron.deny** files.

These files work in the same way as **at.allow** and **at.deny**. When the **cron.allow** file exists, then your username has to be in it, otherwise you cannot use **cron**. When the **cron.allow** file does not exist, then your username cannot be in the **cron.deny** file if you want to use **cron**.

/etc/crontab

The **/etc/crontab** file contains entries for when to run hourly/daily/weekly/monthly tasks. It will look similar to this output.


```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

20 3 * * *      root    run-parts --report /etc/cron.daily
40 3 * * 7      root    run-parts --report /etc/cron.weekly
55 3 1 * *      root    run-parts --report /etc/cron.monthly
```

/etc/cron.*

The directories shown in the next screenshot contain the tasks that are run at the times scheduled in **/etc/crontab**. The **/etc/cron.d** directory is for special cases, to schedule jobs that require finer control than hourly/daily/weekly/monthly.

```
paul@laika:~$ ls -ld /etc/cron.*
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.d
drwxr-xr-x 2 root root 4096 2008-04-19 15:04 /etc/cron.daily
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.hourly
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.monthly
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.weekly
```

42.3. practice : scheduling

1. Schedule two jobs with **at**, display the **at queue** and remove a job.
2. As normal user, use **crontab -e** to schedule a script to run every four minutes.
3. As root, display the **crontab** file of your normal user.
4. As the normal user again, remove your **crontab** file.
5. Take a look at the **cron** files and directories in **/etc** and understand them. What is the **run-parts** command doing ?

42.4. solution : scheduling

1. Schedule two jobs with **at**, display the **at queue** and remove a job.

```
root@rhel55 ~# at 9pm today
at> echo go to bed >> /root/todo.txt
at> <EOT>
job 1 at 2010-11-14 21:00
root@rhel55 ~# at 17h31 today
at> echo go to lunch >> /root/todo.txt
at> <EOT>
job 2 at 2010-11-14 17:31
root@rhel55 ~# atq
2 2010-11-14 17:31 a root
1 2010-11-14 21:00 a root
root@rhel55 ~# atrm 1
root@rhel55 ~# atq
2 2010-11-14 17:31 a root
root@rhel55 ~# date
Sun Nov 14 17:31:01 CET 2010
root@rhel55 ~# cat /root/todo.txt
go to lunch
```

2. As normal user, use **crontab -e** to schedule a script to run every four minutes.

```
paul@rhel55 ~$ crontab -e
no crontab for paul - using an empty one
crontab: installing new crontab
```

3. As root, display the **crontab** file of your normal user.

```
root@rhel55 ~# crontab -l -u paul
*/4 * * * * echo `date` >> /home/paul/crontest.txt
```

4. As the normal user again, remove your **crontab** file.

```
paul@rhel55 ~$ crontab -r
paul@rhel55 ~$ crontab -l
no crontab for paul
```

5. Take a look at the **cron** files and directories in **/etc** and understand them. What is the **run-parts** command doing ?

```
run-parts runs a script in a directory
```

Chapter 43. logging

Table of Contents

43.1. login logging	391
43.2. syslogd	394
43.3. logger	396
43.4. watching logs	396
43.5. rotating logs	397
43.6. practice : logging	398
43.7. solution : logging	399

This chapter has three distinct subjects.

First we look at login logging ; how can we find out who is logging in to the system, when and from where. And who is not logging in, who fails at **su** or **ssh**.

Second we discuss how to configure the syslog daemon, and how to test it with **logger**.

The last part is mostly about **rotating logs** and mentions the **tail -f** and **watch** commands for **watching logs**.

43.1. login logging

To keep track of who is logging into the system, Linux can maintain the `/var/log/wtmp`, `/var/log/btmp`, `/var/run/utmp` and `/var/log/lastlog` files.

`/var/run/utmp (who)`

Use the **who** command to see the `/var/run/utmp` file. This command is showing you all the **currently** logged in users. Notice that the `utmp` file is in `/var/run` and not in `/var/log`.

```
[root@rhel4 ~]# who
paul pts/1 Feb 14 18:21 (192.168.1.45)
sandra pts/2 Feb 14 18:11 (192.168.1.42)
inge pts/3 Feb 14 12:01 (192.168.1.33)
els pts/4 Feb 14 14:33 (192.168.1.19)
```

`/var/log/wtmp (last)`

The `/var/log/wtmp` file is updated by the **login program**. Use **last** to see the `/var/run/wtmp` file.

```
[root@rhel4a ~]# last | head
paul pts/1 192.168.1.45 Wed Feb 14 18:39 still logged in
reboot system boot 2.6.9-42.0.8.ELs Wed Feb 14 18:21 (01:15)
nicolas pts/5 pc-dss.telematic Wed Feb 14 12:32 - 13:06 (00:33)
stefaan pts/3 pc-sde.telematic Wed Feb 14 12:28 - 12:40 (00:12)
nicolas pts/3 pc-nae.telematic Wed Feb 14 11:36 - 12:21 (00:45)
nicolas pts/3 pc-nae.telematic Wed Feb 14 11:34 - 11:36 (00:01)
dirk pts/5 pc-dss.telematic Wed Feb 14 10:03 - 12:31 (02:28)
nicolas pts/3 pc-nae.telematic Wed Feb 14 09:45 - 11:34 (01:48)
dimitri pts/5 rhel4 Wed Feb 14 07:57 - 08:38 (00:40)
stefaan pts/4 pc-sde.telematic Wed Feb 14 07:16 - down (05:50)
[root@rhel4a ~]#
```

The `last` command can also be used to get a list of last reboots.

```
[paul@rekkie ~]$ last reboot
reboot system boot 2.6.16-rekkie Mon Jul 30 05:13 (370+08:42)

wtmp begins Tue May 30 23:11:45 2006
[paul@rekkie ~]$
```

`/var/log/lastlog (lastlog)`

Use **lastlog** to see the `/var/log/lastlog` file.

```
[root@rhel4a ~]# lastlog | tail
tim           pts/5  10.170.1.122    Tue Feb 13 09:36:54 +0100 2007
rm            pts/6  rhel4           Tue Feb 13 10:06:56 +0100 2007
henk                                     **Never logged in**
stefaan       pts/3  pc-sde.telematic Wed Feb 14 12:28:38 +0100 2007
dirk          pts/5  pc-dss.telematic Wed Feb 14 10:03:11 +0100 2007
arsene                                     **Never logged in**
nicolas       pts/5  pc-dss.telematic Wed Feb 14 12:32:18 +0100 2007
dimitri       pts/5  rhel4           Wed Feb 14 07:57:19 +0100 2007
bashuserm     pts/7  rhel4           Tue Feb 13 10:35:40 +0100 2007
kornuserm     pts/5  rhel4           Tue Feb 13 10:06:17 +0100 2007
[root@rhel4a ~]#
```

/var/log/btmp (lastb)

There is also the **lastb** command to display the **/var/log/btmp** file. This file is updated by the login program when entering the wrong password, so it contains failed login attempts. Many computers will not have this file, resulting in no logging of failed login attempts.

```
[root@RHEL4b ~]# lastb
lastb: /var/log/btmp: No such file or directory
Perhaps this file was removed by the operator to prevent logging lastb\
info.
[root@RHEL4b ~]#
```

The reason given for this is that users sometimes type their password by mistake instead of their login, so this world readable file poses a security risk. You can enable bad login logging by simply creating the file. Doing a `chmod o-r /var/log/btmp` improves security.

```
[root@RHEL4b ~]# touch /var/log/btmp
[root@RHEL4b ~]# ll /var/log/btmp
-rw-r--r-- 1 root root 0 Jul 30 06:12 /var/log/btmp
[root@RHEL4b ~]# chmod o-r /var/log/btmp
[root@RHEL4b ~]# lastb

btmp begins Mon Jul 30 06:12:19 2007
[root@RHEL4b ~]#
```

Failed logins via ssh, rlogin or su are not registered in **/var/log/btmp**. Failed logins via tty are.

```
[root@RHEL4b ~]# lastb
HalvarFl tty3           Mon Jul 30 07:10 - 07:10 (00:00)
Maria    tty1           Mon Jul 30 07:09 - 07:09 (00:00)
Roberto  tty1           Mon Jul 30 07:09 - 07:09 (00:00)

btmp begins Mon Jul 30 07:09:32 2007
[root@RHEL4b ~]#
```

su and ssh logins

Depending on the distribution, you may also have the `/var/log/secure` file being filled with messages from the `auth` and/or `authpriv` syslog facilities. This log will include `su` and/or `ssh` failed login attempts. Some distributions put this in `/var/log/auth.log`, verify the syslog configuration.

```
[root@RHEL4b ~]# cat /var/log/secure
Jul 30 07:09:03 sshd[4387]: Accepted publickey for paul from ::ffff:19\
2.168.1.52 port 33188 ssh2
Jul 30 05:09:03 sshd[4388]: Accepted publickey for paul from ::ffff:19\
2.168.1.52 port 33188 ssh2
Jul 30 07:22:27 sshd[4655]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 05:22:27 sshd[4656]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 07:22:30 sshd[4655]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 05:22:30 sshd[4656]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 07:22:33 sshd[4655]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 05:22:33 sshd[4656]: Failed password for Hermione from ::ffff:1\
92.168.1.52 port 38752 ssh2
Jul 30 08:27:33 sshd[5018]: Invalid user roberto from ::ffff:192.168.1\
.52
Jul 30 06:27:33 sshd[5019]: input_userauth_request: invalid user rober\
to
Jul 30 06:27:33 sshd[5019]: Failed none for invalid user roberto from \
::ffff:192.168.1.52 port 41064 ssh2
Jul 30 06:27:33 sshd[5019]: Failed publickey for invalid user roberto \
from ::ffff:192.168.1.52 port 41064 ssh2
Jul 30 08:27:36 sshd[5018]: Failed password for invalid user roberto f\
rom ::ffff:192.168.1.52 port 41064 ssh2
Jul 30 06:27:36 sshd[5019]: Failed password for invalid user roberto f\
rom ::ffff:192.168.1.52 port 41064 ssh2
[root@RHEL4b ~]#
```

You can enable this yourself, with a custom log file by adding the following line to `syslog.conf`.

```
auth.* ,authpriv.* /var/log/customsec.log
```

43.2. syslogd

about syslog

The standard method of logging on Linux is through the **syslogd** daemon. Syslog was developed by **Eric Allman** for sendmail, but quickly became a standard among many Unix applications and was much later written as rfc 3164. The syslog daemon can receive messages on udp **port 514** from many applications (and appliances), and can append to log files, print, display messages on terminals and forward logs to other syslogd daemons on other machines. The syslogd daemon is configured in **/etc/syslog.conf**.

Each line in the configuration file uses a **facility** to determine where the message is coming from. It also contains a **level** for the severity of the message, and an **action** to decide on what to do with the message.

facilities

The **man syslog.conf** will explain the different default facilities for certain daemons, such as mail, lpr, news and kern(el) messages. The local0 to local7 facility can be used for appliances (or any networked device that supports syslog). Here is a list of all facilities for syslog.conf version 1.3. The security keyword is deprecated.

```
auth (security)
authpriv
cron
daemon
ftp
kern
lpr mail
mark (internal use only)
news
syslog
user
uucp
local0-7
```

levels

The worst severity a message can have is **emerg** followed by **alert** and **crit**. Lowest priority should go to **info** and **debug** messages. Specifying a severity will also log all messages with a higher severity. You can prefix the severity with = to obtain only messages that match that severity. You can also specify **.none** to prevent a specific action from any message from a certain facility.

Here is a list of all levels, in ascending order. The keywords warn, error and panic are deprecated.


```
debug
info
notice
warning (warn)
err (error)
crit
alert
emerg (panic)
```

actions

The default action is to send a message to the username listed as action. When the action is prefixed with a / then syslog will send the message to the file (which can be a regular file, but also a printer or terminal). The @ sign prefix will send the message on to another syslog server. Here is a list of all possible actions.

```
root,user1      list of users, separated by comma's
*              message to all logged on users
/              file (can be a printer, a console, a tty, ...)
-/            file, but don't sync after every write
|             named pipe
@             other syslog hostname
```

In addition, you can prefix actions with a - to omit syncing the file after every logging.

configuration

Below a sample configuration of custom local4 messages in **/etc/syslog.conf**.

```
local4.crit          /var/log/critandabove
local4.=crit         /var/log/onlycrit
local4.*             /var/log/alllocal4
```

Don't forget to restart the server.

```
[root@rhel4a ~]# /etc/init.d/syslog restart
Shutting down kernel logger:          [ OK ]
Shutting down system logger:         [ OK ]
Starting system logger:               [ OK ]
Starting kernel logger:               [ OK ]
[root@rhel4a ~]#
```

43.3. logger

The `logger` command can be used to generate syslog test messages. You can also use it in scripts. An example of testing `syslogd` with the **logger** tool.

```
[root@rhel4a ~]# logger -p local4.debug "l4 debug"
[root@rhel4a ~]# logger -p local4.crit "l4 crit"
[root@rhel4a ~]# logger -p local4.emerg "l4 emerg"
[root@rhel4a ~]#
```

The results of the tests with `logger`.

```
[root@rhel4a ~]# cat /var/log/critandabove
Feb 14 19:55:19 rhel4a paul: l4 crit
Feb 14 19:55:28 rhel4a paul: l4 emerg
[root@rhel4a ~]# cat /var/log/onlycrit
Feb 14 19:55:19 rhel4a paul: l4 crit
[root@rhel4a ~]# cat /var/log/alllocal4
Feb 14 19:55:11 rhel4a paul: l4 debug
Feb 14 19:55:19 rhel4a paul: l4 crit
Feb 14 19:55:28 rhel4a paul: l4 emerg
[root@rhel4a ~]#
```

43.4. watching logs

You might want to use the **tail -f** command to look at the last lines of a log file. The **-f** option will dynamically display lines that are appended to the log.

```
paul@ubu1010:~$ tail -f /var/log/udev
SEQNUM=1741
SOUND_INITIALIZED=1
ID_VENDOR_FROM_DATABASE=nVidia Corporation
ID_MODEL_FROM_DATABASE=MCP79 High Definition Audio
ID_BUS=pci
ID_VENDOR_ID=0x10de
ID_MODEL_ID=0x0ac0
ID_PATH=pci-0000:00:08.0
SOUND_FORM_FACTOR=internal
```

You can automatically repeat commands by preceding them with the **watch** command. When executing the following:

```
[root@rhel6 ~]# watch who
```

Something similar to this, repeating the output of the **who** command every two seconds, will appear on the screen.

```
Every 2.0s: who                               Sun Jul 17 15:31:03 2011

root      tty1          2011-07-17 13:28
paul      pts/0         2011-07-17 13:31 (192.168.1.30)
paul      pts/1         2011-07-17 15:19 (192.168.1.30)
```

43.5. rotating logs

A lot of log files are always growing in size. To keep this within bounds, you might want to use **logrotate** to rotate, compress, remove and mail log files. More info on the logrotate command in **/etc/logrotate.conf.** Individual configurations can be found in the **/etc/logrotate.d/** directory.

In this screenshot the configuration file for the logfiles from **aptitude** to configure monthly rotates, keeping the last six and compressing old logs.

```
paul@ubu1010:/var/log$ cat /etc/logrotate.d/aptitude
/var/log/aptitude {
    rotate 6
    monthly
    compress
    missingok
    notifempty
}
```

And this screenshot is the result of the above configuration, for the logfile from **aptitude.**

```
paul@ubu1010:/var/log$ ls -l /var/log/aptitude*
-rw-r--r-- 1 root root 18298 2011-07-17 13:32 /var/log/aptitude
-rw-r--r-- 1 root root  8163 2011-07-01 01:43 /var/log/aptitude.1.gz
-rw-r--r-- 1 root root  8163 2011-06-01 01:43 /var/log/aptitude.2.gz
-rw-r--r-- 1 root root  8163 2011-05-01 01:43 /var/log/aptitude.3.gz
```

43.6. practice : logging

1. Display the `/var/run/utmp` file with the proper command (not with `cat` or `vi`).
2. Display the `/var/log/wtmp` file.
3. Use the `lastlog` and `lastb` commands, understand the difference.
4. Examine `syslog` to find the location of the log file containing `ssh` failed logins.
5. Configure `syslog` to put `local4.error` and above messages in `/var/log/l4e.log` and `local4.info` only `.info` in `/var/log/l4i.log`. Test that it works with the `logger` tool!
6. Configure `/var/log/Mysu.log`, all the `su` to root messages should go in that log. Test that it works!
7. Send the `local5` messages to the `syslog` server of your neighbour. Test that it works.
8. Write a script that executes `logger` to `local4` every 15 seconds (different message). Use `tail -f` and watch on your `local4` log files.

43.7. solution : logging

1. Display the `/var/run/utmp` file.

```
who
```

2. Display the `/var/log/wtmp` file.

```
last
```

3. Use the `lastlog` and `lastb` commands, understand the difference.

```
lastlog : when users last logged on
```

```
lastb: failed (bad) login attempts
```

4. Examine `syslog` to find the location of the log file containing `ssh` failed logins.

```
root@rhel53 ~# grep authpriv /etc/syslog.conf
authpriv.*      /var/log/secure
```

```
Debian/Ubuntu: /var/log/auth.log
```

Ubuntu 9.10 and Debian Lenny have switched to using **rsyslog**.

```
root@ubuntu910:~# grep authpriv /etc/rsyslog.d/50-default.conf
auth,authpriv.*  /var/log/auth.log
```

```
root@deb503:~# grep authpriv /etc/rsyslog.conf
auth,authpriv.*  /var/log/auth.log
```

5. Configure `syslog` to put `local4.error` and above messages in `/var/log/l4e.log` and `local4.info` only `.info` in `/var/log/l4i.log`. Test that it works with the `logger` tool!

```
echo local4.error /var/log/l4e.log >> /etc/syslog.conf
```

```
echo local4.=info /var/log/l4i.log >> /etc/syslog.conf
```

```
/etc/init.d/syslog restart
```

```
logger -p local4.error "l4 error test"
```

```
logger -p local4.alert "l4 alert test"
```

```
logger -p local4.info "l4 info test"
```

```
cat /var/log/l4e.log
```

```
cat /var/log/l4i.log
```

6. Configure `/var/log/Mysu.log`, all the `su` to root messages should go in that log. Test that it works!

```
echo authpriv.* /var/log/Mysu.log >> /etc/syslog.conf
```

This will log more than just the **su** usage.

7. Send the local5 messages to the syslog server of your neighbour. Test that it works.

On RHEL5, edit `/etc/sysconfig/syslog` to enable remote listening on the server.

On Debian/Ubuntu edit `/etc/default/syslog` or `/etc/default/rsyslog`.

on the client: `logger -p local5.info "test local5 to neighbour"`

8. Write a script that executes logger to local4 every 15 seconds (different message). Use `tail -f` and watch on your local4 log files.

```
root@rhel53 scripts# cat logloop
#!/bin/bash

for i in `seq 1 10`
do
logger -p local4.info "local4.info test number $i"
sleep 15
done

root@rhel53 scripts# chmod +x logloop
root@rhel53 scripts# ./logloop &
[1] 8264
root@rhel53 scripts# tail -f /var/log/local4.all.log
Mar 28 13:13:36 rhel53 root: local4.info test number 1
Mar 28 13:13:51 rhel53 root: local4.info test number 2
...
```

Chapter 44. memory management

Table of Contents

44.1. displaying memory and cache	402
44.2. managing swap space	403
44.3. monitoring memory with vmstat	405
44.4. practice : memory	406
44.5. solution : memory	407

This chapter will tell you how to manage RAM memory and cache.

We start with some simple tools to display information about memory: **free -om**, **top** and **cat /proc/meminfo**.

We continue with managing swap space, using terms like **swapping**, **paging** and **virtual memory**.

The last part is about using **vmstat** to monitor swap usage.

44.1. displaying memory and cache

/proc/meminfo

Displaying `/proc/meminfo` will tell you a lot about the memory on your Linux computer.

```
paul@ubul010:~$ cat /proc/meminfo
MemTotal:      3830176 kB
MemFree:       244060 kB
Buffers:       41020 kB
Cached:        2035292 kB
SwapCached:    9892 kB
...
```

The first line contains the total amount of physical RAM, the second line is the unused RAM. **Buffers** is RAM used for buffering files, **cached** is the amount of RAM used as cache and **SwapCached** is the amount of swap used as cache. The file gives us much more information outside of the scope of this course.

free

The `free` tool can display the information provided by `/proc/meminfo` in a more readable format. The example below displays brief memory information in megabytes.

```
paul@ubul010:~$ free -om
              total        used         free       shared    buffers     cached
Mem:           3740          3519           221           0           42          1994
Swap:          6234             82          6152
```

top

The `top` tool is often used to look at processes consuming most of the cpu, but it also displays memory information on line four and five (which can be toggled by pressing `m`).

Below a screenshot of `top` on the same `ubu1010` from above.

```
top - 10:44:34 up 16 days, 9:56, 6 users, load average: 0.13, 0.09, 0.12
Tasks: 166 total,  1 running, 165 sleeping,  0 stopped,  0 zombie
Cpu(s):  5.1%us,  4.6%sy,  0.6%ni, 88.7%id,  0.8%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:    3830176k total,  3613720k used,  216456k free,   45452k buffers
Swap:   6384636k total,   84988k used,  6299648k free,  2050948k cached
```


44.2. managing swap space

about swap space

When the operating system needs more memory than physically present in RAM, it can use **swap space**. Swap space is located on slower but cheaper memory. Notice that, although hard disks are commonly used for swap space, their access times are one hundred thousand times slower.

The swap space can be a file, a partition, or a combination of files and partitions. You can see the swap space with the **free** command, or with **cat /proc/swaps**.

```
paul@ubul010:~$ free -o | grep -v Mem
              total          used          free      shared    buffers     cached
Swap:        6384636       84988       6299648
paul@ubul010:~$ cat /proc/swaps
Filename                Type              Size      Used   Priority
/dev/sda3                partition         6384636   84988   -1
```

The amount of swap space that you need depends heavily on the services that the computer provides.

creating a swap partition

You can activate or deactivate swap space with the **swapon** and **swapoff** commands. New swap space can be created with the **mkswap** command. The screenshot below shows the creation and activation of a swap partition.

```
root@RHELv4u4:~# fdisk -l 2> /dev/null | grep hda
Disk /dev/hda: 536 MB, 536870912 bytes
/dev/hda1                1            1040         524128+   83   Linux
root@RHELv4u4:~# mkswap /dev/hda1
Setting up swapspace version 1, size = 536702 kB
root@RHELv4u4:~# swapon /dev/hda1
```

Now you can see that **/proc/swaps** displays all swap spaces separately, whereas the **free -om** command only makes a human readable summary.

```
root@RHELv4u4:~# cat /proc/swaps
Filename                Type              Size      Used   Priority
/dev/mapper/VolGroup00-LogVol01  partition         1048568    0      -1
/dev/hda1                partition         524120     0      -2
root@RHELv4u4:~# free -om
              total          used          free      shared    buffers     cached
Mem:           249            245            4            0           125          54
Swap:          1535             0           1535
```

creating a swap file

Here is one more example showing you how to create a **swap file**. On Solaris you can use **mkfile** instead of **dd**.

```
root@RHELv4u4:~# dd if=/dev/zero of=/smallswapfile bs=1024 count=4096
4096+0 records in
4096+0 records out
root@RHELv4u4:~# mkswap /smallswapfile
Setting up swapspace version 1, size = 4190 kB
root@RHELv4u4:~# swapon /smallswapfile
root@RHELv4u4:~# cat /proc/swaps
Filename                                Type              Size              Used              Priority
/dev/mapper/VolGroup00-LogVol01         partition         1048568           0                 -1
/dev/hda1                                partition         524120            0                 -2
/smallswapfile                          file              4088              0                 -3
```

swap space in /etc/fstab

If you like these swaps to be permanent, then don't forget to add them to **/etc/fstab**. The lines in /etc/fstab will be similar to the following.

```
/dev/hda1          swap          swap          defaults      0 0
/smallswapfile    swap          swap          defaults      0 0
```

44.3. monitoring memory with vmstat

You can find information about **swap usage** using **vmstat**.

Below a simple **vmstat** displaying information in megabytes.

```
paul@ubul010:~$ vmstat -S m
procs -----memory----- --swap-- ----io---- -system- ----cpu----
 r b swpd free buff cache si so bi bo in cs us sy id wa
 0 0 87 225 46 2097 0 0 2 5 14 8 6 5 89 1
```

Below a sample **vmstat** when (in another terminal) root launches a **find /**. It generates a lot of disk i/o (bi and bo are disk blocks in and out). There is no need for swapping here.

```
paul@ubul010:~$ vmstat 2 100
procs -----memory----- --swap-- ----io---- -system-- ----cpu----
 r b swpd free buff cache si so bi bo in cs us sy id wa
 0 0 84984 1999436 53416 269536 0 0 2 5 2 10 6 5 89 1
 0 0 84984 1999428 53416 269564 0 0 0 0 1713 2748 4 4 92 0
 0 0 84984 1999552 53416 269564 0 0 0 0 1672 1838 4 6 90 0
 0 0 84984 1999552 53424 269560 0 0 0 14 1587 2526 5 7 87 2
 0 0 84984 1999180 53424 269580 0 0 0 100 1748 2193 4 6 91 0
 1 0 84984 1997800 54508 269760 0 0 610 0 1836 3890 17 10 68 4
 1 0 84984 1994620 55040 269748 0 0 250 168 1724 4365 19 17 56 9
 0 1 84984 1978508 55292 269704 0 0 126 0 1957 2897 19 18 58 4
 0 0 84984 1974608 58964 269784 0 0 1826 478 2605 4355 7 7 44 41
 0 2 84984 1971260 62268 269728 0 0 1634 756 2257 3865 7 7 47 39
```

Below a sample **vmstat** when executing (on RHEL6) a simple memory leaking program. Now you see a lot of memory being swapped (si is 'swapped in').

```
[paul@rhel6c ~]$ vmstat 2 100

procs -----memory----- --swap-- ----io---- -system-- ----cpu----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 0 3 245208 5280 232 1916 261 0 42 27 21 0 1 98 1 0
 0 2 263372 4800 72 908 143840 128 0 1138 462 191 2 10 0 88 0
 1 3 350672 4792 56 992 169280 256 0 1092 360 142 1 13 0 86 0
 1 4 449584 4788 56 1024 95880 64 0 606 471 191 2 13 0 85 0
 0 4 471968 4828 56 1140 44832 80 0 390 235 90 2 12 0 87 0
 3 5 505960 4764 56 1136 68008 16 0 538 286 109 1 12 0 87 0
```

The code below was used to simulate a memory leak (and force swapping). This code was found on wikipedia without author.

```
paul@mac:~$ cat memleak.c
#include <stdlib.h>

int main(void)
{
    while (malloc(50));
    return 0;
}
```

44.4. practice : memory

1. Use **dmesg** to find the total amount of memory in your computer.
2. Use **free** to display memory usage in kilobytes (then in megabytes).
3. On a virtual machine, create a swap partition (you might need an extra virtual disk for this).
4. Add a 20 megabyte swap file to the system.
5. Put all swap spaces in **/etc/fstab** and activate them. Test with a reboot that they are mounted.
6. Use **free** to verify usage of current swap.
7. (optional) Display the usage of swap with **vmstat** and **free -s** during a memory leak.

44.5. solution : memory

1. Use **dmesg** to find the total amount of memory in your computer.

```
dmesg | grep Memory
```

2. Use **free** to display memory usage in kilobytes (then in megabytes).

```
free ; free -m
```

3. On a virtual machine, create a swap partition (you might need an extra virtual disk for this).

```
mkswap /dev/sdd1 ; swapon /dev/sdd1
```

4. Add a 20 megabyte swap file to the system.

```
dd if=/dev/zero of=/swapfile20mb bs=1024 count=20000
mkswap /swapfile20mb
swapon /swapfile20mb
```

5. Put all swap spaces in **/etc/fstab** and activate them. Test with a reboot that they are mounted.

```
root@computer# tail -2 /etc/fstab
/dev/sdd1      swap swap defaults 0 0
/swapfile20mb swap swap defaults 0 0
```

6. Use **free** to verify usage of current swap.

```
free -om
```

7. (optional) Display the usage of swap with **vmstat** and **free -s** during a memory leak.

Chapter 45. package management

Table of Contents

45.1. terminology	409
45.2. deb	411
45.3. apt-get	412
45.4. aptitude	415
45.5. apt	416
45.6. rpm	417
45.7. yum	420
45.8. alien	425
45.9. downloading software outside the repository	426
45.10. compiling software	426
45.11. practice: package management	427
45.12. solution: package management	428

Most Linux distributions have a **package management** system with online **repositories** containing thousands of packages. This makes it very easy to install and remove applications, operating system components, documentation and much more.

We discuss the two most used package formats **.rpm** and **.deb** and their respective tools. We also briefly discuss the option of obtaining software from outside the **repository**.

45.1. terminology

repository

A lot of software and documentation for your Linux distribution is available as **packages** in one or more centrally distributed **repositories**. These **packages** in such a **repository** are tested and very easy to install (or remove) with a graphical or command line installer.

.deb packages

Debian, Ubuntu, Mint and all derivatives from Debian and Ubuntu use **.deb** packages. To manage software on these systems, you can use **aptitude** or **apt-get**, both these tools are a front end for **dpkg**.

.rpm packages

Red Hat, Fedora, CentOS, OpenSUSE, Mandriva, Red Flag and others use **.rpm** packages. The tools to manage software packages on these systems are **yum** and **rpm**.

dependency

Some packages need other packages to function. Tools like **apt-get**, **aptitude** and **yum** will install all **dependencies** you need. When using **dpkg** or **rpm**, or when building from **source**, you will need to install dependencies yourself.

open source

These repositories contain a lot of independent **open source software**. Often the source code is customized to integrate better with your distribution. Most distributions also offer this modified source code as a **package** in one or more **source repositories**.

You are free to go to the project website itself (samba.org, apache.org, github.com, ...) and download the **vanilla** (= without the custom distribution changes) source code.

GUI software management

End users have several graphical applications available via the desktop (look for 'add/remove software' or something similar).

Below a screenshot of Ubuntu Software Center running on Ubuntu 12.04. Graphical tools are not discussed in this book.



45.2. deb

about deb

Most people use **aptitude** or **apt-get** to manage their Debian/Ubuntu family of Linux distributions. Both are a front end for **dpkg** and are themselves a back end for **synaptic** and other graphical tools.

dpkg -l

The low level tool to work with **.deb** packages is **dpkg**. Here you see how to obtain a list of all installed packages on a Debian server.

```
root@debian6:~# dpkg -l | wc -l
265
```

Compare this to the same list on a Ubuntu Desktop computer.

```
root@ubul204~# dpkg -l | wc -l
2527
```

dpkg -l \$package

Here is an example on how to get information on an individual package. The **ii** at the beginning means the package is installed.

```
root@debian6:~# dpkg -l rsync | tail -1 | tr -s ' '
ii rsync 3.0.7-2 fast remote file copy program (like rcp)
```

dpkg

You could use **dpkg -i** to install a package and **dpkg -r** to remove a package, but you'd have to manually keep track of dependencies. Using **apt-get** or **aptitude** is much easier.

45.3. apt-get

Debian has been using **apt-get** to manage packages since 1998. Today Debian and many Debian-based distributions still actively support **apt-get**, though some experts claim **aptitude** is better at handling dependencies than **apt-get**.

Both commands use the same configuration files and can be used alternately; whenever you see **apt-get** in documentation, feel free to type **aptitude**.

We will start with **apt-get** and discuss **aptitude** in the next section.

apt-get update

When typing **apt-get update** you are downloading the names, versions and short description of all packages available on all configured repositories for your system.

In the example below you can see some repositories at the url **be.archive.ubuntu.com** because this computer was installed in Belgium. This url can be different for you.

```
root@ubul204~# apt-get update
Ign http://be.archive.ubuntu.com precise InRelease
Ign http://extras.ubuntu.com precise InRelease
Ign http://security.ubuntu.com precise-security InRelease
Ign http://archive.canonical.com precise InRelease
Ign http://be.archive.ubuntu.com precise-updates InRelease
...
Hit http://be.archive.ubuntu.com precise-backports/main Translation-en
Hit http://be.archive.ubuntu.com precise-backports/multiverse Translation-en
Hit http://be.archive.ubuntu.com precise-backports/restricted Translation-en
Hit http://be.archive.ubuntu.com precise-backports/universe Translation-en
Fetched 13.7 MB in 8s (1682 kB/s)
Reading package lists... Done
root@mac~#
```

Run **apt-get update** every time before performing other package operations.

apt-get upgrade

One of the nicest features of **apt-get** is that it allows for a secure update of **all software currently installed** on your computer with just **one** command.

```
root@debian6:~# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@debian6:~#
```

The above screenshot shows that all software is updated to the latest version available for my distribution.

apt-get clean

apt-get keeps a copy of downloaded packages in `/var/cache/apt/archives`, as can be seen in this screenshot.

```
root@ubul204~# ls /var/cache/apt/archives/ | head
accountsservice_0.6.15-2ubuntu9.4_i386.deb
appport_2.0.1-0ubuntu14_all.deb
appport-gtk_2.0.1-0ubuntu14_all.deb
apt_0.8.16~exp12ubuntu10.3_i386.deb
apt-transport-https_0.8.16~exp12ubuntu10.3_i386.deb
apt-utils_0.8.16~exp12ubuntu10.3_i386.deb
bind9-host_1%3a9.8.1.dfsg.P1-4ubuntu0.4_i386.deb
chromium-browser_20.0.1132.47~r144678-0ubuntu0.12.04.1_i386.deb
chromium-browser-l10n_20.0.1132.47~r144678-0ubuntu0.12.04.1_all.deb
chromium-codecs-ffmpeg_20.0.1132.47~r144678-0ubuntu0.12.04.1_i386.deb
```

Running **apt-get clean** removes all `.deb` files from that directory.

```
root@ubul204~# apt-get clean
root@ubul204~# ls /var/cache/apt/archives/*.deb
ls: cannot access /var/cache/apt/archives/*.deb: No such file or directory
```

apt-cache search

Use **apt-cache search** to search for availability of a package. Here we look for **rsync**.

```
root@ubul204~# apt-cache search rsync | grep ^rsync
rsync - fast, versatile, remote (and local) file-copying tool
rsyncrypto - rsync friendly encryption
```

apt-get install

You can install one or more applications by appending their name behind **apt-get install**. The screenshot shows how to install the **rsync** package.

```
root@ubul204~# apt-get install rsync
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  rsync
0 upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
Need to get 299 kB of archives.
After this operation, 634 kB of additional disk space will be used.
Get:1 http://be.archive.ubuntu.com/ubuntu/ precise/main rsync i386 3.0.9-1ubuntu1 [299 k
Fetched 299 kB in 0s (740 kB/s)
Selecting previously unselected package rsync.
(Reading database ... 323649 files and directories currently installed.)
Unpacking rsync (from .../rsync_3.0.9-1ubuntu1_i386.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Setting up rsync (3.0.9-1ubuntu1) ...
Removing any system startup links for /etc/init.d/rsync ...
root@ubul204~#
```

apt-get remove

You can remove one or more applications by appending their name behind **apt-get remove**. The screenshot shows how to remove the **rsync** package.

```
root@ubul204~# apt-get remove rsync
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  rsync ubuntu-standard
0 upgraded, 0 newly installed, 2 to remove and 8 not upgraded.
After this operation, 692 kB disk space will be freed.
Do you want to continue [Y/n]?
(Reading database ... 323681 files and directories currently installed.)
Removing ubuntu-standard ...
Removing rsync ...
 * Stopping rsync daemon rsync
Processing triggers for ureadahead ...
Processing triggers for man-db ...
root@ubul204~#
```

Note however that some configuration information is not removed.

```
root@ubul204~# dpkg -l rsync | tail -1 | tr -s ' '
rc rsync 3.0.9-1ubuntu1 fast, versatile, remote (and local) file-copying tool
```

apt-get purge

You can purge one or more applications by appending their name behind **apt-get purge**. Purging will also remove all existing configuration files related to that application. The screenshot shows how to purge the **rsync** package.

```
root@ubul204~# apt-get purge rsync
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  rsync*
0 upgraded, 0 newly installed, 1 to remove and 8 not upgraded.
After this operation, 0 B of additional disk space will be used.
Do you want to continue [Y/n]?
(Reading database ... 323651 files and directories currently installed.)
Removing rsync ...
Purging configuration files for rsync ...
Processing triggers for ureadahead ...
root@ubul204~#
```

Note that **dpkg** has no information about a purged package, except that it is uninstalled and no configuration is left on the system.

```
root@ubul204~# dpkg -l rsync | tail -1 | tr -s ' '
un rsync <none> (no description available)
```

45.4. aptitude

Most people use **aptitude** for package management on Debian, Mint and Ubuntu systems.

To synchronize with the repositories.

```
aptitude update
```

To patch and upgrade all software to the latest version on Debian.

```
aptitude upgrade
```

To patch and upgrade all software to the latest version on Ubuntu and Mint.

```
aptitude safe-upgrade
```

To install an application with all dependencies.

```
aptitude install $package
```

To search the repositories for applications that contain a certain string in their name or description.

```
aptitude search $string
```

To remove an application.

```
aptitude remove $package
```

To remove an application and all configuration files.

```
aptitude purge $package
```

45.5. apt

Both **apt-get** and **aptitude** use the same configuration information in **/etc/apt/**. Thus adding a repository for one of them, will automatically add it for both.

/etc/apt/sources.list

The resource list used by **apt-get** and **aptitude** is located in **/etc/apt/sources.list**. This file contains a list of http or ftp sources where packages for the distribution can be downloaded.

This is what that list looks like on my Debian server.

```
root@debian6:~# cat /etc/apt/sources.list
deb http://ftp.be.debian.org/debian/ squeeze main
deb-src http://ftp.be.debian.org/debian/ squeeze main

deb http://security.debian.org/ squeeze/updates main
deb-src http://security.debian.org/ squeeze/updates main

# squeeze-updates, previously known as 'volatile'
deb http://ftp.be.debian.org/debian/ squeeze-updates main
deb-src http://ftp.be.debian.org/debian/ squeeze-updates main
```

On my Ubuntu there are four times as many online repositories in use.

```
root@ubul204~# wc -l /etc/apt/sources.list
63 /etc/apt/sources.list
```

There is much more to learn about **apt**, explore commands like **add-apt-repository**, **apt-key** and **apropos apt**.

45.6. rpm

about rpm

The **Red Hat package manager** can be used on the command line with **rpm** or in a graphical way going to Applications--System Settings--Add/Remove Applications. Type **rpm --help** to see some of the options.

Software distributed in the **rpm** format will be named **foo-version.platform.rpm** .

rpm -qa

To obtain a list of all installed software, use the **rpm -qa** command.

```
[root@RHEL52 ~]# rpm -qa | grep samba
system-config-samba-1.2.39-1.e15
samba-3.0.28-1.e15_2.1
samba-client-3.0.28-1.e15_2.1
samba-common-3.0.28-1.e15_2.1
```

rpm -q

To verify whether one package is installed, use **rpm -q**.

```
root@RHELv4u4:~# rpm -q gcc
gcc-3.4.6-3
root@RHELv4u4:~# rpm -q laika
package laika is not installed
```

rpm -q --redhatprovides

To check whether a package is provided by Red Hat, use the **--redhatprovides** option.

```
root@RHELv4u4:~# rpm -q --redhatprovides bash
bash-3.0-19.3
root@RHELv4u4:~# rpm -q --redhatprovides gcc
gcc-3.4.6-3
root@RHELv4u4:~# rpm -q --redhatprovides laika
no package provides laika
```

rpm -Uvh

To install or upgrade a package, use the **-Uvh** switches. The **-U** switch is the same as **-i** for install, except that older versions of the software are removed. The **-vh** switches are for nicer output.

```
root@RHELv4u4:~# rpm -Uvh gcc-3.4.6-3
```

rpm -e

To remove a package, use the `-e` switch.

```
root@RHELv4u4:~# rpm -e gcc-3.4.6-3
```

rpm -e verifies dependencies, and thus will prevent you from accidentally erasing packages that are needed by other packages.

```
[root@RHEL52 ~]# rpm -e gcc-4.1.2-42.el5
error: Failed dependencies:
gcc = 4.1.2-42.el5 is needed by (installed) gcc-c++-4.1.2-42.el5.i386
gcc = 4.1.2-42.el5 is needed by (installed) gcc-gfortran-4.1.2-42.el5.i386
gcc is needed by (installed) systemtap-0.6.2-1.el5_2.2.i386
```

/var/lib/rpm

The **rpm** database is located at `/var/lib/rpm`. This database contains all meta information about packages that are installed (via rpm). It keeps track of all files, which enables complete removes of software.

rpm2cpio

We can use **rpm2cpio** to convert an **rpm** to a **cpio** archive.

```
[root@RHEL53 ~]# file kernel.src.rpm
kernel.src.rpm: RPM v3 src PowerPC kernel-2.6.18-92.1.13.el5
[root@RHEL53 ~]# rpm2cpio kernel.src.rpm > kernel.cpio
[root@RHEL53 ~]# file kernel.cpio
kernel.cpio: ASCII cpio archive (SVR4 with no CRC)
```

But why would you want to do this ?

Perhaps just to see of list of files in the **rpm** file.

```
[root@RHEL53 ~]# rpm2cpio kernel.src.rpm | cpio -t | head -5
COPYING.modules
Config.mk
Module.kabi_i686
Module.kabi_i686PAE
Module.kabi_i686xen
```

Or to extract one file from an **rpm** package.

```
[root@RHEL53 ~]# rpm2cpio kernel.src.rpm | cpio -iv Config.mk
Config.mk
246098 blocks
```


45.7. yum

about yum

The **Yellowdog Updater, Modified (yum)** is an easier command to work with **rpm** packages. It is installed by default on Fedora and Red Hat Enterprise Linux since version 5.2.

yum list

Issue **yum list available** to see a list of available packages. The **available** parameter is optional.

```
[root@rhel155 ~]# yum list | wc -l
2471
```

Issue **yum list \$package** to get all versions (in different repositories) of one package.

```
[root@rhel155 ~]# yum list samba
Loaded plugins: rhnplugin, security
Installed Packages
samba.i386                3.0.33-3.28.el5          installed
Available Packages
samba.i386                3.0.33-3.29.el5_5        rhel-i386-server-5
```

yum search

To search for a package containing a certain string in the description or name use **yum search \$string**.

```
[root@rhel155 ~]# yum search gcc44
Loaded plugins: rhnplugin, security
===== Matched: gcc44 =====
gcc44.i386 : Preview of GCC version 4.4
gcc44-c++.i386 : C++ support for GCC version 4.4
gcc44-gfortran.i386 : Fortran support for GCC 4.4 previe
```

yum provides

To search for a package containing a certain file (you might need for compiling things) use **yum provides \$filename**.

```
[root@rhel155 ~]# yum provides /usr/share/man/man1/gzip.1.gz
Loaded plugins: rhnplugin, security
Importing additional filelist information
gzip-1.3.5-9.el5.i386 : The GNU data compression program.
Repo                : rhel-i386-server-5
Matched from:
Filename           : /usr/share/man/man1/gzip.1.gz
...
```

yum install

To install an application, use **yum install \$package**. Naturally **yum** will install all the necessary dependencies.

```
[root@rhel55 ~]# yum install sudo
Loaded plugins: rhnplugin, security
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package sudo.i386 0:1.7.2p1-7.el5_5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch      Version      Repository      Size
=====
Installing:
sudo         i386      1.7.2p1-7.el5_5  rhel-i386-server-5  230 k

Transaction Summary
=====
Install      1 Package(s)
Upgrade     0 Package(s)

Total download size: 230 k
Is this ok [y/N]: y
Downloading Packages:
sudo-1.7.2p1-7.el5_5.i386.rpm           | 230 kB      00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing      : sudo                               1/1

Installed:
sudo.i386 0:1.7.2p1-7.el5_5

Complete!
```

You can add more than one parameter here.

```
yum install $package1 $package2 $package3
```

yum update

To bring all applications up to date, by downloading and installing them, issue **yum update**. All software that was installed via **yum** will be updated to the latest version that is available in the repository.

```
yum update
```

If you only want to update one package, use **yum update \$package**.

```
[root@rhel55 ~]# yum update sudo
Loaded plugins: rhnplugin, security
Skipping security plugin, no data
Setting up Update Process
```

```
Resolving Dependencies
Skipping security plugin, no data
--> Running transaction check
---> Package sudo.i386 0:1.7.2p1-7.el5_5 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch      Version           Repository          Size
=====
Updating:
sudo         i386      1.7.2p1-7.el5_5  rhel-i386-server-5 230 k

Transaction Summary
=====
Install      0 Package(s)
Upgrade     1 Package(s)

Total download size: 230 k
Is this ok [y/N]: y
Downloading Packages:
sudo-1.7.2p1-7.el5_5.i386.rpm | 230 kB    00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Updating      : sudo                      1/2
  Cleanup      : sudo                      2/2

Updated:
sudo.i386 0:1.7.2p1-7.el5_5

Complete!
```

yum software groups

Issue **yum grouplist** to see a list of all available software groups.

```
[root@rhel55 ~]# yum grouplist
Loaded plugins: rhnplugin, security
Setting up Group Process
Installed Groups:
  Administration Tools
  Authoring and Publishing
  DNS Name Server
  Development Libraries
  Development Tools
  Editors
  GNOME Desktop Environment
  GNOME Software Development
  Graphical Internet
  Graphics
  Legacy Network Server
  Legacy Software Development
  Legacy Software Support
  Mail Server
  Network Servers
  Office/Productivity
  Printing Support
  Server Configuration Tools
```

```
System Tools
Text-based Internet
Web Server
Windows File Server
X Software Development
X Window System
Available Groups:
  Engineering and Scientific
  FTP Server
  Games and Entertainment
  Java Development
  KDE (K Desktop Environment)
  KDE Software Development
  MySQL Database
  News Server
  OpenFabrics Enterprise Distribution
  PostgreSQL Database
  Sound and Video
Done
```

To install a set of applications, brought together via a group, use **yum groupinstall \$groupname**.

```
[root@rhel55 ~]# yum groupinstall 'Sound and video'
Loaded plugins: rhnplugin, security
Setting up Group Process
Package alsa-utils-1.0.17-1.el5.i386 already installed and latest version
Package sox-12.18.1-1.i386 already installed and latest version
Package 9:mkisofs-2.01-10.7.el5.i386 already installed and latest version
Package 9:cdrecord-2.01-10.7.el5.i386 already installed and latest version
Package cdrdao-1.2.1-2.i386 already installed and latest version
Resolving Dependencies
--> Running transaction check
---> Package cdda2wav.i386 9:2.01-10.7.el5 set to be updated
---> Package cdparanoia.i386 0:alpha9.8-27.2 set to be updated
---> Package sound-juicer.i386 0:2.16.0-3.el5 set to be updated
--> Processing Dependency: libmusicbrainz >= 2.1.0 for package: sound-juicer
--> Processing Dependency: libmusicbrainz.so.4 for package: sound-juicer
---> Package vorbis-tools.i386 1:1.1.1-3.el5 set to be updated
--> Processing Dependency: libao >= 0.8.4 for package: vorbis-tools
--> Processing Dependency: libao.so.2 for package: vorbis-tools
--> Running transaction check
---> Package libao.i386 0:0.8.6-7 set to be updated
---> Package libmusicbrainz.i386 0:2.1.1-4.1 set to be updated
--> Finished Dependency Resolution
...
```

Read the manual page of **yum** for more information about managing groups in **yum**.

/etc/yum.conf and repositories

The configuration of **yum** repositories is done in **/etc/yum/yum.conf** and **/etc/yum/repos.d/**.

Configuring **yum** itself is done in **/etc/yum.conf**. This file will contain the location of a log file and a cache directory for **yum** and can also contain a list of repositories.

Recently **yum** started accepting several **repo** files with each file containing a list of **repositories**. These **repo** files are located in the **/etc/yum.repos.d/** directory.

One important flag for yum is **enablerepo**. Use this command if you want to use a repository that is not enabled by default.

```
yum $command $foo --enablerepo=$repo
```

An example of the contents of the repo file: MyRepo.repo

```
[$repo]
name=My Repository
baseurl=http://path/to/MyRepo
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-MyRep
```

45.8. alien

alien is experimental software that converts between **rpm** and **deb** package formats (and others).

Below an example of how to use **alien** to convert an **rpm** package to a **deb** package.

```
paul@barry:~$ ls -l netcat*
-rw-r--r-- 1 paul paul 123912 2009-06-04 14:58 netcat-0.7.1-1.i386.rpm
paul@barry:~$ alien --to-deb netcat-0.7.1-1.i386.rpm
netcat_0.7.1-2_i386.deb generated
paul@barry:~$ ls -l netcat*
-rw-r--r-- 1 paul paul 123912 2009-06-04 14:58 netcat-0.7.1-1.i386.rpm
-rw-r--r-- 1 root root 125236 2009-06-04 14:59 netcat_0.7.1-2_i386.deb
```

*In real life, use the **netcat** tool provided by your distribution, or use the **.deb** file from their website.*

45.9. downloading software outside the repository

First and most important, whenever you download software, start by reading the README file!

Normally the readme will explain what to do after download. You will probably receive a `.tar.gz` or a `.tgz` file. Read the documentation, then put the compressed file in a directory. You can use the following to find out where the package wants to install.

```
tar tvzpf $downloadedFile.tgz
```

You unpack them like with **tar xzf**, it will create a directory called `applicationName-1.2.3`

```
tar xzf $applicationName.tgz
```

Replace the `z` with a `j` when the file ends in `.tar.bz2`. The **tar**, **gzip** and **bzip2** commands are explained in detail in the Linux Fundamentals course.

If you download a **.deb** file, then you'll have to use **dpkg** to install it, **.rpm**'s can be installed with the **rpm** command.

45.10. compiling software

First and most important, whenever you download source code for installation, start by reading the README file!

Usually the steps are always the same three : running **./configure** followed by **make** (which is the actual compiling) and then by **make install** to copy the files to their proper location.

```
./configure  
make  
make install
```


45.11. practice: package management

1. Find the Graphical application on all computers to add and remove applications.
2. Verify on both systems whether gcc is installed.
3. Use aptitude or yum to search for and install the 'dict', 'samba' and 'wesnoth' applications. Did you find all them all ?
4. Search the internet for 'webmin' and install it.
5. If time permits, uninstall Samba from the ubuntu machine, download the latest version from samba.org and install it.

45.12. solution: package management

1. Find the Graphical application on all computers to add and remove applications.

2. Verify on both systems whether gcc is installed.

```
dpkg -l | grep gcc
```

```
rpm -qa | grep gcc
```

3. Use aptitude or yum to search for and install the 'dict', 'samba' and 'wesnoth' applications. Did you find all them all ?

```
aptitude search wesnoth (Debian, Ubuntu and family)
```

```
yum search wesnoth (Red Hat and family)
```

4. Search the internet for 'webmin' and install it.

Google should point you to webmin.com.

There are several formats available there choose .rpm, .deb or .tgz .

5. If time permits, uninstall Samba from the ubuntu machine, download the latest version from samba.org and install it.

Part XIII. network management

Chapter 46. general networking

Table of Contents

46.1. network layers	431
46.2. unicast, multicast, broadcast, anycast	434
46.3. lan-wan-man	436
46.4. internet - intranet - extranet	437
46.5. tcp/ip	438

While this chapter is not directly about **Linux**, it does contain general networking concepts that will help you in troubleshooting networks on **Linux**.

46.1. network layers

seven OSI layers

When talking about protocol layers, people usually mention the seven layers of the **osi** protocol (Application, Presentation, Session, Transport, Network, Data Link and Physical). We will discuss layers 2 and 3 in depth, and focus less on the other layers. The reason is that these layers are important for understanding networks. You will hear administrators use words like "this is a layer 2 device" or "this is a layer 3 broadcast", and you should be able to understand what they are talking about.

four DoD layers

The **DoD** (or tcp/ip) model has only four layers, roughly mapping its **network access layer** to OSI layers 1 and 2 (Physical and Datalink), its **internet (IP) layer** to the OSI **network layer**, its **host-to-host** (tcp, udp) layer to OSI layer 4 (transport) and its **application layer** to OSI layers 5, 6 and 7.

Below an attempt to put OSI and DoD layers next to some protocols and devices.

OSI Model	DoD Model	protocols		devices/apps
layer 5, 6, 7	application	dns, dhcp, ntp, snmp, https, ftp, ssh, telnet, http, pop3... others		web server, mail server, browser, mail client...
layer 4	host-to-host	tcp	udp	gateway
layer 3	internet	ip, icmp, igmp		router, firewall layer 3 switch
layer 2	network access	arp (mac), rarp		bridge layer 2 switch
layer 1		ethernet, token ring		hub

short introduction to the physical layer

The physical layer, or **layer 1**, is all about voltage, electrical signals and mechanical connections. Some networks might still use **coax** cables, but most will have migrated to **utp** (cat 5 or better) with **rj45** connectors.

Devices like **repeaters** and **hubs** are part of this layer. You cannot use software to 'see' a **repeater** or **hub** on the network. The only thing these devices are doing is amplifying electrical signals on cables. **Passive hubs** are multiport amplifiers that amplify an incoming electrical signal on all other connections. **Active hubs** do this by reading and retransmitting bits, without interpreting any meaning in those bits.

Network technologies like **csma/cd** and **token ring** are defined on this layer.

This is all we have to say about **layer 1** in this book.

short introduction to the data link layer

The data link layer, or **layer 2** is about frames. A frame has a **crc** (cyclic redundancy check). In the case of ethernet (802.3), each network card is identifiable by a unique 48-bit **mac** address (media access control address).

On this layer we find devices like bridges and switches. A bridge is more intelligent than a hub because a **bridge** can make decisions based on the mac address of computers. A **switch** also understands mac addresses.

In this book we will discuss commands like **arp** and **ifconfig** to explore this layer.

short introduction to the network layer

Layer 3 is about ip packets. This layer gives every host a unique 32-bit ip address. But **ip** is not the only protocol on this layer, there is also **icmp**, **igmp**, **ipv6** and more. A complete list can be found in the **/etc/protocols** file.

On this layer we find devices like **routers** and layer 3 switches, devices that know (and have) an ip address.

In tcp/ip this layer is commonly referred to as the **internet layer**.

short introduction to the transport layer

We will discuss the **tcp** and **udp** protocols in the context of layer 4. The DoD model calls this the host-to-host layer.

layers 5, 6 and 7

The tcp/ip application layer includes layers 5, 6 and 7. Details on the difference between these layers are out of scope of this course.

network layers in this book

Stacking of layers in this book is based on the **Protocols in Frame** explanation in the **wireshark** sniffer. When sniffing a dhcp packet, we notice the following in the sniffer.

[Protocols in Frame: eth:ip:udp:bootp]

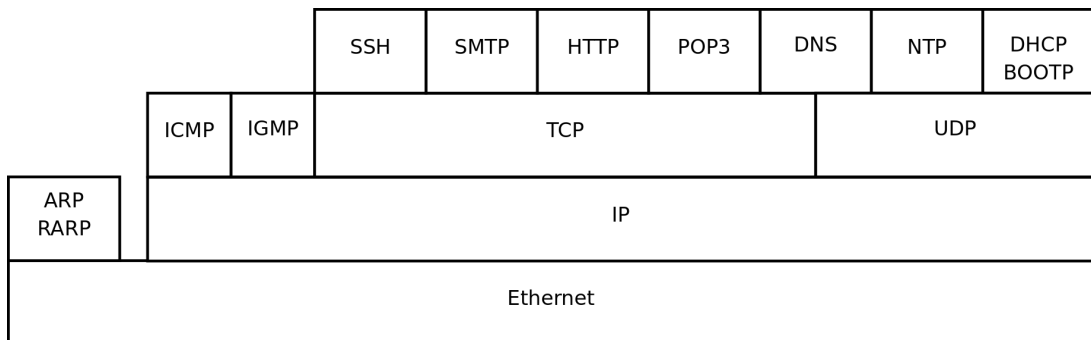
Sniffing for ntp (Network Time Protocol) packets gives us this line, which makes us conclude to put **ntp** next to **bootp** in the protocol chart below.

[Protocols in Frame: eth:ip:udp:ntp]

Sniffing an **arp** broadcast makes us put arp next to **ip**. All these protocols are explained later in this chapter.

[Protocols in Frame: eth:arp]

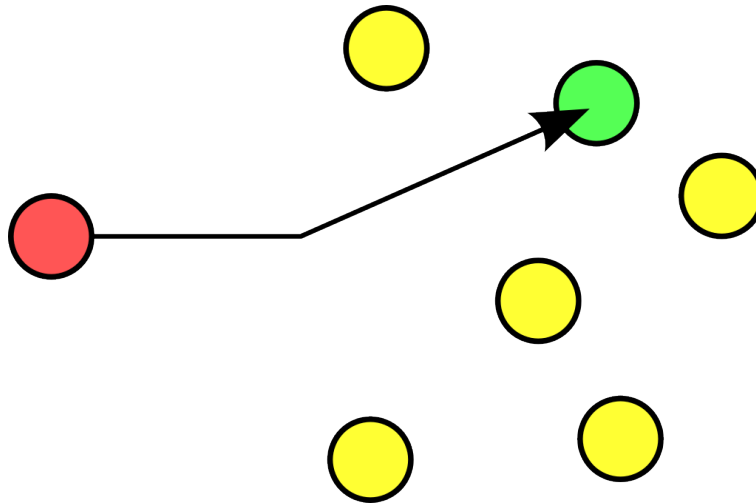
Below is a protocol chart based on wireshark's knowledge. It contains some very common protocols that are discussed in this book. The chart does not contain all protocols.



46.2. unicast, multicast, broadcast, anycast

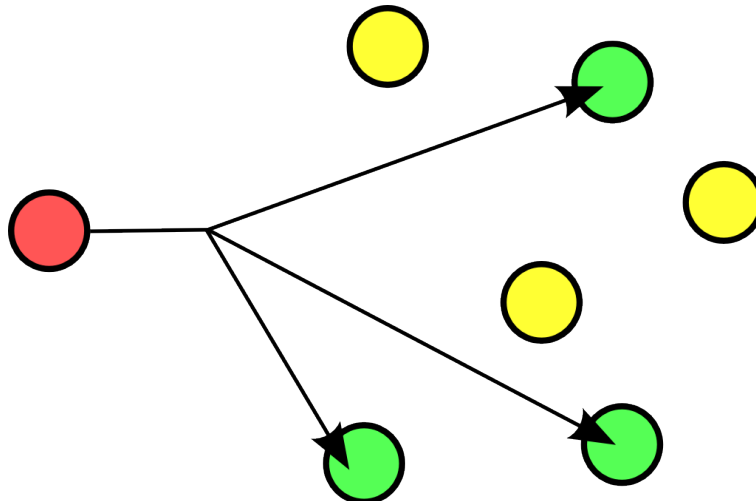
unicast

A **unicast** communication originates from one computer and is destined for exactly one other computer (or host). It is common for computers to have many **unicast** communications.



multicast

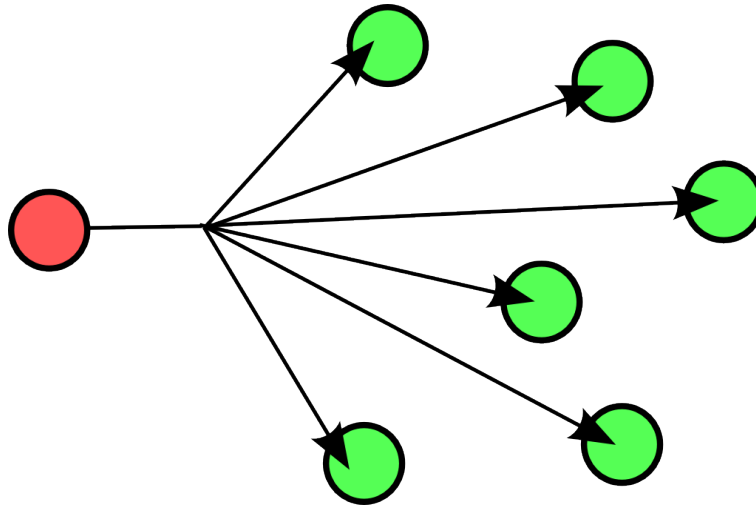
A **multicast** is destined for a group (of computers).



Some examples of **multicast** are Realplayer (.sdp files) and **ripv2** (a routing protocol).

broadcast

A **broadcast** is meant for everyone.

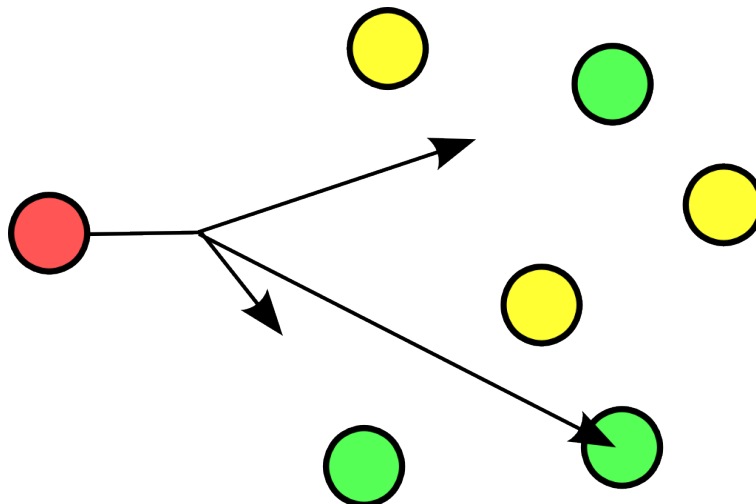


Typical example here is the BBC (British Broadcasting Corporation) broadcasting to everyone. In datacommunications a broadcast is most common confined to the **lan**.

Careful, a **layer 2 broadcast** is very different from a **layer 3 broadcast**. A layer two broadcast is received by all network cards on the same segment (it does not pass any router), whereas a layer 3 broadcast is received by all hosts in the same ip subnet.

anycast

The **root name servers** of the internet use **anycast**. An **anycast** signal goes to the (geographically) nearest of a well defined group.



With thanks to the nice anonymous wikipedia contributor to put these pictures in the public domain.

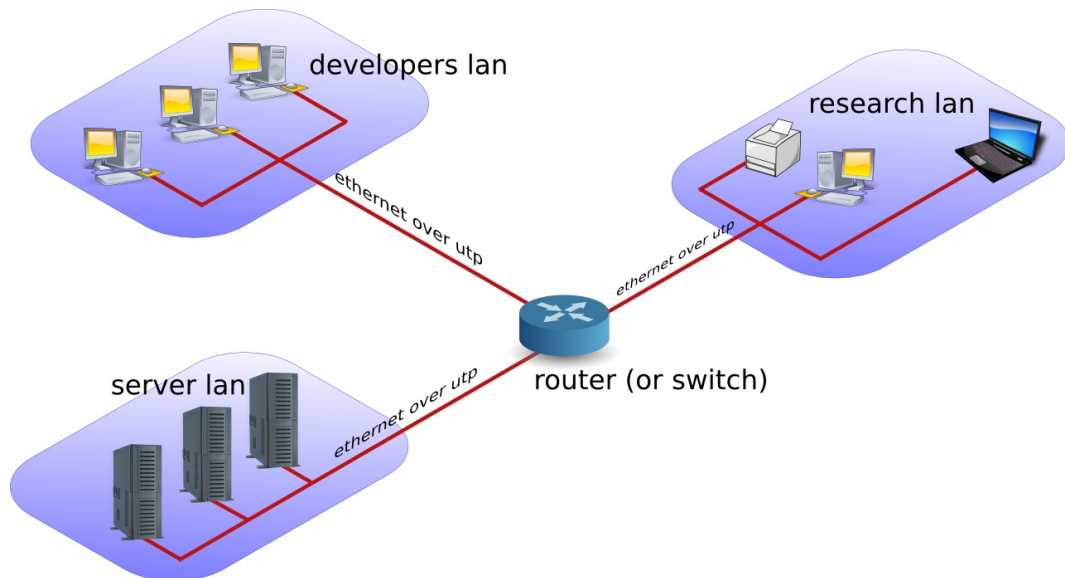
46.3. lan-wan-man

The term **lan** is used for local area networks, as opposed to a **wan** for wide area networks. The difference between the two is determined by the **distance** between the computers, and not by the number of computers in a network. Some protocols like **atm** are designed for use in a **wan**, others like **ethernet** are designed for use in a **lan**.

lan

A **lan** (Local Area Network) is a local network. This can be one room, or one floor, or even one big building. We say **lan** as long as computers are **close** to each other. You can also define a **lan** when all computers are **ethernet** connected.

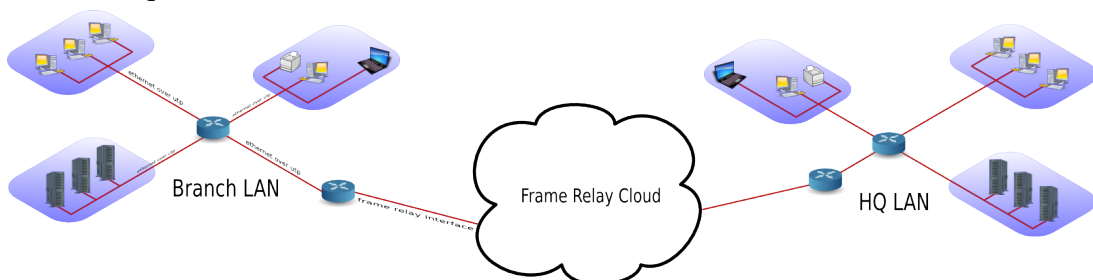
A **lan** can contain multiple smaller **lan**'s. The picture below shows three **lan**'s that together make up one **lan**.



wan

A **wan** (Wide Area Network) is a network with a lot of distance between the computers (or hosts). These hosts are often connected by **leased lines**. A **wan** does not use **ethernet**, but protocols like **fdi**, **frame relay**, **ATM** or **X.25** to connect computers (and networks).

The picture below shows a branch office that is connected through **Frame Relay** with headquarters.



The acronym **wan** is also used for large surface area networks like the **internet**.

Cisco is known for their **wan** technology. They make **routers** that connect many **lan** networks using **wan** protocols.

man

A **man** (Metropolitan Area Network) is something inbetween a **lan** and a **wan**, often comprising several buildings on the same campus or in the same city. A **man** can use **fdi** or **ethernet** or other protocols for connectivity.

pan-wpan

Your home network is called a **pan** (Personal Area Network). A wireless **pan** is a **wpan**.

46.4. internet - intranet - extranet

The **internet** is a global network. It connects many networks using the **tcp/ip** protocol stack.

The origin of the **internet** is the **arpanet**. The **arpanet** was created in 1969, that year only four computers were connected in the network. In 1971 the first **e-mail** was sent over the **arpanet**. **E-mail** took 75 percent of all **arpanet** traffic in 1973. 1973 was also the year **ftp** was introduced, and saw the connection of the first European countries (Norway and UK). In 2009 the internet was available to 25 percent of the world population. In 2011 it is estimated that only a quarter of internet webpages are in English.

An **intranet** is a private **tcp/ip** network. An **intranet** uses the same protocols as the **internet**, but is only accessible to people from within one organization.

An **extranet** is similar to an **intranet**, but some trusted organizations (partners/clients/suppliers/...) also get access.

46.5. tcp/ip

history of tcp/ip

In the Sixties development of the **tcp/ip** protocol stack was started by the US Department of Defense. In the Eighties a lot of commercial enterprises developed their own protocol stack: IBM created **sna**, Novell had **ipx/spx**, Microsoft completed **netbeui** and Apple worked with **appletalk**. All the efforts from the Eighties failed to survive the Nineties. By the end of the Nineties, almost all computers in the world were able to speak tcp/ip.

In my humble opinion, the main reason for the survival of **tcp/ip** over all the other protocols is its openness. Everyone is free to develop and use the tcp/ip protocol suite.

rfc (request for comment)

The protocols that are used on the internet are defined in **rfc's**. An rfc or **request for comment** describes the inner working of all internet protocols. The **IETF** (Internet Engineering Task Force) is the sole publisher of these protocols since 1986.

The official website for the rfc's is <http://www.rfc-editor.org>. This website contains all rfc's in plain text, for example rfc2132 (which defines dhcp and bootp) is accessible at <http://www.rfc-editor.org/rfc/rfc2132.txt>.

many protocols

For reliable connections, you use **tcp**, whereas **udp** is connectionless but faster. The **icmp** error messages are used by **ping**, multicast groups are managed by **igmp**.

These protocols are visible in the protocol field of the ip header, and are listed in the **/etc/protocols** file.

```
paul@debian5:~$ grep tcp /etc/protocols
tcp      6          TCP          # transmission control protocol
```

many services

Network cards are uniquely identified by their **mac address**, hosts by their **ip address** and applications by their **port number**.

Common application level protocols like smtp, http, ssh, telnet and ftp have fixed **port numbers**. There is a list of **port numbers** in **/etc/services**.

```
paul@ubul010:~$ grep ssh /etc/services
ssh      22/tcp     # SSH Remote Login Protocol
ssh      22/udp
```

Chapter 47. interface configuration

Table of Contents

47.1. to gui or not to gui	440
47.2. Debian/Ubuntu nic configuration	441
47.3. Red Hat/Fedora nic configuration	443
47.4. ifconfig	445
47.5. hostname	447
47.6. arp	448
47.7. route	449
47.8. ping	449
47.9. optional: ethtool	450
47.10. practice: interface configuration	451
47.11. solution: interface configuration	452

This chapter explains how to configure **network interface cards** to work with **tcp/ip**.

47.1. to gui or not to gui

Recent Linux distributions often include a graphical application to configure the network. Some people complain that these applications mess networking configurations up when used simultaneously with command line configurations. Notably **Network Manager** (often replaced by **wicd**) and **yast** are known to not care about configuration changes via the command line.

Since the goal of this course is **server** administration, we will assume our Linux servers are always administered through the command line.

This chapter only focuses on using the command line for network interface configuration!

Unfortunately there is no single combination of Linux commands and **/etc** files that works on all Linux distributions. We discuss networking on two (large but distinct) Linux distribution families.

We start with **Debian/Ubuntu**, then continue with **Fedora/RHEL**.

47.2. Debian/Ubuntu nic configuration

/etc/network/interfaces

The **/etc/network/interfaces** file is a core network interface card configuration file on Ubuntu and Debian.

dhcp client

The screenshot below shows that our current Ubuntu 11.04 is configured for **dhcp** on **eth0** (the first network interface card or nic).

```
root@ubull104srv:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp
```

Configuring network cards for **dhcp** is good practice for clients, but servers usually require a **fixed ip address**.

fixed ip

The screenshot below shows **/etc/network/interfaces** configured with a **fixed ip address**.

```
root@ubull104srv:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.33.100
network 192.168.33.0
netmask 255.255.255.0
gateway 192.168.33.1
```

The screenshot above also shows that you can provide more configuration than just the ip address. See **interfaces(5)** for help on setting a **gateway**, **netmask** or any of the other options.

/sbin/ifdown

It is advised (but not mandatory) to down an interface before changing its configuration. This can be done with the **ifdown** command.

The command will not give any output when downing an interface with a fixed ip address. However **ifconfig** will no longer show the interface.

```
root@ubull104srv:~# ifdown eth0
root@ubull104srv:~# ifconfig
lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:106 errors:0 dropped:0 overruns:0 frame:0
      TX packets:106 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:11162 (11.1 KB)  TX bytes:11162 (11.1 KB)
```

An interface that is down cannot be used to connect to the network.

/sbin/ifup

Below a screenshot of **ifup** bringing the **eth0** ethernet interface up using dhcp. (Note that this is a Ubuntu 10.10 screenshot, Ubuntu 11.04 omits **ifup** output by default.)

```
root@ubul010srv:/etc/network# ifup eth0
Internet Systems Consortium DHCP Client V3.1.3
Copyright 2004-2009 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/08:00:27:cd:7f:fc
Sending on   LPF/eth0/08:00:27:cd:7f:fc
Sending on   Socket/fallback
DHCPREQUEST of 192.168.1.34 on eth0 to 255.255.255.255 port 67
DHCPNAK from 192.168.33.100
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3
DHCPOFFER of 192.168.33.77 from 192.168.33.100
DHCPREQUEST of 192.168.33.77 on eth0 to 255.255.255.255 port 67
DHCPACK of 192.168.33.77 from 192.168.33.100
bound to 192.168.33.77 -- renewal in 95 seconds.
ssh stop/waiting
ssh start/running, process 1301
root@ubul010srv:/etc/network#
```

The details of **dhcp** are covered in a separate chapter in the **Linux Servers** course.

47.3. Red Hat/Fedora nic configuration

/etc/sysconfig/network

The **/etc/sysconfig/network** file is a global (across all network cards) configuration file. It allows us to define whether we want networking (**NETWORKING=yes|no**), what the hostname should be (**HOSTNAME=**) and which gateway to use (**GATEWAY=**).

```
[root@rhel6 ~]# cat /etc/sysconfig/network
NETWORKING=yes
HOSTNAME=rhel6
GATEWAY=192.168.1.1
```

There are a dozen more option settable in this file, details can be found in **/usr/share/doc/initscripts-*/sysconfig.txt**.

/etc/sysconfig/network-scripts/ifcfg-

Each network card can be configured individually using the **/etc/sysconfig/network-scripts/ifcfg-*** files. When you have only one network card, then this will probably be **/etc/sysconfig/network-scripts/ifcfg-eth0**.

dhcp client

Below a screenshot of **/etc/sysconfig/network-scripts/ifcfg-eth0** configured for dhcp (**BOOTPROTO="dhcp"**). Note also the **NM_CONTROLLED** parameter to disable control of this nic by **Network Manager**. This parameter is not explained (not even mentioned) in **/usr/share/doc/initscripts-*/sysconfig.txt**, but many others are.

```
[root@rhel6 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
HWADDR="08:00:27:DD:0D:5C"
NM_CONTROLLED="no"
BOOTPROTO="dhcp"
ONBOOT="yes"
```

The **BOOTPROTO** variable can be set to either **dhcp** or **bootp**, anything else will be considered **static** meaning there should be no protocol used at boot time to set the interface values.

fixed ip

Below a screenshot of a **fixed ip** configuration in **/etc/sysconfig/network-scripts/ifcfg-eth0**.

```
[root@rhel6 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
HWADDR="08:00:27:DD:0D:5C"
```

```
NM_CONTROLLED="no"
BOOTPROTO="none"
IPADDR="192.168.1.99"
NETMASK="255.255.255.0"
GATEWAY="192.168.1.1"
ONBOOT="yes"
```

The `HWADDR` can be used to make sure that each network card gets the correct name when multiple network cards are present in the computer. It can not be used to assign a **mac address** to a network card. For this, you need to specify the `MACADDR` variable. Do not use `HWADDR` and `MACADDR` in the same **`ifcfg-ethx`** file.

The `BROADCAST=` and `NETWORK=` parameters from previous RHEL/Fedora versions are obsoleted.

`/sbin/ifup` and `/sbin/ifdown`

The **`ifup`** and **`ifdown`** commands will set an interface up or down, using the configuration discussed above. This is identical to their behaviour in Debian and Ubuntu.

```
[root@rhel6 ~]# ifdown eth0 && ifup eth0
[root@rhel6 ~]# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 08:00:27:DD:0D:5C
      inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fedd:d5c/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
      RX packets:2452 errors:0 dropped:0 overruns:0 frame:0
      TX packets:1881 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:257036 (251.0 KiB) TX bytes:184767 (180.4 KiB)
```

47.4. ifconfig

The use of `/sbin/ifconfig` without any arguments will present you with a list of all active network interface cards, including wireless and the loopback interface. In the screenshot below `eth0` has no ip address.

```
root@ubul010:~# ifconfig
eth0 Link encap:Ethernet HWaddr 00:26:bb:5d:2e:52
     UP BROADCAST MULTICAST MTU:1500 Metric:1
     RX packets:0 errors:0 dropped:0 overruns:0 frame:0
     TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
     Interrupt:43 Base address:0xe000

eth1 Link encap:Ethernet HWaddr 00:26:bb:12:7a:5e
     inet addr:192.168.1.30 Bcast:192.168.1.255 Mask:255.255.255.0
     inet6 addr: fe80::226:bbff:fe12:7a5e/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
     RX packets:11141791 errors:202 dropped:0 overruns:0 frame:11580126
     TX packets:6473056 errors:3860 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:3476531617 (3.4 GB) TX bytes:2114919475 (2.1 GB)
     Interrupt:23

lo    Link encap:Local Loopback
     inet addr:127.0.0.1 Mask:255.0.0.0
     inet6 addr: ::1/128 Scope:Host
     UP LOOPBACK RUNNING MTU:16436 Metric:1
     RX packets:2879 errors:0 dropped:0 overruns:0 frame:0
     TX packets:2879 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:0
     RX bytes:486510 (486.5 KB) TX bytes:486510 (486.5 KB)
```

You can also use `ifconfig` to obtain information about just one network card.

```
[root@rhel6 ~]# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 08:00:27:DD:0D:5C
     inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
     inet6 addr: fe80::a00:27ff:fedd:d5c/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
     RX packets:2969 errors:0 dropped:0 overruns:0 frame:0
     TX packets:1918 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:335942 (328.0 KiB) TX bytes:190157 (185.7 KiB)
```

When `/sbin` is not in the `$PATH` of a normal user you will have to type the full path, as seen here on Debian.

```
paul@debian5:~$ /sbin/ifconfig eth3
eth3 Link encap:Ethernet HWaddr 08:00:27:ab:67:30
     inet addr:192.168.1.29 Bcast:192.168.1.255 Mask:255.255.255.0
     inet6 addr: fe80::a00:27ff:feab:6730/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
     RX packets:27155 errors:0 dropped:0 overruns:0 frame:0
     TX packets:30527 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:13095386 (12.4 MiB) TX bytes:25767221 (24.5 MiB)
```

up and down

You can also use **ifconfig** to bring an interface up or down. The difference with **ifup** is that **ifconfig eth0 up** will re-activate the nic keeping its existing (current) configuration, whereas **ifup** will read the correct file that contains a (possibly new) configuration and use this config file to bring the interface up.

```
[root@rhel6 ~]# ifconfig eth0 down
[root@rhel6 ~]# ifconfig eth0 up
[root@rhel6 ~]# ifconfig eth0
eth0 Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
      inet addr:192.168.1.99  Bcast:192.168.1.255  Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fedd:d5c/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:2995  errors:0  dropped:0  overruns:0  frame:0
      TX packets:1927  errors:0  dropped:0  overruns:0  carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:339030 (331.0 KiB)  TX bytes:191583 (187.0 KiB)
```

setting ip address

You can **temporary** set an ip address with **ifconfig**. This ip address is only valid until the next **ifup/ifdown** cycle or until the next **reboot**.

```
[root@rhel6 ~]# ifconfig eth0 | grep 192
      inet addr:192.168.1.99  Bcast:192.168.1.255  Mask:255.255.255.0
[root@rhel6 ~]# ifconfig eth0 192.168.33.42 netmask 255.255.0.0
[root@rhel6 ~]# ifconfig eth0 | grep 192
      inet addr:192.168.33.42  Bcast:192.168.255.255  Mask:255.255.0.0
[root@rhel6 ~]# ifdown eth0 && ifup eth0
[root@rhel6 ~]# ifconfig eth0 | grep 192
      inet addr:192.168.1.99  Bcast:192.168.1.255  Mask:255.255.255.0
```

setting mac address

You can also use **ifconfig** to set another **mac address** than the one hard coded in the network card. This screenshot shows you how.

```
[root@rhel6 ~]# ifconfig eth0 | grep HWaddr
eth0 Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
[root@rhel6 ~]# ifconfig eth0 hw ether 00:42:42:42:42:42
[root@rhel6 ~]# ifconfig eth0 | grep HWaddr
eth0 Link encap:Ethernet  HWaddr 00:42:42:42:42:42
```

dhclient

Home and client Linux desktops often have **/sbin/dhclient** running. This is a daemon that enables a network interface to lease an ip configuration from a **dhcp server**. When your adapter is configured for **dhcp** or **bootp**, then **/sbin/ifup** will start the **dhclient** daemon.

When a lease is renewed, **dhclient** will override your **ifconfig** set ip address!

47.5. hostname

Every host receives a **hostname**, often placed in a **DNS name space** forming the **fqdn** or Fully Qualified Domain Name.

This screenshot shows the **hostname** command and the configuration of the hostname on Red Hat/Fedora.

```
[root@rhel6 ~]# grep rhel /etc/sysconfig/network
HOSTNAME=rhel6
[root@rhel6 ~]# hostname
rhel6
```

Ubuntu/Debian uses the **/etc/hostname** file to configure the **hostname**.

```
paul@ubul010:~$ cat /etc/hostname
ubul010
paul@ubul010:~$ hostname
ubul010
```

On all Linux distributions you can change the **hostname** using the **hostname \$newname** command. This is not a permanent change.

```
[root@rhel6 ~]# hostname server42
[root@rhel6 ~]# hostname
server42
```

On any Linux you can use **sysctl** to display and set the hostname.

```
[root@rhel6 ~]# sysctl kernel.hostname
kernel.hostname = server42
[root@rhel6 ~]# sysctl kernel.hostname=rhel6
kernel.hostname = rhel6
[root@rhel6 ~]# sysctl kernel.hostname
kernel.hostname = rhel6
[root@rhel6 ~]# hostname
rhel6
```

47.6. arp

The **ip to mac** resolution is handled by the **layer two broadcast** protocol **arp**. The **arp table** can be displayed with the **arp tool**. The screenshot below shows the list of computers that this computer recently communicated with.

```
root@barry:~# arp -a
? (192.168.1.191) at 00:0C:29:3B:15:80 [ether] on eth1
agapi (192.168.1.73) at 00:03:BA:09:7F:D2 [ether] on eth1
anya (192.168.1.1) at 00:12:01:E2:87:FB [ether] on eth1
faith (192.168.1.41) at 00:0E:7F:41:0D:EB [ether] on eth1
kiss (192.168.1.49) at 00:D0:E0:91:79:95 [ether] on eth1
laika (192.168.1.40) at 00:90:F5:4E:AE:17 [ether] on eth1
pasha (192.168.1.71) at 00:03:BA:02:C3:82 [ether] on eth1
shaka (192.168.1.72) at 00:03:BA:09:7C:F9 [ether] on eth1
root@barry:~#
```

Anya is a Cisco Firewall, faith is a laser printer, kiss is a Kiss DP600, laika is a laptop and Agapi, Shaka and Pasha are SPARC servers. The question mark is a Red Hat Enterprise Linux server running on a virtual machine.

You can use **arp -d** to remove an entry from the **arp table**.

```
[root@rhel6 ~]# arp
Address          HWtype  HWaddress          Flags Mask          Iface
ubu1010         ether   00:26:bb:12:7a:5e  C                   eth0
anya            ether   00:02:cf:aa:68:f0  C                   eth0
[root@rhel6 ~]# arp -d anya
[root@rhel6 ~]# arp
Address          HWtype  HWaddress          Flags Mask          Iface
ubu1010         ether   00:26:bb:12:7a:5e  C                   eth0
anya            (incomplete)
[root@rhel6 ~]# ping anya
PING anya (192.168.1.1) 56(84) bytes of data.
64 bytes from anya (192.168.1.1): icmp_seq=1 ttl=254 time=10.2 ms
...
[root@rhel6 ~]# arp
Address          HWtype  HWaddress          Flags Mask          Iface
ubu1010         ether   00:26:bb:12:7a:5e  C                   eth0
anya            ether   00:02:cf:aa:68:f0  C                   eth0
```

47.7. route

You can see the computer's local routing table with the `/sbin/route` command (and also with `netstat -r`).

```
root@RHEL4b ~]# netstat -r
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0      *              255.255.255.0   U        0  0        0 eth0
[root@RHEL4b ~]# route
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0      *              255.255.255.0   U        0      0      0 eth0
[root@RHEL4b ~]#
```

It appears this computer does not have a **gateway** configured, so we use **route add default gw** to add a **default gateway** on the fly.

```
[root@RHEL4b ~]# route add default gw 192.168.1.1
[root@RHEL4b ~]# route
Kernel IP routing table
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0      *              255.255.255.0   U        0      0      0 eth0
default          192.168.1.1    0.0.0.0         UG        0      0      0 eth0
[root@RHEL4b ~]#
```

Unless you configure the gateway in one of the `/etc/` file from the start of this chapter, your computer will forget this **gateway** after a reboot.

47.8. ping

If you can **ping** to another host, then **tcp/ip** is configured.

```
[root@RHEL4b ~]# ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5) 56(84) bytes of data.
64 bytes from 192.168.1.5: icmp_seq=0 ttl=64 time=1004 ms
64 bytes from 192.168.1.5: icmp_seq=1 ttl=64 time=1.19 ms
64 bytes from 192.168.1.5: icmp_seq=2 ttl=64 time=0.494 ms
64 bytes from 192.168.1.5: icmp_seq=3 ttl=64 time=0.419 ms

--- 192.168.1.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/mdev = 0.419/251.574/1004.186/434.520 ms, pipe 2
[root@RHEL4b ~]#
```

47.9. optional: ethtool

To display or change network card settings, use **ethtool**. The results depend on the capabilities of your network card. The example shows a network that auto-negotiates its bandwidth.

```
root@laika:~# ethtool eth0
Settings for eth0:
Supported ports: [ TP ]
Supported link modes:   10baseT/Half 10baseT/Full
                       100baseT/Half 100baseT/Full
                       1000baseT/Full

Supports auto-negotiation: Yes
Advertised link modes:  10baseT/Half 10baseT/Full
                       100baseT/Half 100baseT/Full
                       1000baseT/Full

Advertised auto-negotiation: Yes
Speed: 1000Mb/s
Duplex: Full
Port: Twisted Pair
PHYAD: 0
Transceiver: internal
Auto-negotiation: on
Supports Wake-on: pumbg
Wake-on: g
Current message level: 0x00000033 (51)
Link detected: yes
```

This example shows how to use ethtool to switch the bandwidth from 1000Mbit to 100Mbit and back. Note that some time passes before the nic is back to 1000Mbit.

```
root@laika:~# ethtool eth0 | grep Speed
Speed: 1000Mb/s
root@laika:~# ethtool -s eth0 speed 100
root@laika:~# ethtool eth0 | grep Speed
Speed: 100Mb/s
root@laika:~# ethtool -s eth0 speed 1000
root@laika:~# ethtool eth0 | grep Speed
Speed: 1000Mb/s
```


47.10. practice: interface configuration

1. Verify whether **dhclient** is running.
2. Display your current ip address(es).
3. Display the configuration file where this **ip address** is defined.
4. Follow the **nic configuration** in the book to change your ip address from **dhcp client** to **fixed**. Keep the same **ip address** to avoid conflicts!
5. Did you also configure the correct **gateway** in the previous question ? If not, then do this now.
6. Verify that you have a gateway.
7. Verify that you can connect to the gateway, that it is alive.
8. Change the last two digits of your **mac address**.
9. Which ports are used by http, pop3, ssh, telnet, nntp and ftp ?

Note that **sctp** was omitted from the screenshot.

10. Explain why e-mail and websites are sent over **tcp** and not **udp**.
11. Display the **hostname** of your computer.
12. Which ip-addresses did your computer recently have contact with ?

47.11. solution: interface configuration

1. Verify whether **dhclient** is running.

```
paul@debian5:~$ ps fax | grep dhclient
```

2. Display your current ip address(es).

```
paul@debian5:~$ /sbin/ifconfig | grep 'inet '
    inet addr:192.168.1.31 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:127.0.0.1 Mask:255.0.0.0
```

3. Display the configuration file where this **ip address** is defined.

```
Ubuntu/Debian: cat /etc/network/interfaces
Redhat/Fedora: cat /etc/sysconfig/network-scripts/ifcfg-eth*
```

4. Follow the **nic configuration** in the book to change your ip address from **dhcp client** to **fixed**. Keep the same **ip address** to avoid conflicts!

```
Ubuntu/Debian:
ifdown eth0
vi /etc/network/interfaces
ifup eth0
```

```
Redhat/Fedora:
ifdown eth0
vi /etc/sysconfig/network-scripts/ifcfg-eth0
ifup eth0
```

5. Did you also configure the correct **gateway** in the previous question ? If not, then do this now.

6. Verify that you have a gateway.

```
paul@debian5:~$ /sbin/route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.1.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.1.1 0.0.0.0 UG 0 0 0 eth0
```

7. Verify that you can connect to the gateway, that it is alive.

```
paul@debian5:~$ ping -c3 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=254 time=2.28 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=254 time=2.94 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=254 time=2.34 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 2.283/2.524/2.941/0.296 ms
```

8. Change the last two digits of your **mac address**.

```
[root@rhel6 ~]# ifconfig eth0 hw ether 08:00:27:ab:67:XX
```

9. Which ports are used by http, pop3, ssh, telnet, nntp and ftp ?

```
root@rhel6 ~# grep '^http ' /etc/services
```

```
http      80/tcp      www www-http    # WorldWideWeb HTTP
http      80/udp      www www-http    # HyperText Transfer Protocol
root@rhel6 ~# grep '^smtp ' /etc/services
smtp      25/tcp      mail
smtp      25/udp      mail
root@rhel6 ~# grep '^ssh ' /etc/services
ssh       22/tcp      # The Secure Shell (SSH) Protocol
ssh       22/udp      # The Secure Shell (SSH) Protocol
root@rhel6 ~# grep '^telnet ' /etc/services
telnet    23/tcp
telnet    23/udp
root@rhel6 ~# grep '^nntp ' /etc/services
nntp      119/tcp     readnews untp   # USENET News Transfer Protocol
nntp      119/udp     readnews untp   # USENET News Transfer Protocol
root@rhel6 ~# grep '^ftp ' /etc/services
ftp       21/tcp
ftp       21/udp      fsp fspd
```

Note that **sctp** was omitted from the screenshot.

10. Explain why e-mail and websites are sent over **tcp** and not **udp**.

Because tcp is reliable and udp is not.

11. Display the **hostname** of your computer.

```
paul@debian5:~$ hostname
debian5
```

12. Which ip-addresses did your computer recently have contact with ?

```
root@rhel6 ~# arp -a
? (192.168.1.1) at 00:02:cf:aa:68:f0 [ether] on eth2
? (192.168.1.30) at 00:26:bb:12:7a:5e [ether] on eth2
? (192.168.1.31) at 08:00:27:8e:8a:a8 [ether] on eth2
```

Chapter 48. network sniffing

Table of Contents

48.1. wireshark	455
48.2. tcpdump	457
48.3. practice: network sniffing	458
48.4. solution: network sniffing	459

A good network administrator should be able to use a sniffer like **wireshark** or **tcpdump** to troubleshoot network problems.

A good student will often use a sniffer to learn about networking. This chapter introduces you to **network sniffing**.

48.1. wireshark

installing wireshark

This example shows how to install **wireshark** on **.deb** based distributions like Ubuntu and Debian.

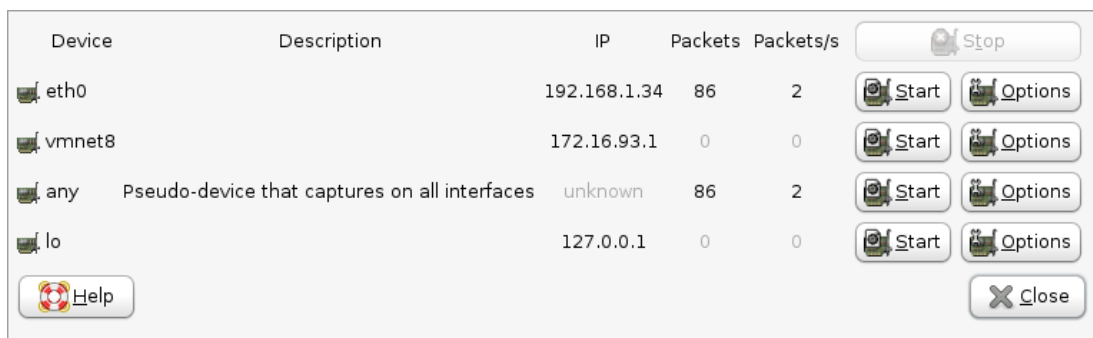
```
aptitude install wireshark
```

On **.rpm** based distributions you can use **yum** to install wireshark.

```
yum install wireshark
```

selecting interface

When you first fire up wireshark, you will need to select an interface to sniff. You will see a dialog box that looks similar to this. Choose the interface that you want to sniff.



On some distributions only root is allowed to sniff the network. You might need to use **sudo wireshark**.

start sniffing

In this example here, we sniffed a ping between two computers. The top pane shows that wireshark recognizes the icmp protocol, and captured all the ping packets between the two computers.

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.34	192.168.1.1	ICMP	Echo (ping) request
2	0.000389	192.168.1.1	192.168.1.34	ICMP	Echo (ping) reply
3	1.000001	192.168.1.34	192.168.1.1	ICMP	Echo (ping) request
4	1.000378	192.168.1.1	192.168.1.34	ICMP	Echo (ping) reply
5	1.999996	192.168.1.34	192.168.1.1	ICMP	Echo (ping) request
6	2.000391	192.168.1.1	192.168.1.34	ICMP	Echo (ping) reply
7	3.000004	192.168.1.34	192.168.1.1	ICMP	Echo (ping) request
8	3.000380	192.168.1.1	192.168.1.34	ICMP	Echo (ping) reply

looking inside packets

The middle can be expanded. When selecting a line in this panel, you can see the corresponding bytes in the frame in the bottom panel.

The screenshot shows a network packet analysis tool interface. The top panel displays the following details:

- Frame 1 (98 bytes on wire, 98 bytes captured)
- Ethernet II, Src: Clevo_4e:ae:17 (00:90:f5:4e:ae:17), Dst: Arcadyan_2a:c5:0b (00:12:bf:2a:c5:0b)
 - Destination: Arcadyan_2a:c5:0b (00:12:bf:2a:c5:0b)
 - Source: Clevo_4e:ae:17 (00:90:f5:4e:ae:17)
- Type: IP (0x0800)
- Internet Protocol, Src: 192.168.1.34 (192.168.1.34), Dst: 192.168.1.1 (192.168.1.1)
- Internet Control Message Protocol

The bottom panel shows a hex dump of the packet data:

```

0000 00 12 bf 2a c5 0b 00 90 f5 4e ae 17 08 00 45 00  ...*... .N...E.
0010 00 54 00 00 40 00 40 01 b7 35 c0 a8 01 22 c0 a8  .T..@.@. .5...."
0020 01 01 08 00 4b f2 43 2a 00 99 1f c6 89 49 d1 37  ...K.C* .....I.7
0030 03 00 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15  .....
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  ..... !"#%
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,- ./012345
0060 36 37 67
  
```

use filters

You might get lost in too many packets. A quick solution to see only the packets that are of interest to you is to apply filters. When you type **arp** and click apply, you will only see **arp** packets displayed.

You can combine two protocols with a logical **or** between them. The example below shows how to filter only **arp** and **bootp** (or **dhcp**) packets.

The screenshot shows a filter input field with the text "arp or bootp". To the right of the input field are three buttons: "+ Expression...", "Clear" (with a trash icon), and "Apply" (with a checkmark icon).

This example shows how to filter for **dns** traffic containing a certain **ip address**.

The screenshot shows a filter input field with the text "dns and ip.addr==192.168.1.5". To the right of the input field are three buttons: "+ Expression...", "Clear" (with a trash icon), and "Apply" (with a checkmark icon).

48.2. tcpdump

Sniffing on the command line can be done with **tcpdump**. Here are some examples.

Using the **tcpdump host \$ip** command displays all traffic with one host (192.168.1.38 in this example).

```
root@ubuntu910:~# tcpdump host 192.168.1.38
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
```

Capturing only ssh (tcp port 22) traffic can be done with **tcpdump tcp port \$port**. This screenshot is cropped to 76 characters for readability in the pdf.

```
root@deb503:~# tcpdump tcp port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
14:22:20.716313 IP deb503.local.37973 > rhel53.local.ssh: P 666050963:66605
14:22:20.719936 IP rhel53.local.ssh > deb503.local.37973: P 1:49(48) ack 48
14:22:20.720922 IP rhel53.local.ssh > deb503.local.37973: P 49:113(64) ack
14:22:20.721321 IP rhel53.local.ssh > deb503.local.37973: P 113:161(48) ack
14:22:20.721820 IP deb503.local.37973 > rhel53.local.ssh: . ack 161 win 200
14:22:20.722492 IP rhel53.local.ssh > deb503.local.37973: P 161:225(64) ack
14:22:20.760602 IP deb503.local.37973 > rhel53.local.ssh: . ack 225 win 200
14:22:23.108106 IP deb503.local.54424 > ubuntu910.local.ssh: P 467252637:46
14:22:23.116804 IP ubuntu910.local.ssh > deb503.local.54424: P 1:81(80) ack
14:22:23.116844 IP deb503.local.54424 > ubuntu910.local.ssh: . ack 81 win 2
^C
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

Same as above, but write the output to a file with the **tcpdump -w \$filename** command.

```
root@ubuntu910:~# tcpdump -w sshdump.tcpdump tcp port 22
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
^C
17 packets captured
17 packets received by filter
0 packets dropped by kernel
```

With **tcpdump -r \$filename** the file created above can be displayed.

```
root@ubuntu910:~# tcpdump -r sshdump.tcpdump
```

Many more examples can be found in the manual page of **tcpdump**.

48.3. practice: network sniffing

1. Install wireshark on your computer (not inside a virtual machine).
2. Start a ping between your computer and another computer.
3. Start sniffing the network.
4. Display only the ping echo's in the top pane using a filter.
5. Now ping to a name (like `www.linux-training.be`) and try to sniff the DNS query and response. Which DNS server was used ? Was it a tcp or udp query and response ?

48.4. solution: network sniffing

1. Install wireshark on your computer (not inside a virtual machine).

Debian/Ubuntu: `aptitude install wireshark`

Red Hat/Mandriva/Fedora: `yum install wireshark`

2. Start a ping between your computer and another computer.

`ping $ip_address`

3. Start sniffing the network.

`(sudo) wireshark`

select an interface (probably eth0)

4. Display only the ping echo's in the top pane using a filter.

type 'icmp' (without quotes) in the filter box, and then click 'apply'

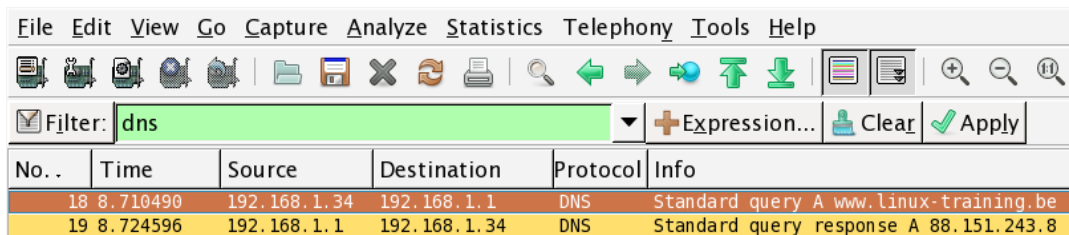
5. Now ping to a name (like `www.linux-training.be`) and try to sniff the DNS query and response. Which DNS server was used ? Was it a tcp or udp query and response ?

First start the sniffer.

Enter 'dns' in the filter box and click apply.

```
root@ubuntu910:~# ping www.linux-training.be
PING www.linux-training.be (88.151.243.8) 56(84) bytes of data.
64 bytes from fosfor.openminds.be (88.151.243.8): icmp_seq=1 ttl=58 time=14.9 ms
64 bytes from fosfor.openminds.be (88.151.243.8): icmp_seq=2 ttl=58 time=16.0 ms
^C
--- www.linux-training.be ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 14.984/15.539/16.095/0.569 ms
```

The wireshark screen should look something like this.



No.	Time	Source	Destination	Protocol	Info
18	8.710490	192.168.1.34	192.168.1.1	DNS	Standard query A www.linux-training.be
19	8.724596	192.168.1.1	192.168.1.34	DNS	Standard query response A 88.151.243.8

The details in wireshark will say the DNS query was inside a udp packet.

Chapter 49. binding and bonding

Table of Contents

49.1. binding on Redhat/Fedora	461
49.2. binding on Debian/Ubuntu	462
49.3. bonding on Redhat/Fedora	463
49.4. bonding on Debian/Ubuntu	465
49.5. practice: binding and bonding	467
49.6. solution: binding and bonding	468

Sometimes a server needs more than one **ip address** on the same network card, we call this **binding** ip addresses.

Linux can also activate multiple network cards behind the same **ip address**, this is called **bonding**.

This chapter will teach you how to configure **binding** and **bonding** on the most common Linux distributions.

49.1. binding on Redhat/Fedora

binding extra ip addresses

To bind more than one **ip address** to the same interface, use **ifcfg-eth0:0**, where the last zero can be anything else. Only two directives are required in the files.

```
[root@rhel6 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0:0
DEVICE="eth0:0"
IPADDR="192.168.1.133"
[root@rhel6 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0:1
DEVICE="eth0:0"
IPADDR="192.168.1.142"
```

enabling extra ip-addresses

To activate a virtual network interface, use **ifup**, to deactivate it, use **ifdown**.

```
[root@rhel6 ~]# ifup eth0:0
[root@rhel6 ~]# ifconfig | grep 'inet '
    inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:192.168.1.133 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:127.0.0.1 Mask:255.0.0.0
[root@rhel6 ~]# ifup eth0:1
[root@rhel6 ~]# ifconfig | grep 'inet '
    inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:192.168.1.133 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:192.168.1.142 Bcast:192.168.1.255 Mask:255.255.255.0
    inet addr:127.0.0.1 Mask:255.0.0.0
```

verifying extra ip-addresses

Use **ping** from another computer to check the activation, or use **ifconfig** like in this screenshot.

```
[root@rhel6 ~]# ifconfig
eth0    Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
        inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
        inet6 addr: fe80::a00:27ff:fedd:d5c/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1259 errors:0 dropped:0 overruns:0 frame:0
        TX packets:545 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:115260 (112.5 KiB)  TX bytes:84293 (82.3 KiB)

eth0:0  Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
        inet addr:192.168.1.133 Bcast:192.168.1.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth0:1  Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
        inet addr:192.168.1.142 Bcast:192.168.1.255 Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

49.2. binding on Debian/Ubuntu

binding extra ip addresses

The configuration of multiple ip addresses on the same network card is done in **/etc/network/interfaces** by adding **eth0:x** devices. Adding the **netmask** is mandatory.

```
debian5:~# cat /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
iface eth0 inet static
address 192.168.1.34
network 192.168.1.0
netmask 255.255.255.0
gateway 192.168.1.1
auto eth0

auto eth0:0
iface eth0:0 inet static
address 192.168.1.233
netmask 255.255.255.0

auto eth0:1
iface eth0:1 inet static
address 192.168.1.242
netmask 255.255.255.0
```

enabling extra ip-addresses

Use **ifup** to enable the extra addresses.

```
debian5:~# ifup eth0:0
debian5:~# ifup eth0:1
```

verifying extra ip-addresses

Use **ping** from another computer to check the activation, or use **ifconfig** like in this screenshot.

```
debian5:~# ifconfig | grep 'inet '
  inet addr:192.168.1.34 Bcast:192.168.1.255 Mask:255.255.255.0
  inet addr:192.168.1.233 Bcast:192.168.1.255 Mask:255.255.255.0
  inet addr:192.168.1.242 Bcast:192.168.1.255 Mask:255.255.255.0
  inet addr:127.0.0.1 Mask:255.0.0.0
```

49.3. bonding on Redhat/Fedora

We start with **ifconfig -a** to get a list of all the network cards on our system.

```
[root@rhel6 network-scripts]# ifconfig -a | grep Ethernet
eth0      Link encap:Ethernet  HWaddr 08:00:27:DD:0D:5C
eth1      Link encap:Ethernet  HWaddr 08:00:27:DA:C1:49
eth2      Link encap:Ethernet  HWaddr 08:00:27:40:03:3B
```

In this demo we decide to bond **eth1** and **eth2**.

We will name are bond **bond0** and add this entry to **modprobe** so the kernel can load the **bonding module** when we bring the interface up.

```
[root@rhel6 network-scripts]# cat /etc/modprobe.d/bonding.conf
alias bond0 bonding
```

Then we create **/etc/sysconfig/network-scripts/ifcfg-bond0** to configure our **bond0** interface.

```
[root@rhel6 network-scripts]# pwd
/etc/sysconfig/network-scripts
[root@rhel6 network-scripts]# cat ifcfg-bond0
DEVICE=bond0
IPADDR=192.168.1.199
NETMASK=255.255.255.0
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

Next we create two files, one for each network card that we will use as slave in **bond0**.

```
[root@rhel6 network-scripts]# cat ifcfg-eth1
DEVICE=eth1
BOOTPROTO=none
ONBOOT=yes
MASTER=bond0
SLAVE=yes
USERCTL=no
[root@rhel6 network-scripts]# cat ifcfg-eth2
DEVICE=eth2
BOOTPROTO=none
ONBOOT=yes
MASTER=bond0
SLAVE=yes
USERCTL=no
```

Finally we bring the interface up with **ifup bond0**.

```
[root@rhel6 network-scripts]# ifup bond0
[root@rhel6 network-scripts]# ifconfig bond0
bond0      Link encap:Ethernet  HWaddr 08:00:27:DA:C1:49
inet addr:192.168.1.199  Bcast:192.168.1.255  Mask:255.255.255.0
inet6 addr: fe80::a00:27ff:fedc:149/64 Scope:Link
UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
RX packets:251 errors:0 dropped:0 overruns:0 frame:0
TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:39852 (38.9 KiB)  TX bytes:1070 (1.0 KiB)
```

The **bond** should also be visible in **/proc/net/bonding**.

binding and bonding

```
[root@rhel6 network-scripts]# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.5.0 (November 4, 2008)
```

```
Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
```

```
Slave Interface: eth1
MII Status: up
Link Failure Count: 0
Permanent HW addr: 08:00:27:da:c1:49
```

```
Slave Interface: eth2
MII Status: up
Link Failure Count: 0
Permanent HW addr: 08:00:27:40:03:3b
```

49.4. bonding on Debian/Ubuntu

We start with **ifconfig -a** to get a list of all the network cards on our system.

```
debian5:~# ifconfig -a | grep Ethernet
eth0      Link encap:Ethernet  HWaddr 08:00:27:bb:18:a4
eth1      Link encap:Ethernet  HWaddr 08:00:27:63:9a:95
eth2      Link encap:Ethernet  HWaddr 08:00:27:27:a4:92
```

In this demo we decide to bond **eth1** and **eth2**.

We also need to install the **ifenslave** package.

```
debian5:~# aptitude search ifenslave
p ifenslave      - Attach and detach slave interfaces to a bonding device
p ifenslave-2.6 - Attach and detach slave interfaces to a bonding device
debian5:~# aptitude install ifenslave
Reading package lists... Done
...
```

Next we update the **/etc/network/interfaces** file with information about the **bond0** interface.

```
debian5:~# tail -7 /etc/network/interfaces
iface bond0 inet static
    address 192.168.1.42
    netmask 255.255.255.0
    gateway 192.168.1.1
    slaves eth1 eth2
    bond-mode active-backup
    bond_primary eth1
```

On older version of Debian/Ubuntu you needed to **modprobe bonding**, but this is no longer required. Use **ifup** to bring the interface up, then test that it works.

```
debian5:~# ifup bond0
debian5:~# ifconfig bond0
bond0      Link encap:Ethernet  HWaddr 08:00:27:63:9a:95
            inet addr:192.168.1.42  Bcast:192.168.1.255  Mask:255.255.255.0
            inet6 addr: fe80::a00:27ff:fe63:9a95/64  Scope:Link
            UP BROADCAST RUNNING MASTER MULTICAST  MTU:1500  Metric:1
            RX packets:212 errors:0 dropped:0 overruns:0 frame:0
            TX packets:39 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:31978 (31.2 KiB)  TX bytes:6709 (6.5 KiB)
```

The **bond** should also be visible in **/proc/net/bonding**.

```
debian5:~# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.2.5 (March 21, 2008)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: eth1
Currently Active Slave: eth1
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth1
MII Status: up
```

binding and bonding

Link Failure Count: 0
Permanent HW addr: 08:00:27:63:9a:95

Slave Interface: eth2
MII Status: up
Link Failure Count: 0
Permanent HW addr: 08:00:27:27:a4:92

49.5. practice: binding and bonding

1. Add an extra **ip address** to one of your network cards. Test that it works (have your neighbour ssh to it)!
2. Use **ifdown** to disable this extra **ip address**.
3. Make sure your neighbour also succeeded in **binding** an extra ip address before you continue.
4. Add an extra network card (or two) to your virtual machine and use the theory to **bond** two network cards.

49.6. solution: binding and bonding

1. Add an extra **ip address** to one of your network cards. Test that it works (have your neighbour ssh to it)!

Redhat/Fedora:
add an `/etc/sysconfig/network-scripts/ifcfg-ethX:X` file
as shown in the theory

Debian/Ubuntu:
expand the `/etc/network/interfaces` file
as shown in the theory

2. Use **ifdown** to disable this extra **ip address**.

```
ifdown eth0:0
```

3. Make sure your neighbour also succeeded in **binding** an extra ip address before you continue.

```
ping $extra_ip_neighbour  
or  
ssh $extra_ip_neighbour
```

4. Add an extra network card (or two) to your virtual machine and use the theory to **bond** two network cards.

Redhat/Fedora:
add `ifcfg-ethX` and `ifcfg-bondX` files in `/etc/sysconfig/network-scripts`
as shown in the theory
and don't forget the `modprobe.conf`

Debian/Ubuntu:
expand the `/etc/network/interfaces` file
as shown in the theory
and don't forget to install the `ifenslave` package

Chapter 50. ssh client and server

Table of Contents

50.1. about ssh	470
50.2. log on to a remote server	472
50.3. executing a command in remote	472
50.4. scp	473
50.5. setting up passwordless ssh	474
50.6. X forwarding via ssh	476
50.7. troubleshooting ssh	476
50.8. sshd	477
50.9. sshd keys	477
50.10. ssh-agent	477
50.11. practice: ssh	478
50.12. solution: ssh	479

The **secure shell** or **ssh** is a collection of tools using a secure protocol for communications with remote Linux computers.

This chapter gives an overview of the most common commands related to the use of the **sshd** server and the **ssh** client.

50.1. about ssh

secure shell

Avoid using **telnet**, **rlogin** and **rsh** to remotely connect to your servers. These older protocols do not encrypt the login session, which means your user id and password can be sniffed by tools like **wireshark** or **tcpdump**. To securely connect to your servers, use **ssh**.

The **ssh protocol** is secure in two ways. Firstly the connection is **encrypted** and secondly the connection is **authenticated** both ways.

An ssh connection always starts with a cryptographic handshake, followed by **encryption** of the transport layer using a symmetric cypher. In other words, the tunnel is encrypted before you start typing anything.

Then **authentication** takes place (using user id/password or public/private keys) and communication can begin over the encrypted connection.

The **ssh protocol** will remember the servers it connected to (and warn you in case something suspicious happened).

The **openssh** package is maintained by the **OpenBSD** people and is distributed with a lot of operating systems (it may even be the most popular package in the world).

/etc/ssh/

Configuration of **ssh** client and server is done in the **/etc/ssh** directory. In the next sections we will discuss most of the files found in **/etc/ssh/**.

ssh protocol versions

The **ssh** protocol has two versions (1 and 2). Avoid using version 1 anywhere, since it contains some known vulnerabilities. You can control the protocol version via **/etc/ssh/ssh_config** for the client side and **/etc/ssh/sshd_config** for the openssh-server daemon.

```
paul@ubul204:/etc/ssh$ grep Protocol ssh_config
# Protocol 2,1
paul@ubul204:/etc/ssh$ grep Protocol sshd_config
Protocol 2
```

public and private keys

The **ssh** protocol uses the well known system of **public and private keys**. The below explanation is succinct, more information can be found on wikipedia.

http://en.wikipedia.org/wiki/Public-key_cryptography

Imagine Alice and Bob, two people that like to communicate with each other. Using **public and private keys** they can communicate with **encryption** and with **authentication**.

When Alice wants to send an encrypted message to Bob, she uses the **public key** of Bob. Bob shares his **public key** with Alice, but keeps his **private key** private! Since Bob is the only one to have Bob's **private key**, Alice is sure that Bob is the only one that can read the encrypted message.

When Bob wants to verify that the message came from Alice, Bob uses the **public key** of Alice to verify that Alice signed the message with her **private key**. Since Alice is the only one to have Alice's **private key**, Bob is sure the message came from Alice.

rsa and dsa algorithms

This chapter does not explain the technical implementation of cryptographic algorithms, it only explains how to use the ssh tools with **rsa** and **dsa**. More information about these algorithms can be found here:

[http://en.wikipedia.org/wiki/RSA_\(algorithm\)](http://en.wikipedia.org/wiki/RSA_(algorithm))
http://en.wikipedia.org/wiki/Digital_Signature_Algorithm

50.2. log on to a remote server

The following screenshot shows how to use **ssh** to log on to a remote computer running Linux. The local user is named **paul** and he is logging on as user **admin42** on the remote system.

```
paul@ubul204:~$ ssh admin42@192.168.1.30
The authenticity of host '192.168.1.30 (192.168.1.30)' can't be established.
RSA key fingerprint is b5:fb:3c:53:50:b4:ab:81:f3:cd:2e:bb:ba:44:d3:75.
Are you sure you want to continue connecting (yes/no)?
```

As you can see, the user **paul** is presented with an **rsa** authentication fingerprint from the remote system. The user can accept this by typing **yes**. We will see later that an entry will be added to the `~/.ssh/known_hosts` file.

```
paul@ubul204:~$ ssh admin42@192.168.1.30
The authenticity of host '192.168.1.30 (192.168.1.30)' can't be established.
RSA key fingerprint is b5:fb:3c:53:50:b4:ab:81:f3:cd:2e:bb:ba:44:d3:75.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.30' (RSA) to the list of known hosts.
admin42@192.168.1.30's password:
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-26-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

1 package can be updated.
0 updates are security updates.

Last login: Wed Jun  6 19:25:57 2012 from 172.28.0.131
admin42@ubuserver:~$
```

The user can get log out of the remote server by typing **exit** or by using **Ctrl-d**.

```
admin42@ubuserver:~$ exit
logout
Connection to 192.168.1.30 closed.
paul@ubul204:~$
```

50.3. executing a command in remote

This screenshot shows how to execute the **pwd** command on the remote server. There is no need to **exit** the server manually.

```
paul@ubul204:~$ ssh admin42@192.168.1.30 pwd
admin42@192.168.1.30's password:
/home/admin42
paul@ubul204:~$
```

50.4. scp

The **scp** command works just like **cp**, but allows the source and destination of the copy to be behind **ssh**. Here is an example where we copy the **/etc/hosts** file from the remote server to the home directory of user paul.

```
paul@ubul204:~$ scp admin42@192.168.1.30:/etc/hosts /home/paul/serverhosts
admin42@192.168.1.30's password:
hosts                               100% 809      0.8KB/s   00:00
```

Here is an example of the reverse, copying a local file to a remote server.

```
paul@ubul204:~$ scp ~/serverhosts admin42@192.168.1.30:/etc/hosts.new
admin42@192.168.1.30's password:
serverhosts                          100% 809      0.8KB/s   00:00
```

50.5. setting up passwordless ssh

To set up passwordless ssh authentication through public/private keys, use **ssh-keygen** to generate a key pair without a passphrase, and then copy your public key to the destination server. Let's do this step by step.

In the example that follows, we will set up ssh without password between Alice and Bob. Alice has an account on a Red Hat Enterprise Linux server, Bob is using Ubuntu on his laptop. Bob wants to give Alice access using ssh and the public and private key system. This means that even if Bob changes his password on his laptop, Alice will still have access.

ssh-keygen

The example below shows how Alice uses **ssh-keygen** to generate a key pair. Alice does not enter a passphrase.

```
[alice@RHEL5 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/alice/.ssh/id_rsa):
Created directory '/home/alice/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alice/.ssh/id_rsa.
Your public key has been saved in /home/alice/.ssh/id_rsa.pub.
The key fingerprint is:
9b:ac:ac:56:c2:98:e5:d9:18:c4:2a:51:72:bb:45:eb alice@RHEL5
[alice@RHEL5 ~]$
```

You can use **ssh-keygen -t dsa** in the same way.

~/.ssh

While **ssh-keygen** generates a public and a private key, it will also create a hidden **.ssh** directory with proper permissions. If you create the **.ssh** directory manually, then you need to **chmod 700** it! Otherwise ssh will refuse to use the keys (world readable private keys are not secure!).

As you can see, the **.ssh** directory is secure in Alice's home directory.

```
[alice@RHEL5 ~]$ ls -ld .ssh
drwx----- 2 alice alice 4096 May  1 07:38 .ssh
[alice@RHEL5 ~]$
```

Bob is using Ubuntu at home. He decides to manually create the **.ssh** directory, so he needs to manually secure it.

```
bob@laika:~$ mkdir .ssh
bob@laika:~$ ls -ld .ssh
drwxr-xr-x 2 bob bob 4096 2008-05-14 16:53 .ssh
bob@laika:~$ chmod 700 .ssh/
bob@laika:~$
```


id_rsa and id_rsa.pub

The `ssh-keygen` command generate two keys in `.ssh`. The public key is named `~/.ssh/id_rsa.pub`. The private key is named `~/.ssh/id_rsa`.

```
[alice@RHEL5 ~]$ ls -l .ssh/
total 16
-rw----- 1 alice alice 1671 May  1 07:38 id_rsa
-rw-r--r-- 1 alice alice  393 May  1 07:38 id_rsa.pub
```

The files will be named `id_dsa` and `id_dsa.pub` when using `dsa` instead of `rsa`.

copy the public key to the other computer

To copy the public key from Alice's server tot Bob's laptop, Alice decides to use `scp`.

```
[alice@RHEL5 .ssh]$ scp id_rsa.pub bob@192.168.48.92:~/.ssh/authorized_keys
bob@192.168.48.92's password:
id_rsa.pub                                100%  393      0.4KB/s   00:00
```

Be careful when copying a second key! Do not overwrite the first key, instead append the key to the same `~/.ssh/authorized_keys` file!

```
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Alice could also have used `ssh-copy-id` like in this example.

```
ssh-copy-id -i .ssh/id_rsa.pub bob@192.168.48.92
```

authorized_keys

In your `~/.ssh` directory, you can create a file called `authorized_keys`. This file can contain one or more public keys from people you trust. Those trusted people can use their private keys to prove their identity and gain access to your account via ssh (without password). The example shows Bob's `authorized_keys` file containing the public key of Alice.

```
bob@laika:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEApCQ9xzyLzJes1sR+hPyqW2vyzt1D4zTLqk\
MDWBR4mMFuUZD/O583I3Lg/Q+JIq0RSksNzaL/BNLDouljMpBe2Dmf/u22u4KmqlJBfDhe\
yTmGSBzeNYCYRSMq78CT9l9a+y6x/shucwhaILsy8A2XfJ9VCgkVtu7XlWFDL2cum08/0\
mRFwVrfc/uPsAn5XkkTsc14g21mQbnp9wJC40pGSJXXMuFOk8MgCb5ieSnpKFniAKM+tEo\
/vjDGSi3F/bxu691jscrU0VUdIoOs098HUfEf7jKBRikxGAC7I4HLA+/zX73OIvRFAb2hv\
tUhn6RHrBtUJUjbsGiYeFTLDfctQ== alice@RHEL5
```

passwordless ssh

Alice can now use ssh to connect passwordless to Bob's laptop. In combination with `ssh`'s capability to execute commands on the remote host, this can be useful in pipes across different machines.

```
[alice@RHEL5 ~]$ ssh bob@192.168.48.92 "ls -l .ssh"
```

```
total 4
-rw-r--r-- 1 bob bob 393 2008-05-14 17:03 authorized_keys
[alice@RHEL5 ~]$
```

50.6. X forwarding via ssh

Another popular feature of **ssh** is called **X11 forwarding** and is implemented with **ssh -X**.

Below an example of X forwarding: user paul logs in as user greet on her computer to start the graphical application mozilla-thunderbird. Although the application will run on the remote computer from greet, it will be displayed on the screen attached locally to paul's computer.

```
paul@debian5:~/PDF$ ssh -X greet@greet.dyndns.org -p 55555
Warning: Permanently added the RSA host key for IP address \
'81.240.174.161' to the list of known hosts.
Password:
Linux raika 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux

Last login: Thu Jan 18 12:35:56 2007
greet@raika:~$ ps fax | grep thun
greet@raika:~$ mozilla-thunderbird &
[1] 30336
```

50.7. troubleshooting ssh

Use **ssh -v** to get debug information about the ssh connection attempt.

```
paul@debian5:~$ ssh -v bert@192.168.1.192
OpenSSH_4.3p2 Debian-8ubuntu1, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /home/paul/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 192.168.1.192 [192.168.1.192] port 22.
debug1: Connection established.
debug1: identity file /home/paul/.ssh/identity type -1
debug1: identity file /home/paul/.ssh/id_rsa type 1
debug1: identity file /home/paul/.ssh/id_dsa type -1
debug1: Remote protocol version 1.99, remote software version OpenSSH_3
debug1: match: OpenSSH_3.9pl pat OpenSSH_3.*
debug1: Enabling compatibility mode for protocol 2.0
...
```

50.8. sshd

The ssh server is called **sshd** and is provided by the **openssh-server** package.

```
root@ubul204~# dpkg -l openssh-server | tail -1
ii openssh-server 1:5.9p1-5ubuntu1 secure shell (SSH) server,...
```

50.9. sshd keys

The public keys used by the sshd server are located in **/etc/ssh** and are world readable. The private keys are only readable by root.

```
root@ubul204~# ls -l /etc/ssh/ssh_host_*
-rw----- 1 root root 668 Jun 7 2011 /etc/ssh/ssh_host_dsa_key
-rw-r--r-- 1 root root 598 Jun 7 2011 /etc/ssh/ssh_host_dsa_key.pub
-rw----- 1 root root 1679 Jun 7 2011 /etc/ssh/ssh_host_rsa_key
-rw-r--r-- 1 root root 390 Jun 7 2011 /etc/ssh/ssh_host_rsa_key.pub
```

50.10. ssh-agent

When generating keys with **ssh-keygen**, you have the option to enter a passphrase to protect access to the keys. To avoid having to type this passphrase every time, you can add the key to **ssh-agent** using **ssh-add**.

Most Linux distributions will start the **ssh-agent** automatically when you log on.

```
root@ubul204~# ps -ef | grep ssh-agent
paul      2405  2365  0 08:13 ?          00:00:00 /usr/bin/ssh-agent...
```

This clipped screenshot shows how to use **ssh-add** to list the keys that are currently added to the **ssh-agent**

```
paul@debian5:~$ ssh-add -L
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAvgI+Vx5UrIsusZP18da8URHGsxG7yivv3/\
...
wMGqa48Kelwom8TGb4Sgcwpp/VO/ldA5m+BGCw== paul@deb503
```

50.11. practice: ssh

0. Make sure that you have access to **two Linux computers**, or work together with a partner for this exercise. For this practice, we will name one of the machines the server.

1. Install **sshd** on the server

2. Verify in the ssh configuration files that only protocol version 2 is allowed.

3. Use **ssh** to log on to the server, show your current directory and then exit the server.

4. Use **scp** to copy a file from your computer to the server.

5. Use **scp** to copy a file from the server to your computer.

6. (optional, only works when you have a graphical install of Linux) Install the xeyes package on the server and use ssh to run xeyes on the server, but display it on your client.

7. (optional, same as previous) Create a bookmark in firefox, then quit firefox on client and server. Use **ssh -X** to run firefox on your display, but on your neighbour's computer. Do you see your neighbour's bookmark ?

8. Use **ssh-keygen** to create a key pair without passphrase. Setup passwordless ssh between you and your neighbour. (or between your client and your server)

9. Verify that the permissions on the server key files are correct; world readable for the public keys and only root access for the private keys.

10. Verify that the **ssh-agent** is running.

11. (optional) Protect your keypair with a **passphrase**, then add this key to the **ssh-agent** and test your passwordless ssh to the server.

50.12. solution: ssh

0. Make sure that you have access to **two Linux computers**, or work together with a partner for this exercise. For this practice, we will name one of the machines the server.

1. Install **sshd** on the server

```
apt-get install openssh-server (on Ubuntu/Debian)
yum -y install openssh-server (on Centos/Fedora/Red Hat)
```

2. Verify in the ssh configuration files that only protocol version 2 is allowed.

```
grep Protocol /etc/ssh/ssh*_config
```

3. Use **ssh** to log on to the server, show your current directory and then exit the server.

```
user@client$ ssh user@server-ip-address
user@server$ pwd
/home/user
user@server$ exit
```

4. Use **scp** to copy a file from your computer to the server.

```
scp localfile user@server:~
```

5. Use **scp** to copy a file from the server to your computer.

```
scp user@server:~/serverfile .
```

6. (optional, only works when you have a graphical install of Linux) Install the **xeyes** package on the server and use **ssh** to run **xeyes** on the server, but display it on your client.

```
on the server:
apt-get install xeyes
on the client:
ssh -X user@server-ip
xeyes
```

7. (optional, same as previous) Create a bookmark in **firefox**, then quit **firefox** on client and server. Use **ssh -X** to run **firefox** on your display, but on your neighbour's computer. Do you see your neighbour's bookmark ?

8. Use **ssh-keygen** to create a key pair without passphrase. Setup passwordless ssh between you and your neighbour. (or between your client and your server)

```
See solution in book "setting up passwordless ssh"
```

9. Verify that the permissions on the server key files are correct; world readable for the public keys and only root access for the private keys.

```
ls -l /etc/ssh/ssh_host_*
```

10. Verify that the **ssh-agent** is running.

```
ps fax | grep ssh-agent
```

11. (optional) Protect your keypair with a **passphrase**, then add this key to the **ssh-agent** and test your passwordless ssh to the server.

```
man ssh-keygen
```

```
man ssh-agent
```

```
man ssh-add
```

Chapter 51. introduction to nfs

Table of Contents

51.1. nfs protocol versions	482
51.2. rpcinfo	482
51.3. server configuration	483
51.4. /etc/exports	483
51.5. exportfs	483
51.6. client configuration	484
51.7. practice : network file system	484

The **network file system** (or simply **nfs**) enables us since the eighties to share a directory with other computers on the network.

In this chapter we see how to setup an **nfs** server and an **nfs** client computer.

51.1. nfs protocol versions

The older **nfs** versions 2 and 3 are stateless (**udp**) by default (but they can use **tcp**).

The more recent **nfs version 4** brings a stateful protocol with better performance and stronger security.

NFS version 4 was defined in **rfc 3010** in 2000 and **rfc 3530** in 2003 and requires **tcp** (port 2049). It also supports **Kerberos** user authentication as an option when mounting a share. NFS versions 2 and 3 authenticate only the host.

51.2. rpcinfo

Clients connect to the server using **rpc** (on Linux this can be managed by the **portmap** daemon. Look at **rpcinfo** to verify that **nfs** and its related services are running.

```
root@RHELv4u2:~# /etc/init.d/portmap status
portmap (pid 1920) is running...
root@RHELv4u2:~# rpcinfo -p
program vers proto  port
100000    2    tcp    111  portmapper
100000    2    udp    111  portmapper
100024    1    udp    32768 status
100024    1    tcp    32769 status
root@RHELv4u2:~# service nfs start
Starting NFS services:                [ OK ]
Starting NFS quotas:                 [ OK ]
Starting NFS daemon:                 [ OK ]
Starting NFS mountd:                 [ OK ]
```

The same **rpcinfo** command when **nfs** is started.

```
root@RHELv4u2:~# rpcinfo -p
program vers proto  port
100000    2    tcp    111  portmapper
100000    2    udp    111  portmapper
100024    1    udp    32768 status
100024    1    tcp    32769 status
100011    1    udp    985  rquotad
100011    2    udp    985  rquotad
100011    1    tcp    988  rquotad
100011    2    tcp    988  rquotad
100003    2    udp    2049 nfs
100003    3    udp    2049 nfs
100003    4    udp    2049 nfs
100003    2    tcp    2049 nfs
100003    3    tcp    2049 nfs
100003    4    tcp    2049 nfs
100021    1    udp    32770 nlockmgr
100021    3    udp    32770 nlockmgr
100021    4    udp    32770 nlockmgr
100021    1    tcp    32789 nlockmgr
100021    3    tcp    32789 nlockmgr
100021    4    tcp    32789 nlockmgr
100005    1    udp    1004 mountd
100005    1    tcp    1007 mountd
100005    2    udp    1004 mountd
100005    2    tcp    1007 mountd
```



```
100005    3    udp    1004    mountd
100005    3    tcp    1007    mountd
root@RHELv4u2:~#
```

51.3. server configuration

nfs is configured in `/etc/exports`. You might want some way (**ldap**?) to synchronize `userid`'s across computers when using **nfs** a lot.

The **root squash** option will change UID 0 to the UID of a **nobody** (or similar) user account. The **sync** option will write writes to disk before completing the client request.

51.4. /etc/exports

Here is a sample `/etc/exports` to explain the syntax:

```
paul@laika:~$ cat /etc/exports
# Everyone can read this share
/mnt/data/iso *(ro)

# Only the computers named pasha and barry can readwrite this one
/var/www pasha(rw) barry(rw)

# same, but without root squashing for barry
/var/ftp pasha(rw) barry(rw,no_root_squash)

# everyone from the netsec.local domain gets access
/var/backup *.netsec.local(rw)

# ro for one network, rw for the other
/var/upload 192.168.1.0/24(ro) 192.168.5.0/24(rw)
```

More recent incarnations of **nfs** require the **subtree_check** option to be explicitly set (or unset with **no_subtree_check**). The `/etc/exports` file then looks like this:

```
root@debian6 ~# cat /etc/exports
# Everyone can read this share
/srv/iso *(ro,no_subtree_check)

# Only the computers named pasha and barry can readwrite this one
/var/www pasha(rw,no_subtree_check) barry(rw,no_subtree_check)

# same, but without root squashing for barry
/var/ftp pasha(rw,no_subtree_check) barry(rw,no_root_squash,no_subtree_check)
```

51.5. exportfs

You don't need to restart the **nfs** server to start exporting your newly created exports. You can use the **exportfs -va** command to do this. It will write the exported directories to `/var/lib/nfs/etab`, where they are immediately applied.

```
root@debian6 ~# exportfs -va
```

```
exporting pasha:/var/ftp
exporting barry:/var/ftp
exporting pasha:/var/www
exporting barry:/var/www
exporting */srv/iso
```

51.6. client configuration

We have seen the **mount** command and the **/etc/fstab** file before.

```
root@RHELv4u2:~# mount -t nfs barry:/mnt/data/iso /home/project55/
root@RHELv4u2:~# cat /etc/fstab | grep nfs
barry:/mnt/data/iso /home/iso nfs defaults 0 0
root@RHELv4u2:~#
```

Here is another simple example. Suppose the **project55** people tell you they only need a couple of CD-ROM images, and you already have them available on an **nfs** server. You could issue the following command to mount this storage on their **/home/project55** mount point.

```
root@RHELv4u2:~# mount -t nfs 192.168.1.40:/mnt/data/iso /home/project55/
root@RHELv4u2:~# ls -lh /home/project55/
total 3.6G
drwxr-xr-x  2 1000 1000 4.0K Jan 16 17:55 RHELv4u1
drwxr-xr-x  2 1000 1000 4.0K Jan 16 14:14 RHELv4u2
drwxr-xr-x  2 1000 1000 4.0K Jan 16 14:54 RHELv4u3
drwxr-xr-x  2 1000 1000 4.0K Jan 16 11:09 RHELv4u4
-rw-r--r--  1 root  root  1.6G Oct 13 15:22 sled10-vmwarews5-vm.zip
root@RHELv4u2:~#
```

51.7. practice : network file system

1. Create two directories with some files. Use **nfs** to share one of them as read only, the other must be writable. Have your neighbour connect to them to test.
2. Investigate the user owner of the files created by your neighbour.
3. Protect a share by ip-address or hostname, so only your neighbour can connect.

Chapter 52. introduction to networking

Table of Contents

52.1. introduction to iptables	486
52.2. practice : iptables	487
52.3. solution : iptables	488
52.4. xinetd and inetd	489
52.5. practice : inetd and xinetd	491
52.6. network file system	492
52.7. practice : network file system	494

52.1. introduction to iptables

iptables firewall

The Linux kernel has a built-in stateful firewall named **iptables**. To stop the **iptables** firewall on Red Hat, use the service command.

```
root@RHELv4u4:~# service iptables stop
Flushing firewall rules:                [ OK ]
Setting chains to policy ACCEPT: filter  [ OK ]
Unloading iptables modules:             [ OK ]
root@RHELv4u4:~#
```

The easy way to configure iptables, is to use a graphical tool like KDE's **kmyfirewall** or **Security Level Configuration Tool**. You can find the latter in the graphical menu, somewhere in System Tools - Security, or you can start it by typing **system-config-securitylevel** in bash. These tools allow for some basic firewall configuration. You can decide whether to enable or disable the firewall, and what typical standard ports are allowed when the firewall is active. You can even add some custom ports. When you are done, the configuration is written to **/etc/sysconfig/iptables** on Red Hat.

```
root@RHELv4u4:~# cat /etc/sysconfig/iptables
# Firewall configuration written by system-config-securitylevel
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
:RH-Firewall-1-INPUT - [0:0]
-A INPUT -j RH-Firewall-1-INPUT
-A FORWARD -j RH-Firewall-1-INPUT
-A RH-Firewall-1-INPUT -i lo -j ACCEPT
-A RH-Firewall-1-INPUT -p icmp --icmp-type any -j ACCEPT
-A RH-Firewall-1-INPUT -p 50 -j ACCEPT
-A RH-Firewall-1-INPUT -p 51 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp --dport 5353 -d 224.0.0.251 -j ACCEPT
-A RH-Firewall-1-INPUT -p udp -m udp --dport 631 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A RH-F...NPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A RH-F...NPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A RH-F...NPUT -m state --state NEW -m tcp -p tcp --dport 21 -j ACCEPT
-A RH-F...NPUT -m state --state NEW -m tcp -p tcp --dport 25 -j ACCEPT
-A RH-Firewall-1-INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
root@RHELv4u4:~#
```

To start the service, issue the **service iptables start** command. You can configure iptables to start at boot time with **chkconfig**.

```
root@RHELv4u4:~# service iptables start
Applying iptables firewall rules:      [ OK ]
root@RHELv4u4:~# chkconfig iptables on
root@RHELv4u4:~#
```

One of the nice features of iptables is that it displays extensive **status** information when queried with the **service iptables status** command.

```
root@RHELv4u4:~# service iptables status
Table: filter
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
RH-Firewall-1-INPUT  all  --  0.0.0.0/0             0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination
RH-Firewall-1-INPUT  all  --  0.0.0.0/0             0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination

Chain RH-Firewall-1-INPUT (2 references)
target     prot opt source                destination
ACCEPT    all  --  0.0.0.0/0             0.0.0.0/0
ACCEPT    icmp --  0.0.0.0/0             0.0.0.0/0    icmp type 255
ACCEPT    esp  --  0.0.0.0/0             0.0.0.0/0
ACCEPT    ah   --  0.0.0.0/0             0.0.0.0/0
ACCEPT    udp  --  0.0.0.0/0             224.0.0.251    udp dpt:5353
ACCEPT    udp  --  0.0.0.0/0             0.0.0.0/0    udp dpt:631
ACCEPT    all  --  0.0.0.0/0             0.0.0.0/0    state RELATED,ESTABLISHED
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0    state NEW tcp dpt:22
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0    state NEW tcp dpt:80
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0    state NEW tcp dpt:21
ACCEPT    tcp  --  0.0.0.0/0             0.0.0.0/0    state NEW tcp dpt:25
REJECT    all  --  0.0.0.0/0             0.0.0.0/0    reject-with icmp-host-prohibited

root@RHELv4u4:~#
```

Mastering firewall configuration requires a decent knowledge of tcp/ip. Good iptables tutorials can be found online here <http://iptables-tutorial.frozentux.net/iptables-tutorial.html> and here <http://tldp.org/HOWTO/IP-Masquerade-HOWTO/>.

52.2. practice : iptables

1. Verify whether the firewall is running.
2. Stop the running firewall.

52.3. solution : iptables

1. Verify whether the firewall is running.

```
root@rhel55 ~# service iptables status | head
Table: filter
Chain INPUT (policy ACCEPT)
num target      prot opt source                destination
1    RH-Firewall-1-INPUT  all  --  0.0.0.0/0              0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num target      prot opt source                destination
1    RH-Firewall-1-INPUT  all  --  0.0.0.0/0              0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
```

2. Stop the running firewall.

```
root@rhel55 ~# service iptables stop
Flushing firewall rules:                [ OK ]
Setting chains to policy ACCEPT: filter [ OK ]
Unloading iptables modules:             [ OK ]
root@rhel55 ~# service iptables status
Firewall is stopped.
```

52.4. xinetd and inetd

the superdaemon

Back when resources like RAM memory were limited, a super-server was devised to listen to all sockets and start the appropriate daemon only when needed. Services like **swat**, **telnet** and **ftp** are typically served by such a super-server. The **xinetd** superdaemon is more recent than **inetd**. We will discuss the configuration both daemons.

Recent Linux distributions like RHEL5 and Ubuntu10.04 do not activate **inetd** or **xinetd** by default, unless an application requires it.

inetd or xinetd

First verify whether your computer is running **inetd** or **xinetd**. This Debian 4.0 Etch is running **inetd**.

```
root@barry:~# ps fax | grep inet
3870 ?          Ss      0:00 /usr/sbin/inetd
```

This Red Hat Enterprise Linux 4 update 4 is running **xinetd**.

```
[root@RHEL4b ~]# ps fax | grep inet
3003 ?          Ss      0:00 xinetd -stayalive -pidfile /var/run/xinetd.pid
```

Both daemons have the same functionality (listening to many ports, starting other daemons when they are needed), but they have different configuration files.

xinetd superdaemon

The **xinetd** daemon is often called a superdaemon because it listens to a lot of incoming connections, and starts other daemons when they are needed. When a connection request is received, **xinetd** will first check TCP wrappers (`/etc/hosts.allow` and `/etc/hosts.deny`) and then give control of the connection to the other daemon. This superdaemon is configured through `/etc/xinetd.conf` and the files in the directory `/etc/xinetd.d`. Let's first take a look at `/etc/xinetd.conf`.

```
paul@RHELv4u2:~$ cat /etc/xinetd.conf
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
```

```
instances          = 60
log_type           = SYSLOG authpriv
log_on_success     = HOST PID
log_on_failure     = HOST
cps                = 25 30
}
```

```
includedir /etc/xinetd.d
```

```
paul@RHELv4u2:~$
```

According to the settings in this file, xinetd can handle 60 client requests at once. It uses the **authpriv** facility to log the host ip-address and pid of successful daemon spawns. When a service (aka protocol linked to daemon) gets more than 25 cps (connections per second), it holds subsequent requests for 30 seconds.

The directory **/etc/xinetd.d** contains more specific configuration files. Let's also take a look at one of them.

```
paul@RHELv4u2:~$ ls /etc/xinetd.d
amanda      chargen-udp  echo        klogin      rexec      talk
amandaidx  cups-lpd    echo-udp    krb5-telnet rlogin     telnet
amidxtape   daytime     eklogin     kshell      rsh        tftp
auth        daytime-udp finger      ktalk       rsync       time
chargen     dbskkd-cdb  gssftp      ntalk       swat       time-udp
paul@RHELv4u2:~$ cat /etc/xinetd.d/swat
# default: off
# description: SWAT is the Samba Web Admin Tool. Use swat \
#              to configure your Samba server. To use SWAT, \
#              connect to port 901 with your favorite web browser.
service swat
{
port                = 901
socket_type         = stream
wait                = no
only_from           = 127.0.0.1
user                = root
server              = /usr/sbin/swat
log_on_failure     += USERID
disable             = yes
}
paul@RHELv4u2:~$
```

The services should be listed in the **/etc/services** file. Port determines the service port, and must be the same as the port specified in **/etc/services**. The **socket_type** should be set to **stream** for tcp services (and to **dgram** for udp). The **log_on_failure +=** concats the userid to the log message formatted in **/etc/xinetd.conf**. The last setting **disable** can be set to yes or no. Setting this to **no** means the service is enabled!

Check the xinetd and xinetd.conf manual pages for many more configuration options.

inetd superdaemon

This superdaemon has only one configuration file **/etc/inetd.conf**. Every protocol or daemon that it is listening for, gets one line in this file.


```
root@barry:~# grep ftp /etc/inetd.conf
tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /boot/tftp
root@barry:~#
```

You can disable a service in `inetd.conf` above by putting a `#` at the start of that line. Here an example of the disabled vmware web interface (listening on tcp port 902).

```
paul@laika:~$ grep vmware /etc/inetd.conf
#902 stream tcp nowait root /usr/sbin/vmware-authd vmware-authd
```

52.5. practice : inetd and xinetd

1. Verify on all systems whether they are using xinetd or inetd.
2. Look at the configuration files.
3. (If telnet is installable, then replace swat in these questions with telnet) Is swat installed ? If not, then install swat and look at the changes in the (x)inetd configuration. Is swat enabled or disabled ?
4. Disable swat, test it. Enable swat, test it.

52.6. network file system

protocol versions

The older **nfs** versions 2 and 3 are stateless (udp) by default, but they can use tcp. Clients connect to the server using **rpc** (on Linux this is controlled by the **portmap** daemon. Look at **rpcinfo** to verify that **nfs** and its related services are running.

```
root@RHELv4u2:~# /etc/init.d/portmap status
portmap (pid 1920) is running...
root@RHELv4u2:~# rpcinfo -p
program vers proto  port
100000    2    tcp    111  portmapper
100000    2    udp    111  portmapper
100024    1    udp    32768 status
100024    1    tcp    32769 status
root@RHELv4u2:~# service nfs start
Starting NFS services:           [ OK ]
Starting NFS quotas:           [ OK ]
Starting NFS daemon:           [ OK ]
Starting NFS mountd:           [ OK ]
```

The same **rpcinfo** command when **nfs** is started.

```
root@RHELv4u2:~# rpcinfo -p
program vers proto  port
100000    2    tcp    111  portmapper
100000    2    udp    111  portmapper
100024    1    udp    32768 status
100024    1    tcp    32769 status
100011    1    udp    985  rquotad
100011    2    udp    985  rquotad
100011    1    tcp    988  rquotad
100011    2    tcp    988  rquotad
100003    2    udp    2049 nfs
100003    3    udp    2049 nfs
100003    4    udp    2049 nfs
100003    2    tcp    2049 nfs
100003    3    tcp    2049 nfs
100003    4    tcp    2049 nfs
100021    1    udp    32770 nlockmgr
100021    3    udp    32770 nlockmgr
100021    4    udp    32770 nlockmgr
100021    1    tcp    32789 nlockmgr
100021    3    tcp    32789 nlockmgr
100021    4    tcp    32789 nlockmgr
100005    1    udp    1004 mountd
100005    1    tcp    1007 mountd
100005    2    udp    1004 mountd
100005    2    tcp    1007 mountd
100005    3    udp    1004 mountd
100005    3    tcp    1007 mountd
root@RHELv4u2:~#
```

nfs version 4 requires tcp (port 2049) and supports **Kerberos** user authentication as an option. **nfs** authentication only takes place when mounting the share. **nfs** versions 2 and 3 authenticate only the host.

server configuration

nfs is configured in **/etc/exports**. Here is a sample **/etc/exports** to explain the syntax. You need some way (NIS domain or LDAP) to synchronize userid's across computers when using **nfs** a lot. The **rootsquash** option will change UID 0 to the UID of the **nfsnobody** user account. The **sync** option will write writes to disk before completing the client request.

```
paul@laika:~$ cat /etc/exports
# Everyone can read this share
/mnt/data/iso *(ro)

# Only the computers barry and pasha can readwrite this one
/var/www pasha(rw) barry(rw)

# same, but without root squashing for barry
/var/ftp pasha(rw) barry(rw,no_root_squash)

# everyone from the netsec.lan domain gets access
/var/backup *.netsec.lan(rw)

# ro for one network, rw for the other
/var/upload 192.168.1.0/24(ro) 192.168.5.0/24(rw)
```

You don't need to restart the **nfs** server to start exporting your newly created exports. You can use the **exportfs -va** command to do this. It will write the exported directories to **/var/lib/nfs/etab**, where they are immediately applied.

client configuration

We have seen the **mount** command and the **/etc/fstab** file before.

```
root@RHELv4u2:~# mount -t nfs barry:/mnt/data/iso /home/project55/
root@RHELv4u2:~# cat /etc/fstab | grep nfs
barry:/mnt/data/iso /home/iso nfs defaults 0 0
root@RHELv4u2:~#
```

Here is another simple example. Suppose the **project55** people tell you they only need a couple of CD-ROM images, and you already have them available on an **nfs** server. You could issue the following command to mount this storage on their **/home/project55** mount point.

```
root@RHELv4u2:~# mount -t nfs 192.168.1.40:/mnt/data/iso /home/project55/
root@RHELv4u2:~# ls -lh /home/project55/
total 3.6G
drwxr-xr-x 2 1000 1000 4.0K Jan 16 17:55 RHELv4u1
```

```
drwxr-xr-x  2 1000 1000 4.0K Jan 16 14:14 RHELv4u2
drwxr-xr-x  2 1000 1000 4.0K Jan 16 14:54 RHELv4u3
drwxr-xr-x  2 1000 1000 4.0K Jan 16 11:09 RHELv4u4
-rw-r--r--  1 root  root 1.6G Oct 13 15:22 sled10-vmwarews5-vm.zip
root@RHELv4u2:~#
```

52.7. practice : network file system

1. Create two directories with some files. Use **nfs** to share one of them as read only, the other must be writable. Have your neighbour connect to them to test.
2. Investigate the user owner of the files created by your neighbour.
3. Protect a share by ip-address or hostname, so only your neighbour can connect.

Part XIV. kernel management

Chapter 53. the Linux kernel

Table of Contents

53.1. about the Linux kernel	496
53.2. Linux kernel source	498
53.3. kernel boot files	502
53.4. Linux kernel modules	503
53.5. compiling a kernel	507
53.6. compiling one module	510

53.1. about the Linux kernel

kernel versions

In 1991 Linux Torvalds wrote (the first version of) the Linux kernel. He put it online, and other people started contributing code. Over 4000 individuals contributed source code to the latest kernel release (version 2.6.27 in November 2008).

Major Linux kernel versions used to come in even and odd numbers. Versions **2.0**, **2.2**, **2.4** and **2.6** are considered stable kernel versions. Whereas **2.1**, **2.3** and **2.5** were unstable (read development) versions. Since the release of 2.6.0 in January 2004, all development has been done in the 2.6 tree. There is currently no v2.7.x and according to Linus the even/stable vs odd/development scheme is abandoned forever.

uname -r

To see your current Linux kernel version, issue the **uname -r** command as shown below.

This first example shows Linux major version **2.6** and minor version **24**. The rest **-22-generic** is specific to the distribution (Ubuntu in this case).

```
paul@laika:~$ uname -r
2.6.24-22-generic
```

The same command on Red Hat Enterprise Linux shows an older kernel (2.6.18) with **-92.1.17.el5** being specific to the distribution.

```
[paul@RHEL52 ~]$ uname -r
2.6.18-92.1.17.el5
```

/proc/cmdline

The parameters that were passed to the kernel at boot time are in **/proc/cmdline**.

```
paul@RHELv4u4:~$ cat /proc/cmdline
ro root=/dev/VolGroup00/LogVol00 rhgb quiet
```

single user mode

When booting the kernel with the **single** parameter, it starts in **single user mode**. Linux can start in a bash shell with the **root** user logged on (without password).

Some distributions prevent the use of this feature (at kernel compile time).

init=/bin/bash

Normally the kernel invokes **init** as the first daemon process. Adding **init=/bin/bash** to the kernel parameters will instead invoke bash (again with root logged on without providing a password).

/var/log/messages

The kernel reports during boot to **syslog** which writes a lot of kernel actions in **/var/log/messages**. Looking at this file reveals when the kernel was started, including all the devices that were detected at boot time.

```
[root@RHEL53 ~]# grep -A16 "syslogd 1.4.1:" /var/log/messages|cut -b24-
syslogd 1.4.1: restart.
kernel: klogd 1.4.1, log source = /proc/kmsg started.
kernel: Linux version 2.6.18-128.el5 (mockbuild@hs20-bc1-5.build.red...)
kernel: BIOS-provided physical RAM map:
kernel:  BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
kernel:  BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
kernel:  BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
kernel:  BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
kernel:  BIOS-e820: 0000000001000000 - 000000001fef0000 (usable)
kernel:  BIOS-e820: 000000001fef0000 - 000000001feff000 (ACPI data)
kernel:  BIOS-e820: 000000001feff000 - 000000001ff00000 (ACPI NVS)
kernel:  BIOS-e820: 000000001ff00000 - 0000000020000000 (usable)
kernel:  BIOS-e820: 00000000fec00000 - 00000000fec10000 (reserved)
kernel:  BIOS-e820: 00000000fee00000 - 00000000fee01000 (reserved)
kernel:  BIOS-e820: 00000000fffe0000 - 0000000100000000 (reserved)
kernel: OMB HIGHMEM available.
kernel: 512MB LOWMEM available.
```

This example shows how to use **/var/log/messages** to see kernel information about **/dev/sda**.

```
[root@RHEL53 ~]# grep sda /var/log/messages | cut -b24-
kernel: SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
kernel: sda: Write Protect is off
```

```
kernel: sda: cache data unavailable
kernel: sda: assuming drive cache: write through
kernel: SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
kernel: sda: Write Protect is off
kernel: sda: cache data unavailable
kernel: sda: assuming drive cache: write through
kernel: sda: sdal sda2
kernel: sd 0:0:0:0: Attached scsi disk sda
kernel: EXT3 FS on sdal, internal journal
```

dmesg

The **dmesg** command prints out all the kernel bootup messages (from the last boot).

```
[root@RHEL53 ~]# dmesg | head
Linux version 2.6.18-128.el5 (mockbuild@hs20-bc1-5.build.redhat.com)
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
 BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
 BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 000000001fef0000 (usable)
 BIOS-e820: 000000001fef0000 - 000000001feff000 (ACPI data)
 BIOS-e820: 000000001feff000 - 000000001ff00000 (ACPI NVS)
 BIOS-e820: 000000001ff00000 - 0000000020000000 (usable)
```

Thus to find information about `/dev/sda`, using **dmesg** will yield only kernel messages from the last boot.

```
[root@RHEL53 ~]# dmesg | grep sda
SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
sda: Write Protect is off
sda: Mode Sense: 5d 00 00 00
sda: cache data unavailable
sda: assuming drive cache: write through
SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
sda: Write Protect is off
sda: Mode Sense: 5d 00 00 00
sda: cache data unavailable
sda: assuming drive cache: write through
sda: sdal sda2
sd 0:0:0:0: Attached scsi disk sda
EXT3 FS on sdal, internal journal
```

53.2. Linux kernel source

ftp.kernel.org

The home of the Linux kernel source is **ftp.kernel.org**. It contains all official releases of the Linux kernel source code from 1991. It provides free downloads over http, ftp and rsync of all these releases, as well as changelogs and patches. More information can be obtained on the website **www.kernel.org**.

Anyone can anonymously use an ftp client to access ftp.kernel.org

```
paul@laika:~$ ftp ftp.kernel.org
Connected to pub3.kernel.org.
220 Welcome to ftp.kernel.org.
Name (ftp.kernel.org:paul): anonymous
331 Please specify the password.
Password:
230-      Welcome to the
230-
230-    LINUX KERNEL ARCHIVES
230-      ftp.kernel.org
```

All the Linux kernel versions are located in the pub/linux/kernel/ directory.

```
ftp> ls pub/linux/kernel/v*
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxrwsr-x   2 536      536          4096 Mar 20  2003 v1.0
drwxrwsr-x   2 536      536         20480 Mar 20  2003 v1.1
drwxrwsr-x   2 536      536          8192 Mar 20  2003 v1.2
drwxrwsr-x   2 536      536         40960 Mar 20  2003 v1.3
drwxrwsr-x   3 536      536         16384 Feb 08  2004 v2.0
drwxrwsr-x   2 536      536         53248 Mar 20  2003 v2.1
drwxrwsr-x   3 536      536         12288 Mar 24  2004 v2.2
drwxrwsr-x   2 536      536         24576 Mar 20  2003 v2.3
drwxrwsr-x   5 536      536         28672 Dec 02 08:14 v2.4
drwxrwsr-x   4 536      536         32768 Jul 14  2003 v2.5
drwxrwsr-x   7 536      536        110592 Dec 05 22:36 v2.6
226 Directory send OK.
ftp>
```

/usr/src

On your local computer, the kernel source is located in **/usr/src**. Note though that the structure inside **/usr/src** might be different depending on the distribution that you are using.

First let's take a look at **/usr/src on Debian**. There appear to be two versions of the complete Linux source code there. Looking for a specific file (e1000_main.c) with find reveals it's exact location.

```
paul@barry:~$ ls -l /usr/src/
drwxr-xr-x 20 root root    4096 2006-04-04 22:12 linux-source-2.6.15
drwxr-xr-x 19 root root    4096 2006-07-15 17:32 linux-source-2.6.16
paul@barry:~$ find /usr/src -name e1000_main.c
/usr/src/linux-source-2.6.15/drivers/net/e1000/e1000_main.c
/usr/src/linux-source-2.6.16/drivers/net/e1000/e1000_main.c
```

This is very similar to **/usr/src on Ubuntu**, except there is only one kernel here (and it is newer).

```
paul@laika:~$ ls -l /usr/src/
```

```
drwxr-xr-x 23 root root      4096 2008-11-24 23:28 linux-source-2.6.24
paul@laika:~$ find /usr/src -name "e1000_main.c"
/usr/src/linux-source-2.6.24/drivers/net/e1000/e1000_main.c
```

Now take a look at **/usr/src on Red Hat Enterprise Linux**.

```
[paul@RHEL52 ~]$ ls -l /usr/src/
drwxr-xr-x 5 root root 4096 Dec  5 19:23 kernels
drwxr-xr-x 7 root root 4096 Oct 11 13:22 redhat
```

We will have to dig a little deeper to find the kernel source on Red Hat!

```
[paul@RHEL52 ~]$ cd /usr/src/redhat/BUILD/
[paul@RHEL52 BUILD]$ find . -name "e1000_main.c"
./kernel-2.6.18/linux-2.6.18.i686/drivers/net/e1000/e1000_main.c
```

downloading the kernel source

Debian

Installing the kernel source on Debian is really simple with **aptitude install linux-source**. You can do a search for all linux-source packages first, like in this screenshot.

```
root@barry:~# aptitude search linux-source
v  linux-source          -
v  linux-source-2.6      -
id linux-source-2.6.15   - Linux kernel source for version 2.6.15
i  linux-source-2.6.16   - Linux kernel source for version 2.6.16
p  linux-source-2.6.18   - Linux kernel source for version 2.6.18
p  linux-source-2.6.24   - Linux kernel source for version 2.6.24
```

And then use **aptitude install** to download and install the Debian Linux kernel source code.

```
root@barry:~# aptitude install linux-source-2.6.24
```

When the aptitude is finished, you will see a new file named **/usr/src/linux-source-<version>.tar.bz2**

```
root@barry:/usr/src# ls -lh
drwxr-xr-x 20 root root 4.0K 2006-04-04 22:12 linux-source-2.6.15
drwxr-xr-x 19 root root 4.0K 2006-07-15 17:32 linux-source-2.6.16
-rw-r--r--  1 root root 45M 2008-12-02 10:56 linux-source-2.6.24.tar.bz2
```

Ubuntu

Ubuntu is based on Debian and also uses **aptitude**, so the task is very similar.

```
root@laika:~# aptitude search linux-source
i   linux-source          - Linux kernel source with Ubuntu patches
v   linux-source-2.6      -
i A linux-source-2.6.24   - Linux kernel source for version 2.6.24
root@laika:~# aptitude install linux-source
```

And when aptitude finishes, we end up with a `/usr/src/linux-source-<version>.tar.bz` file.

```
oot@laika:~# ll /usr/src
total 45M
-rw-r--r--  1 root root  45M 2008-11-24 23:30 linux-source-2.6.24.tar.bz2
```

Red Hat Enterprise Linux

The Red Hat kernel source is located on the fourth source cdrom. The file is called **kernel-2.6.9-42.EL.src.rpm** (example for RHELv4u4). It is also available online at `ftp://ftp.redhat.com/pub/redhat/linux/enterprise/5Server/en/os/SRPMS/` (example for RHEL5).

To download the kernel source on RHEL, use this long `wget` command (on one line, without the trailing `\`).

```
wget ftp://ftp.redhat.com/pub/redhat/linux/enterprise/5Server/en/os/\
SRPMS/kernel-`uname -r`.src.rpm
```

When the `wget` download is finished, you end up with a 60M `.rpm` file.

```
[root@RHEL52 src]# ll
total 60M
-rw-r--r--  1 root root  60M Dec  5 20:54 kernel-2.6.18-92.1.17.el5.src.rpm
drwxr-xr-x  5 root root  4.0K Dec  5 19:23 kernels
drwxr-xr-x  7 root root  4.0K Oct 11 13:22 redhat
```

We will need to perform some more steps before this can be used as kernel source code.

First, we issue the `rpm -i kernel-2.6.9-42.EL.src.rpm` command to install this Red Hat package.

```
[root@RHEL52 src]# ll
total 60M
-rw-r--r--  1 root root  60M Dec  5 20:54 kernel-2.6.18-92.1.17.el5.src.rpm
drwxr-xr-x  5 root root  4.0K Dec  5 19:23 kernels
drwxr-xr-x  7 root root  4.0K Oct 11 13:22 redhat
[root@RHEL52 src]# rpm -i kernel-2.6.18-92.1.17.el5.src.rpm
```

The we move to the SPECS directory and perform an **rpmbuild**.

```
[root@RHEL52 ~]# cd /usr/src/redhat/SPECS
[root@RHEL52 SPECS]# rpmbuild -bp -vv --target=i686 kernel-2.6.spec
```

The `rpmbuild` command put the RHEL Linux kernel source code in `/usr/src/redhat/BUILD/kernel-<version>/`.

```
[root@RHEL52 kernel-2.6.18]# pwd
/usr/src/redhat/BUILD/kernel-2.6.18
[root@RHEL52 kernel-2.6.18]# ll
total 20K
drwxr-xr-x  2 root root 4.0K Dec  6 2007 config
-rw-r--r--  1 root root 3.1K Dec  5 20:58 Config.mk
drwxr-xr-x 20 root root 4.0K Dec  5 20:58 linux-2.6.18.i686
drwxr-xr-x 19 root root 4.0K Sep 20 2006 vanilla
drwxr-xr-x  8 root root 4.0K Dec  6 2007 xen
```

53.3. kernel boot files

vmlinuz

The `vmlinuz` file in `/boot` is the compressed kernel.

```
paul@barry:~$ ls -lh /boot | grep vmlinuz
-rw-r--r-- 1 root root 1.2M 2006-03-06 16:22 vmlinuz-2.6.15-1-486
-rw-r--r-- 1 root root 1.1M 2006-03-06 16:30 vmlinuz-2.6.15-1-686
-rw-r--r-- 1 root root 1.3M 2008-02-11 00:00 vmlinuz-2.6.18-6-686
paul@barry:~$
```

initrd

The kernel uses `initrd` (an initial RAM disk) at boot time. The `initrd` is mounted before the kernel loads, and can contain additional drivers and modules. It is a **compressed cpio archive**, so you can look at the contents in this way.

```
root@RHELv4u4:/boot# mkdir /mnt/initrd
root@RHELv4u4:/boot# cp initrd-2.6.9-42.0.3.EL.img TMPinitrd.gz
root@RHELv4u4:/boot# gunzip TMPinitrd.gz
root@RHELv4u4:/boot# file TMPinitrd
TMPinitrd: ASCII cpio archive (SVR4 with no CRC)
root@RHELv4u4:/boot# cd /mnt/initrd/
root@RHELv4u4:/mnt/initrd# cpio -i | /boot/TMPinitrd
4985 blocks
root@RHELv4u4:/mnt/initrd# ls -l
total 76
drwxr-xr-x  2 root root 4096 Feb  5 08:36 bin
drwxr-xr-x  2 root root 4096 Feb  5 08:36 dev
drwxr-xr-x  4 root root 4096 Feb  5 08:36 etc
-rwxr-xr-x  1 root root 1607 Feb  5 08:36 init
drwxr-xr-x  2 root root 4096 Feb  5 08:36 lib
drwxr-xr-x  2 root root 4096 Feb  5 08:36 loopfs
drwxr-xr-x  2 root root 4096 Feb  5 08:36 proc
```

```
lrwxrwxrwx 1 root root    3 Feb  5 08:36 sbin -> bin
drwxr-xr-x 2 root root 4096 Feb  5 08:36 sys
drwxr-xr-x 2 root root 4096 Feb  5 08:36 sysroot
root@RHELv4u4:/mnt/initrd#
```

System.map

The **System.map** contains the symbol table and changes with every kernel compile. The symbol table is also present in **/proc/kallsyms** (pre 2.6 kernels name this file /proc/ksyms).

```
root@RHELv4u4:/boot# head System.map-`uname -r`
00000400 A __kernel_vsyscall
0000041a A SYSENTER_RETURN_OFFSET
00000420 A __kernel_sigreturn
00000440 A __kernel_rt_sigreturn
c0100000 A _text
c0100000 T startup_32
c01000c6 t checkCPUtype
c0100147 t is486
c010014e t is386
c010019f t L6
root@RHELv4u4:/boot# head /proc/kallsyms
c0100228 t _stext
c0100228 t calibrate_delay_direct
c0100228 t stext
c0100337 t calibrate_delay
c01004db t rest_init
c0100580 t do_pre_smp_initcalls
c0100585 t run_init_process
c01005ac t init
c0100789 t early_param_test
c01007ad t early_setup_test
root@RHELv4u4:/boot#
```

.config

The last file copied to the /boot directory is the kernel configuration used for compilation. This file is not necessary in the /boot directory, but it is common practice to put a copy there. It allows you to recompile a kernel, starting from the same configuration as an existing working one.

53.4. Linux kernel modules

about kernel modules

The Linux kernel is a monolithic kernel with loadable modules. These modules contain parts of the kernel used typically for device drivers, file systems and network protocols. Most of the time the necessary kernel modules are loaded automatically and dynamically without administrator interaction.

/lib/modules

The modules are stored in the `/lib/modules/<kernel-version>` directory. There is a separate directory for each kernel that was compiled for your system.

```
paul@laika:~$ ll /lib/modules/
total 12K
drwxr-xr-x 7 root root 4.0K 2008-11-10 14:32 2.6.24-16-generic
drwxr-xr-x 8 root root 4.0K 2008-12-06 15:39 2.6.24-21-generic
drwxr-xr-x 8 root root 4.0K 2008-12-05 12:58 2.6.24-22-generic
```

<module>.ko

The file containing the modules usually ends in `.ko`. This screenshot shows the location of the `isdn` module files.

```
paul@laika:~$ find /lib/modules -name isdn.ko
/lib/modules/2.6.24-21-generic/kernel/drivers/isdn/i4l/isdn.ko
/lib/modules/2.6.24-22-generic/kernel/drivers/isdn/i4l/isdn.ko
/lib/modules/2.6.24-16-generic/kernel/drivers/isdn/i4l/isdn.ko
```

lsmod

To see a list of currently loaded modules, use `lsmod`. You see the name of each loaded module, the size, the use count, and the names of other modules using this one.

```
[root@RHEL52 ~]# lsmod | head -5
Module                Size  Used by
autofs4                24517  2
hidp                   23105  2
rfcomm                 42457  0
l2cap                  29505  10 hidp,rfcomm
```

/proc/modules

`/proc/modules` lists all modules loaded by the kernel. The output would be too long to display here, so lets `grep` for the `vm` module.

We see that `vmmon` and `vmnet` are both loaded. You can display the same information with `lsmod`. Actually `lsmod` only reads and reformats the output of `/proc/modules`.

```
paul@laika:~$ cat /proc/modules | grep vm
vmnet 36896 13 - Live 0xffffffff88b21000 (P)
vmmon 194540 0 - Live 0xffffffff88af0000 (P)
paul@laika:~$ lsmod | grep vm
vmnet                36896  13
vmmon                194540  0
paul@laika:~$
```

module dependencies

Some modules depend on others. In the following example, you can see that the `nfsd` module is used by `exportfs`, `lockd` and `sunrpc`.

```
paul@laika:~$ cat /proc/modules | grep nfsd
nfsd 267432 17 - Live 0xffffffff88a40000
exportfs 7808 1 nfsd, Live 0xffffffff88a3d000
lockd 73520 3 nfs,nfsd, Live 0xffffffff88a2a000
sunrpc 185032 12 nfs,nfsd,lockd, Live 0xffffffff889fb000
paul@laika:~$ lsmod | grep nfsd
nfsd                267432  17
exportfs            7808    1 nfsd
lockd               73520   3 nfs,nfsd
sunrpc             185032  12 nfs,nfsd,lockd
paul@laika:~$
```

insmod

Kernel modules can be manually loaded with the `insmod` command. This is a very simple (and obsolete) way of loading modules. The screenshot shows `insmod` loading the `fat` module (for fat file system support).

```
root@barry:/lib/modules/2.6.17-2-686# pwd
/lib/modules/2.6.17-2-686
root@barry:/lib/modules/2.6.17-2-686# lsmod | grep fat
root@barry:/lib/modules/2.6.17-2-686# insmod kernel/fs/fat/fat.ko
root@barry:/lib/modules/2.6.17-2-686# lsmod | grep fat
fat                46588   0
```

`insmod` is not detecting dependencies, so it fails to load the `isdn` module (because the `isdn` module depends on the `slhc` module).

```
[root@RHEL52 drivers]# pwd
/lib/modules/2.6.18-92.1.18.el5/kernel/drivers
[root@RHEL52 kernel]# insmod isdn/i4l/isdn.ko
insmod: error inserting 'isdn/i4l/isdn.ko': -1 Unknown symbol in module
```

modinfo

As you can see in the screenshot of `modinfo` below, the `isdn` module depends in the `slhc` module.

```
[root@RHEL52 drivers]# modinfo isdn/i4l/isdn.ko | head -6
filename:          isdn/i4l/isdn.ko
license:           GPL
author:            Fritz Elfert
```

```
description:   ISDN4Linux: link layer
srcversion:   99650346E708173496F6739
depends:      slhc
```

modprobe

The big advantage of **modprobe** over **insmod** is that modprobe will load all necessary modules, whereas insmod requires manual loading of dependencies. Another advantage is that you don't need to point to the filename with full path.

This screenshot shows how modprobe loads the isdn module, automatically loading slhc in background.

```
[root@RHEL52 kernel]# lsmod | grep isdn
[root@RHEL52 kernel]# modprobe isdn
[root@RHEL52 kernel]# lsmod | grep isdn
isdn                122433  0
slhc                10561  1 isdn
[root@RHEL52 kernel]#
```

/lib/modules/<kernel>/modules.dep

Module dependencies are stored in **modules.dep**.

```
[root@RHEL52 2.6.18-92.1.18.el5]# pwd
/lib/modules/2.6.18-92.1.18.el5
[root@RHEL52 2.6.18-92.1.18.el5]# head -3 modules.dep
/lib/modules/2.6.18-92.1.18.el5/kernel/drivers/net/tokenring/3c359.ko:
/lib/modules/2.6.18-92.1.18.el5/kernel/drivers/net/pcmcia/3c574_cs.ko:
/lib/modules/2.6.18-92.1.18.el5/kernel/drivers/net/pcmcia/3c589_cs.ko:
```

depmod

The **modules.dep** file can be updated (recreated) with the **depmod** command. In this screenshot no modules were added, so **depmod** generates the same file.

```
root@barry:/lib/modules/2.6.17-2-686# ls -l modules.dep
-rw-r--r-- 1 root root 310676 2008-03-01 16:32 modules.dep
root@barry:/lib/modules/2.6.17-2-686# depmod
root@barry:/lib/modules/2.6.17-2-686# ls -l modules.dep
-rw-r--r-- 1 root root 310676 2008-12-07 13:54 modules.dep
```

rmmod

Similar to insmod, the **rmmod** command is rarely used anymore.


```
[root@RHELv4u3 ~]# modprobe isdn
[root@RHELv4u3 ~]# rmmod slhc
ERROR: Module slhc is in use by isdn
[root@RHELv4u3 ~]# rmmod isdn
[root@RHELv4u3 ~]# rmmod slhc
[root@RHELv4u3 ~]# lsmod | grep isdn
[root@RHELv4u3 ~]#
```

modprobe -r

Contrary to `rmmod`, **modprobe** will automatically remove unneeded modules.

```
[root@RHELv4u3 ~]# modprobe isdn
[root@RHELv4u3 ~]# lsmod | grep isdn
isdn                133537  0
slhc                 7233  1 isdn
[root@RHELv4u3 ~]# modprobe -r isdn
[root@RHELv4u3 ~]# lsmod | grep isdn
[root@RHELv4u3 ~]# lsmod | grep slhc
[root@RHELv4u3 ~]#
```

/etc/modprobe.conf

The `/etc/modprobe.conf` file and the `/etc/modprobe.d` directory can contain aliases (used by humans) and options (for dependent modules) for `modprobe`.

```
[root@RHEL52 ~]# cat /etc/modprobe.conf
alias scsi_hostadapter mptbase
alias scsi_hostadapter1 mptspi
alias scsi_hostadapter2 ata_piix
alias eth0 pcnet32
alias eth2 pcnet32
alias eth1 pcnet32
```

53.5. compiling a kernel

extraversion

Enter into `/usr/src/redhat/BUILD/kernel-2.6.9/linux-2.6.9/` and change the **extraversion** in the Makefile.

```
[root@RHEL52 linux-2.6.18.i686]# pwd
/usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.i686
[root@RHEL52 linux-2.6.18.i686]# vi Makefile
[root@RHEL52 linux-2.6.18.i686]# head -4 Makefile
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 18
EXTRAVERSION = -paul2008
```

make mrproper

Now clean up the source from any previous installs with **make mrproper**. If this is your first after downloading the source code, then this is not needed.

```
[root@RHEL52 linux-2.6.18.i686]# make mrproper
CLEAN   scripts/basic
CLEAN   scripts/kconfig
CLEAN   include/config
CLEAN   .config .config.old
```

.config

Now copy a working **.config** from /boot to our kernel directory. This file contains the configuration that was used for your current working kernel. It determines whether modules are included in compilation or not.

```
[root@RHEL52 linux-2.6.18.i686]# cp /boot/config-2.6.18-92.1.18.el5 .config
```

make menuconfig

Now run **make menuconfig** (or the graphical **make xconfig**). This tool allows you to select whether to compile stuff as a module (m), as part of the kernel (*), or not at all (smaller kernel size). If you remove too much, your kernel will not work. The configuration will be stored in the hidden **.config** file.

```
[root@RHEL52 linux-2.6.18.i686]# make menuconfig
```

make clean

Issue a **make clean** to prepare the kernel for compile. **make clean** will remove most generated files, but keeps your kernel configuration. Running a **make mrproper** at this point would destroy the **.config** file that you built with **make menuconfig**.

```
[root@RHEL52 linux-2.6.18.i686]# make clean
```

make bzImage

And then run **make bzImage**, sit back and relax while the kernel compiles. You can use **time make bzImage** to know how long it takes to compile, so next time you can go for a short walk.

```
[root@RHEL52 linux-2.6.18.i686]# time make bzImage
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/basic/docproc
HOSTCC  scripts/kconfig/conf.o
HOSTCC  scripts/kconfig/kxgettext.o
...
```

This command will end with telling you the location of the **bzImage** file (and with time info if you also specified the time command).

```
Kernel: arch/i386/boot/bzImage is ready (#1)

real 13m59.573s
user 1m22.631s
sys 11m51.034s
[root@RHEL52 linux-2.6.18.i686]#
```

You can already copy this image to `/boot` with **cp arch/i386/boot/bzImage /boot/vmlinuz-<kernel-version>**.

make modules

Now run **make modules**. It can take 20 to 50 minutes to compile all the modules.

```
[root@RHEL52 linux-2.6.18.i686]# time make modules
CHK     include/linux/version.h
CHK     include/linux/utsrelease.h
CC [M]  arch/i386/kernel/msr.o
CC [M]  arch/i386/kernel/cpuid.o
CC [M]  arch/i386/kernel/microcode.o
```

make modules_install

To copy all the compiled modules to `/lib/modules` just run **make modules_install** (takes about 20 seconds). Here's a screenshot from before the command.

```
[root@RHEL52 linux-2.6.18.i686]# ls -l /lib/modules/
total 20
drwxr-xr-x 6 root root 4096 Oct 15 13:09 2.6.18-92.1.13.el5
drwxr-xr-x 6 root root 4096 Nov 11 08:51 2.6.18-92.1.17.el5
drwxr-xr-x 6 root root 4096 Dec  6 07:11 2.6.18-92.1.18.el5
[root@RHEL52 linux-2.6.18.i686]# make modules_install
```

And here is the same directory after. Notice that **make modules_install** created a new directory for the new kernel.

```
[root@RHEL52 linux-2.6.18.i686]# ls -l /lib/modules/
total 24
```

```
drwxr-xr-x 6 root root 4096 Oct 15 13:09 2.6.18-92.1.13.el5
drwxr-xr-x 6 root root 4096 Nov 11 08:51 2.6.18-92.1.17.el5
drwxr-xr-x 6 root root 4096 Dec 6 07:11 2.6.18-92.1.18.el5
drwxr-xr-x 3 root root 4096 Dec 6 08:50 2.6.18-paul2008
```

/boot

We still need to copy the kernel, the System.map and our configuration file to /boot. Strictly speaking the .config file is not obligatory, but it might help you in future compilations of the kernel.

```
[root@RHEL52 ]# pwd
/usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.i686
[root@RHEL52 ]# cp System.map /boot/System.map-2.6.18-paul2008
[root@RHEL52 ]# cp .config /boot/config-2.6.18-paul2008
[root@RHEL52 ]# cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.18-paul2008
```

mkinitrd

The kernel often uses an initrd file at bootup. We can use **mkinitrd** to generate this file. Make sure you use the correct kernel name!

```
[root@RHEL52 ]# pwd
/usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.i686
[root@RHEL52 ]# mkinitrd /boot/initrd-2.6.18-paul2008 2.6.18-paul2008
```

bootloader

Compilation is now finished, don't forget to create an additional stanza in grub or lilo.

53.6. compiling one module

hello.c

A little C program that will be our module.

```
[root@rhel4a kernel_module]# cat hello.c
#include <linux/module.h>
#include <section>

int init_module(void)
{
    printk(KERN_INFO "Start Hello World...\n");
    return 0;
}
```

```
void cleanup_module(void)
{
    printk(KERN_INFO "End Hello World... \n");
}
```

Makefile

The make file for this module.

```
[root@rhel4a kernel_module]# cat Makefile
obj-m += hello.o
all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

These are the only two files needed.

```
[root@rhel4a kernel_module]# ll
total 16
-rw-rw-r-- 1 paul paul 250 Feb 15 19:14 hello.c
-rw-rw-r-- 1 paul paul 153 Feb 15 19:15 Makefile
```

make

The running of the **make** command.

```
[root@rhel4a kernel_module]# make
make -C /lib/modules/2.6.9-paul-2/build M=~/.kernel_module modules
make[1]: Entering dir... `/usr/src/redhat/BUILD/kernel-2.6.9/linux-2.6.9'
CC [M] /home/paul/kernel_module/hello.o
Building modules, stage 2.
MODPOST
CC /home/paul/kernel_module/hello.mod.o
LD [M] /home/paul/kernel_module/hello.ko
make[1]: Leaving dir... `/usr/src/redhat/BUILD/kernel-2.6.9/linux-2.6.9'
[root@rhel4a kernel_module]#
```

Now we have more files.

```
[root@rhel4a kernel_module]# ll
total 172
-rw-rw-r-- 1 paul paul 250 Feb 15 19:14 hello.c
-rw-r--r-- 1 root root 64475 Feb 15 19:15 hello.ko
-rw-r--r-- 1 root root 632 Feb 15 19:15 hello.mod.c
-rw-r--r-- 1 root root 37036 Feb 15 19:15 hello.mod.o
-rw-r--r-- 1 root root 28396 Feb 15 19:15 hello.o
-rw-rw-r-- 1 paul paul 153 Feb 15 19:15 Makefile
[root@rhel4a kernel_module]#
```

hello.ko

Use **modinfo** to verify that it is really a module.

```
[root@rhel4a kernel_module]# modinfo hello.ko
filename:          hello.ko
vermagic:          2.6.9-paul-2 SMP 686 REGPARM 4KSTACKS gcc-3.4
depends:
```

[root@rhel4a kernel_module]#

Good, so now we can load our hello module.

```
[root@rhel4a kernel_module]# lsmod | grep hello
[root@rhel4a kernel_module]# insmod ./hello.ko
[root@rhel4a kernel_module]# lsmod | grep hello
hello                5504  0
[root@rhel4a kernel_module]# tail -1 /var/log/messages
Feb 15 19:16:07 rhel4a kernel: Start Hello World...
[root@rhel4a kernel_module]# rmmod hello
[root@rhel4a kernel_module]#
```

Finally **/var/log/messages** has a little surprise.

```
[root@rhel4a kernel_module]# tail -2 /var/log/messages
Feb 15 19:16:07 rhel4a kernel: Start Hello World...
Feb 15 19:16:35 rhel4a kernel: End Hello World...
[root@rhel4a kernel_module]#
```

Chapter 54. library management

Table of Contents

54.1. introduction	513
54.2. /lib and /usr/lib	513
54.3. ldd	513
54.4. ltrace	514
54.5. dpkg -S and debsums	514
54.6. rpm -qf and rpm -V	514
54.7. tracing with strace	515

54.1. introduction

With **libraries** we are talking about dynamically linked libraries (aka shared objects). These are binaries that contain functions and are not started themselves as programs, but are called by other binaries.

Several programs can use the same library. The name of the library file usually starts with **lib**, followed by the actual name of the library, then the characters **.so** and finally a version number.

54.2. /lib and /usr/lib

When you look at the **/lib** or the **/usr/lib** directory, you will see a lot of symbolic links. Most **libraries** have a detailed version number in their name, but receive a symbolic link from a filename which only contains the major version number.

```
root@rhel53 ~# ls -l /lib/libext*
lrwxrwxrwx 1 root root 16 Feb 18 16:36 /lib/libext2fs.so.2 -> libext2fs.so.2.4
-rwxr-xr-x 1 root root 113K Jun 30 2009 /lib/libext2fs.so.2.4
```

54.3. ldd

Many programs have dependencies on the installation of certain libraries. You can display these dependencies with **ldd**.

This example shows the dependencies of the **su** command.

```
paul@RHEL5 ~$ ldd /bin/su
linux-gate.so.1 => (0x003f7000)
libpam.so.0 => /lib/libpam.so.0 (0x00d5c000)
libpam_misc.so.0 => /lib/libpam_misc.so.0 (0x0073c000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x00aa4000)
libdl.so.2 => /lib/libdl.so.2 (0x00800000)
libc.so.6 => /lib/libc.so.6 (0x00ec1000)
libaudit.so.0 => /lib/libaudit.so.0 (0x0049f000)
/lib/ld-linux.so.2 (0x4769c000)
```

54.4. ltrace

The **ltrace** program allows to see all the calls made to library functions by a program. The example below uses the **-c** option to get only a summary count (there can be many calls), and the **-l** option to only show calls in one library file. All this to see what calls are made when executing **su - serena** as root.

```
root@deb503:~# ltrace -c -l /lib/libpam.so.0 su - serena
serena@deb503:~$ exit
logout
```

% time	seconds	usecs/call	calls	function
70.31	0.014117	14117	1	pam_start
12.36	0.002482	2482	1	pam_open_session
5.17	0.001039	1039	1	pam_acct_mgmt
4.36	0.000876	876	1	pam_end
3.36	0.000675	675	1	pam_close_session
3.22	0.000646	646	1	pam_authenticate
0.48	0.000096	48	2	pam_set_item
0.27	0.000054	54	1	pam_setcred
0.25	0.000050	50	1	pam_getenvlist
0.22	0.000044	44	1	pam_get_item
100.00	0.020079		11	total

54.5. dpkg -S and debsums

Find out on Debian/Ubuntu to which package a library belongs.

```
paul@deb503:/lib$ dpkg -S libext2fs.so.2.4
e2fslibs: /lib/libext2fs.so.2.4
```

You can then verify the integrity of all files in this package using **debsums**.

```
paul@deb503:~$ debsums e2fslibs
/usr/share/doc/e2fslibs/changelog.Debian.gz      OK
/usr/share/doc/e2fslibs/copyright               OK
/lib/libe2p.so.2.3                              OK
/lib/libext2fs.so.2.4                           OK
```

Should a library be broken, then reinstall it with **aptitude reinstall \$package**.

```
root@deb503:~# aptitude reinstall e2fslibs
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Reading task descriptions... Done
The following packages will be REINSTALLED:
  e2fslibs
...
```

54.6. rpm -qf and rpm -V

Find out on Red Hat/Fedora to which package a library belongs.


```
paul@RHEL5 ~$ rpm -qf /lib/libext2fs.so.2.4
e2fsprogs-libs-1.39-8.el5
```

You can then use **rpm -V** to verify all files in this package. In the example below the output shows that the **Size** and the **Time** stamp of the file have changed since installation.

```
root@rhel53 ~# rpm -V e2fsprogs-libs
prelink: /lib/libext2fs.so.2.4: prelinked file size differs
S.?....T    /lib/libext2fs.so.2.4
```

You can then use **yum reinstall \$package** to overwrite the existing library with an original version.

```
root@rhel53 lib# yum reinstall e2fsprogs-libs
Loaded plugins: rhnplugin, security
Setting up Reinstall Process
Resolving Dependencies
--> Running transaction check
---> Package e2fsprogs-libs.i386 0:1.39-23.el5 set to be erased
---> Package e2fsprogs-libs.i386 0:1.39-23.el5 set to be updated
--> Finished Dependency Resolution
...

```

The package verification now reports no problems with the library.

```
root@rhel53 lib# rpm -V e2fsprogs-libs
root@rhel53 lib#
```

54.7. tracing with strace

More detailed tracing of all function calls can be done with **strace**. We start by creating a read only file.

```
root@deb503:~# echo hello > 42.txt
root@deb503:~# chmod 400 42.txt
root@deb503:~# ls -l 42.txt
-r----- 1 root root 6 2011-09-26 12:03 42.txt
```

We open the file with **vi**, but include the **strace** command with an output file for the trace before **vi**. This will create a file with all the function calls done by **vi**.

```
root@deb503:~# strace -o strace.txt vi 42.txt
```

The file is read only, but we still change the contents, and use the **:w!** directive to write to this file. Then we close **vi** and take a look at the trace log.

```
root@deb503:~# grep chmod strace.txt
chmod("42.txt", 0100600)          = -1 ENOENT (No such file or directory)
chmod("42.txt", 0100400)          = 0
root@deb503:~# ls -l 42.txt
-r----- 1 root root 12 2011-09-26 12:04 42.txt
```

Notice that **vi** changed the permissions on the file twice. The trace log is too long to show a complete screenshot in this book.

```
root@deb503:~# wc -l strace.txt
941 strace.txt
```

Part XV. backup management

Chapter 55. backup

Table of Contents

55.1. About tape devices	517
55.2. Compression	518
55.3. tar	519
55.4. Backup Types	521
55.5. dump and restore	521
55.6. cpio	522
55.7. dd	522
55.8. split	524
55.9. practice: backup	524

55.1. About tape devices

Don't forget that the name of a device strictly speaking has no meaning since the kernel will use the major and minor number to find the hardware! See the man page of **mknod** and the devices.txt file in the linux kernel source for more info.

SCSI tapes

On the official Linux device list (<http://www.lanana.org/docs/device-list/>) we find the names for SCSI tapes (major 9 char). SCSI tape devices are located underneath /**dev/st** and are numbered starting with 0 for the first tape device.

```
/dev/st0   First tape device
/dev/st1   Second tape device
/dev/st2   Third tape device
```

To prevent **automatic rewinding of tapes**, prefix them with the letter n.

```
/dev/nst0  First no rewind tape device
/dev/nst1  Second no rewind tape device
/dev/nst2  Third no rewind tape device
```

By default, SCSI tapes on linux will use the highest hardware compression that is supported by the tape device. To lower the compression level, append one of the letters l (low), m (medium) or a (auto) to the tape name.

```
/dev/st0l  First low compression tape device
/dev/st0m  First medium compression tape device
/dev/nst2m Third no rewind medium compression tape device
```

IDE tapes

On the official Linux device list (<http://www.lanana.org/docs/device-list/>) we find the names for IDE tapes (major 37 char). IDE tape devices are located underneath **/dev/ht** and are numbered starting with 0 for the first tape device. No rewind and compression is similar to SCSI tapes.

```
/dev/ht0    First IDE tape device
/dev/nht0   Second no rewind IDE tape device
/dev/ht0m   First medium compression IDE tape device
```

mt

To manage your tapes, use **mt** (Magnetic Tape). Some examples.

To receive information about the status of the tape.

```
mt -f /dev/st0 status
```

To rewind a tape...

```
mt -f /dev/st0 rewind
```

To rewind and eject a tape...

```
mt -f /dev/st0 eject
```

To erase a tape...

```
mt -f /dev/st0 erase
```

55.2. Compression

It can be beneficial to compress files before backup. The two most popular tools for compression of regular files on linux are **gzip/gunzip** and **bzip2/bunzip2**. Below you can see gzip in action, notice that it adds the **.gz** extension to the file.

```
paul@RHELv4u4:~/test$ ls -l allfiles.tx*
-rw-rw-r-- 1 paul paul 8813553 Feb 27 05:38 allfiles.txt
paul@RHELv4u4:~/test$ gzip allfiles.txt
paul@RHELv4u4:~/test$ ls -l allfiles.tx*
-rw-rw-r-- 1 paul paul 931863 Feb 27 05:38 allfiles.txt.gz
paul@RHELv4u4:~/test$ gunzip allfiles.txt.gz
paul@RHELv4u4:~/test$ ls -l allfiles.tx*
-rw-rw-r-- 1 paul paul 8813553 Feb 27 05:38 allfiles.txt
paul@RHELv4u4:~/test$
```

In general, gzip is much faster than bzip2, but the latter one compresses a lot better. Let us compare the two.

```
paul@RHELv4u4:~/test$ cp allfiles.txt bllfiles.txt
```

```

paul@RHELv4u4:~/test$ time gzip allfiles.txt

real    0m0.050s
user    0m0.041s
sys     0m0.009s
paul@RHELv4u4:~/test$ time bzip2 bllfiles.txt

real    0m5.968s
user    0m5.794s
sys     0m0.076s
paul@RHELv4u4:~/test$ ls -l ?llfiles.tx*
-rw-rw-r-- 1 paul paul 931863 Feb 27 05:38 allfiles.txt.gz
-rw-rw-r-- 1 paul paul 708871 May 12 10:52 bllfiles.txt.bz2
paul@RHELv4u4:~/test$

```

55.3. tar

The **tar** utility gets its name from **Tape ARchive**. This tool will receive and send files to a destination (typically a tape or a regular file). The **c** option is used to create a tar archive (or tarfile), the **f** option to name/create the **tarfile**. The example below takes a backup of /etc into the file /backup/etc.tar .

```

root@RHELv4u4:~# tar cf /backup/etc.tar /etc
root@RHELv4u4:~# ls -l /backup/etc.tar
-rw-r--r-- 1 root root 47800320 May 12 11:47 /backup/etc.tar
root@RHELv4u4:~#

```

Compression can be achieved without pipes since tar uses the **z** flag to compress with gzip, and the **j** flag to compress with bzip2.

```

root@RHELv4u4:~# tar czf /backup/etc.tar.gz /etc
root@RHELv4u4:~# tar cjf /backup/etc.tar.bz2 /etc
root@RHELv4u4:~# ls -l /backup/etc.ta*
-rw-r--r-- 1 root root 47800320 May 12 11:47 /backup/etc.tar
-rw-r--r-- 1 root root 6077340 May 12 11:48 /backup/etc.tar.bz2
-rw-r--r-- 1 root root 8496607 May 12 11:47 /backup/etc.tar.gz
root@RHELv4u4:~#

```

The **t** option is used to **list the contents of a tar file**. Verbose mode is enabled with **v** (also useful when you want to see the files being archived during archiving).

```

root@RHELv4u4:~# tar tvf /backup/etc.tar
drwxr-xr-x root/root          0 2007-05-12 09:38:21 etc/
-rw-r--r-- root/root         2657 2004-09-27 10:15:03 etc/warnquota.conf
-rw-r--r-- root/root        13136 2006-11-03 17:34:50 etc/mime.types
drwxr-xr-x root/root          0 2004-11-03 13:35:50 etc/sound/
...

```

To **list a specific file in a tar archive**, use the **t** option, added with the filename (without leading /).

```
root@RHELv4u4:~# tar tvf /backup/etc.tar etc/resolv.conf
-rw-r--r-- root/root          77 2007-05-12 08:31:32 etc/resolv.conf
root@RHELv4u4:~#
```

Use the **x** flag to **restore a tar archive**, or a single file from the archive. Remember that by default tar will restore the file in the current directory.

```
root@RHELv4u4:~# tar xvf /backup/etc.tar etc/resolv.conf
etc/resolv.conf
root@RHELv4u4:~# ls -l /etc/resolv.conf
-rw-r--r--  2 root root 40 May 12 12:05 /etc/resolv.conf
root@RHELv4u4:~# ls -l etc/resolv.conf
-rw-r--r--  1 root root 77 May 12 08:31 etc/resolv.conf
root@RHELv4u4:~#
```

You can **preserve file permissions** with the **p** flag. And you can exclude directories or file with **--exclude**.

```
root ~# tar cpzf /backup/etc_with_perms.tgz /etc
root ~# tar cpzf /backup/etc_no_sysconf.tgz /etc --exclude /etc/sysconfig
root ~# ls -l /backup/etc_*
-rw-r--r--  1 root root 8434293 May 12 12:48 /backup/etc_no_sysconf.tgz
-rw-r--r--  1 root root 8496591 May 12 12:48 /backup/etc_with_perms.tgz
root ~#
```

You can also create a text file with names of files and directories to archive, and then supply this file to tar with the **-T** flag.

```
root@RHELv4u4:~# find /etc -name *.conf > files_to_archive.txt
root@RHELv4u4:~# find /home -name *.pdf >> files_to_archive.txt
root@RHELv4u4:~# tar cpzf /backup/backup.tgz -T files_to_archive.txt
```

The tar utility can receive filenames from the find command, with the help of xargs.

```
find /etc -type f -name "*.conf" | xargs tar czf /backup/confs.tar.gz
```

You can also use tar to copy a directory, this is more efficient than using **cp -r**.

```
(cd /etc; tar -cf - . ) | (cd /backup/copy_of_etc/; tar -xpf - )
```

Another example of tar, this copies a directory securely over the network.

```
(cd /etc;tar -cf - . )|(ssh user@srv 'cd /backup/cp_of_etc/; tar -xf - ')
```

tar can be used together with gzip and copy a file to a remote server through ssh

```
cat backup.tar | gzip | ssh bashuser@192.168.1.105 "cat - > backup.tgz"
```

Compress the tar backup when it is on the network, but leave it uncompressed at the destination.

```
cat backup.tar | gzip | ssh user@192.168.1.105 "gunzip|cat - > backup.tar"
```

Same as the previous, but let ssh handle the compression

```
cat backup.tar | ssh -C bashuser@192.168.1.105 "cat - > backup.tar"
```

55.4. Backup Types

Linux uses **multilevel incremental** backups using distinct levels. A full backup is a backup at level 0. A higher level x backup will include all changes since the last level x-1 backup.

Suppose you take a full backup on Monday (level 0) and a level 1 backup on Tuesday, then the Tuesday backup will contain all changes since Monday. Taking a level 2 on Wednesday will contain all changes since Tuesday (the last level 2-1). A level 3 backup on Thursday will contain all changes since Wednesday (the last level 3-1). Another level 3 on Friday will also contain all changes since Wednesday. A level 2 backup on Saturday would take all changes since the last level 1 from Tuesday.

55.5. dump and restore

While **dump** is similar to tar, it is also very different because it looks at the file system. Where tar receives a lists of files to backup, dump will find files to backup by itself by examining ext2. Files found by dump will be copied to a tape or regular file. In case the target is not big enough to hold the dump (end-of-media), it is broken into multiple volumes.

Restoring files that were backed up with dump is done with the **restore** command. In the example below we take a full level 0 backup of two partitions to a SCSI tape. The no rewind is mandatory to put the volumes behind each other on the tape.

```
dump 0f /dev/nst0 /boot
dump 0f /dev/nst0 /
```

Listing files in a dump archive is done with **dump -t**, and you can compare files with **dump -C**.

You can omit files from a dump by changing the dump attribute with the **chattr** command. The d attribute on ext will tell dump to skip the file, even during a full backup. In the following example, /etc/hosts is excluded from dump archives.

```
chattr +d /etc/hosts
```

To restore the complete file system with **restore**, use the **-r** option. This can be useful to change the size or block size of a file system. You should have a clean file system mounted and cd'd into it. Like this example shows.

```
mke2fs /dev/hda3
mount /dev/hda3 /mnt/data
cd /mnt/data
restore rf /dev/nst0
```

To extract only one file or directory from a dump, use the **-x** option.

```
restore -xf /dev/st0 /etc
```

55.6. cpio

Different from tar and dump is **cpio** (Copy Input and Output). It can be used to receive filenames, but copies the actual files. This makes it an easy companion with find! Some examples below.

find sends filenames to cpio, which puts the files in an archive.

```
find /etc -depth -print | cpio -oaV -O archive.cpio
```

The same, but compressed with gzip

```
find /etc -depth -print | cpio -oaV | gzip -c > archive.cpio.gz
```

Now pipe it through ssh (backup files to a compressed file on another machine)

```
find /etc -depth -print|cpio -oaV|gzip -c|ssh server "cat - > etc.cpio.gz"
```

find sends filenames to cpio | cpio sends files to ssh | ssh sends files to cpio 'cpio extracts files'

```
find /etc -depth -print | cpio -oaV | ssh user@host 'cpio -imVd'
```

the same but reversed: copy a dir from the remote host to the local machine

```
ssh user@host "find path -depth -print | cpio -oaV" | cpio -imVd
```

55.7. dd

About dd

Some people use **dd** to create backups. This can be very powerful, but dd backups can only be restored to very similar partitions or devices. There are however a lot of useful things possible with dd. Some examples.

Create a CDROM image

The easiest way to create a **.ISO file** from any CD. The if switch means Input File, of is the Output File. Any good tool can burn a copy of the CD with this .ISO file.

```
dd if=/dev/cdrom of=/path/to/cdrom.ISO
```

Create a floppy image

A little outdated maybe, but just in case : make an image file from a 1.44MB floppy. Blocksize is defined by bs, and count contains the number of blocks to copy.

```
dd if=/dev/floppy of=/path/to/floppy.img bs=1024 count=1440
```

Copy the master boot record

Use dd to copy the **MBR** (Master Boot Record) of hard disk /dev/hda to a file.

```
dd if=/dev/hda of=/MBR.img bs=512 count=1
```

Copy files

This example shows how dd can copy files. Copy the file summer.txt to copy_of_summer.txt .

```
dd if=~/summer.txt of=~/copy_of_summer.txt
```

Image disks or partitions

And who needs ghost when dd can create a (compressed) image of a partition.

```
dd if=/dev/hdb2 of=/image_of_hdb2.IMG  
dd if=/dev/hdb2 | gzip > /image_of_hdb2.IMG.gz
```

Create files of a certain size

dd can be used to create a file of any size. The first example creates a one MEBIbyte file, the second a one MEGAbyte file.

```
dd if=/dev/zero of=file1MB count=1024 bs=1024  
dd if=/dev/zero of=file1MB count=1000 bs=1024
```

CDROM server example

And there are of course endless combinations with ssh and bzip2. This example puts a bzip2 backup of a cdrom on a remote server.

```
dd if=/dev/cdrom |bzip2|ssh user@host "cat - > /backups/cd/cdrom.iso.bz2"
```

55.8. split

The **split** command is useful to split files into smaller files. This can be useful to fit the file onto multiple instances of a medium too small to contain the complete file. In the example below, a file of size 5000 bytes is split into three smaller files, with maximum 2000 bytes each.

```
paul@laika:~/test$ ls -l
total 8
-rw-r--r-- 1 paul paul 5000 2007-09-09 20:46 bigfile1
paul@laika:~/test$ split -b 2000 bigfile1 splitfile.
paul@laika:~/test$ ls -l
total 20
-rw-r--r-- 1 paul paul 5000 2007-09-09 20:46 bigfile1
-rw-r--r-- 1 paul paul 2000 2007-09-09 20:47 splitfile.aa
-rw-r--r-- 1 paul paul 2000 2007-09-09 20:47 splitfile.ab
-rw-r--r-- 1 paul paul 1000 2007-09-09 20:47 splitfile.ac
```

55.9. practice: backup

!! Careful with tar options and the position of the backup file, mistakes can destroy your system!!

1. Create a directory (or partition if you like) for backups. Link (or mount) it under /mnt/backup.

2a. Use tar to backup /etc in /mnt/backup/etc_date.tgz, the backup must be gzipped. (Replace date with the current date)

2b. Use tar to backup /bin to /mnt/backup/bin_date.tar.bz2, the backup must be bzip2'd.

2c. Choose a file in /etc and /bin and verify with tar that the file is indeed backed up.

2d. Extract those two files to your home directory.

3a. Create a backup directory for your neighbour, make it accessible under /mnt/neighbourName

3b. Combine ssh and tar to put a backup of your /boot on your neighbours computer in /mnt/YourName

4a. Combine find and cpio to create a cpio archive of /etc.

4b. Choose a file in /etc and restore it from the cpio archive into your home directory.

5. Use `dd` and `ssh` to put a backup of the master boot record on your neighbours computer.
6. (On the real computer) Create and mount an ISO image of the ubuntu cdrom.
7. Combine `dd` and `gzip` to create a 'ghost' image of one of your partitions on another partition.
8. Use `dd` to create a five megabyte file in `~/testsplit` and name it `biggest`. Then split this file in smaller two megabyte parts.

```
mkdir testsplit
```

```
dd if=/dev/zero of=~/testsplit/biggest count=5000 bs=1024
```

```
split -b 2000000 biggest parts
```

Part XVI. Introduction to Samba

Chapter 56. introduction to samba

Table of Contents

56.1. verify installed version	528
56.2. installing samba	529
56.3. documentation	530
56.4. starting and stopping samba	531
56.5. samba daemons	532
56.6. the SMB protocol	533
56.7. practice: introduction to samba	534

This introduction to the Samba server simply explains how to install Samba 3 and briefly mentions the SMB protocol.

56.1. verify installed version

.rpm based distributions

To see the version of samba installed on Red Hat, Fedora or CentOS use **rpm -q samba**.

```
[root@RHEL52 ~]# rpm -q samba
samba-3.0.28-1.el5_2.1
```

The screenshot above shows that RHEL5 has **Samba** version 3.0 installed. The last number in the Samba version counts the number of updates or patches.

Below the same command on a more recent version of CentOS with Samba version 3.5 installed.

```
[root@centos6 ~]# rpm -q samba
samba-3.5.10-116.el6_2.i686
```

.deb based distributions

Use **dpkg -l** or **aptitude show** on Debian or Ubuntu. Both Debian 7.0 (Wheezy) and Ubuntu 12.04 (Precise) use version 3.6.3 of the Samba server.

```
root@debian7~# aptitude show samba | grep Version
Version: 2:3.6.3-1
```

Ubuntu 12.04 is currently at Samba version 3.6.3.

```
root@ubul204:~# dpkg -l samba | tail -1
ii samba 2:3.6.3-2ubuntu2.1 SMB/CIFS file, print, and login server for Unix
```

56.2. installing samba

.rpm based distributions

Samba is installed by default on Red Hat Enterprise Linux. If Samba is not yet installed, then you can use the graphical menu (Applications -- System Settings -- Add/Remove Applications) and select "Windows File Server" in the Server section. The non-graphical way is to use **rpm** or **yum**.

When you downloaded the .rpm file, you can install Samba like this.

```
[paul@RHEL52 ~]$ rpm -i samba-3.0.28-1.el5_2.1.rpm
```

When you have a subscription to RHN (Red Hat Network), then **yum** is an easy tool to use. This **yum** command works by default on Fedora and CentOS.

```
[root@centos6 ~]# yum install samba
```

.deb based distributions

Ubuntu and Debian users can use the **aptitude** program (or use a graphical tool like Synaptic).

```
root@debian7~# aptitude install samba
The following NEW packages will be installed:
  samba samba-common{a} samba-common-bin{a} tdb-tools{a}
0 packages upgraded, 4 newly installed, 0 to remove and 1 not upgraded.
Need to get 15.1 MB of archives. After unpacking 42.9 MB will be used.
Do you want to continue? [Y/n/?]
...
```

56.3. documentation

samba howto

Samba comes with excellent documentation in html and pdf format (and also as a free download from samba.org and it is for sale as a printed book).

The documentation is a separate package, so install it if you want it on the server itself.

```
[root@centos6 ~]# yum install samba-doc
...
[root@centos6 ~]# ls -l /usr/share/doc/samba-doc-3.5.10/
total 10916
drwxr-xr-x. 6 root root    4096 May  6 15:50 htmldocs
-rw-r--r--. 1 root root 4605496 Jun 14 2011 Samba3-ByExample.pdf
-rw-r--r--. 1 root root 608260 Jun 14 2011 Samba3-Developers-Guide.pdf
-rw-r--r--. 1 root root 5954602 Jun 14 2011 Samba3-HOWTO.pdf
```

This action is very similar on Ubuntu and Debian except that the pdf files are in a separate package named **samba-doc-pdf**.

```
root@ubul204:~# aptitude install samba-doc-pdf
The following NEW packages will be installed:
  samba-doc-pdf
...
```

samba by example

Besides the howto, there is also an excellent book called **Samba By Example** (again available as printed edition in shops, and as a free pdf and html).

56.4. starting and stopping samba

You can start the daemons by invoking `/etc/init.d/smb start` (some systems use `/etc/init.d/samba`) on any linux.

```
root@laika:~# /etc/init.d/samba stop
* Stopping Samba daemons [ OK ]
root@laika:~# /etc/init.d/samba start
* Starting Samba daemons [ OK ]
root@laika:~# /etc/init.d/samba restart
* Stopping Samba daemons [ OK ]
* Starting Samba daemons [ OK ]
root@laika:~# /etc/init.d/samba status
* SMBD is running [ OK ]
```

Red Hat derived systems are happy with `service smb start`.

```
[root@RHEL4b ~]# /etc/init.d/smb start
Starting SMB services: [ OK ]
Starting NMB services: [ OK ]
[root@RHEL4b ~]# service smb restart
Shutting down SMB services: [ OK ]
Shutting down NMB services: [ OK ]
Starting SMB services: [ OK ]
Starting NMB services: [ OK ]
[root@RHEL4b ~]#
```

56.5. samba daemons

Samba 3 consists of three daemons, they are named **nmbd**, **smbd** and **winbindd**.

nmbd

The **nmbd** daemon takes care of all the names and naming. It registers and resolves names, and handles browsing. According to the Samba documentation, it should be the first daemon to start.

```
[root@RHEL52 ~]# ps -C nmbd
  PID TTY          TIME CMD
 5681 ?            00:00:00 nmbd
```

smbd

The **smbd** daemon manages file transfers and authentication.

```
[root@RHEL52 ~]# ps -C smbd
  PID TTY          TIME CMD
 5678 ?            00:00:00 smbd
 5683 ?            00:00:00 smbd
```

winbindd

The **winbind daemon** (winbindd) is only started to handle Microsoft Windows domain membership.

Note that **winbindd** is started by the `/etc/init.d/winbind` script (two dd's for the daemon and only one d for the script).

```
[root@RHEL52 ~]# /etc/init.d/winbind start
Starting Winbind services:                                [ OK ]
[root@RHEL52 ~]# ps -C winbindd
  PID TTY          TIME CMD
 5752 ?            00:00:00 winbindd
 5754 ?            00:00:00 winbindd
```

On Debian and Ubuntu, the winbindd daemon is installed via a separate package called **winbind**.

56.6. the SMB protocol

brief history

Development of this protocol was started by **IBM** in the early eighties. By the end of the eighties, most development was done by **Microsoft**. SMB is an application level protocol designed to run on top of NetBIOS/NetBEUI, but can also be run on top of tcp/ip.

In 1996 Microsoft was asked to document the protocol. They submitted CIFS (Common Internet File System) as an internet draft, but it never got final rfc status.

In 2004 the European Union decided Microsoft should document the protocol to enable other developers to write compatible software. December 20th 2007 Microsoft came to an agreement. The Samba team now has access to SMB/CIFS, Windows for Workgroups and Active Directory documentation.

broadcasting protocol

SMB uses the NetBIOS **service location protocol**, which is a broadcasting protocol. This means that NetBIOS names have to be unique on the network (even when you have different IP-addresses). Having duplicate names on an SMB network can seriously harm communications.

NetBIOS names

NetBIOS names are similar to **hostnames**, but are always uppercase and only 15 characters in length. Microsoft Windows computers and Samba servers will broadcast this name on the network.

network bandwidth

Having many broadcasting SMB/CIFS computers on your network can cause bandwidth issues. A solution can be the use of a **NetBIOS name server** (NBNS) like **WINS** (Windows Internet Naming Service).

56.7. practice: introduction to samba

0. !! Make sure you know your student number, anything *ANYTHING* you name must include your student number!

1. Verify that you can logon to a Linux/Unix computer. Write down the name and ip address of this computer.

2. Do the same for all the other (virtual) machines available to you.

3. Verify networking by pinging the computer, edit the appropriate hosts files so you can use names. Test the names by pinging them.

4. Make sure Samba is installed, write down the version of Samba.

5. Open the Official Samba-3 howto pdf file that is installed on your computer. How many A4 pages is this file ? Then look at the same pdf on samba.org, it is updated regularly.

6. Stop the Samba server.

Chapter 57. getting started with samba

Table of Contents

57.1. /etc/samba/smb.conf	536
57.2. /usr/bin/testparm	537
57.3. /usr/bin/smbclient	539
57.4. /usr/bin/smbtree	540
57.5. server string	542
57.6. Samba Web Administration Tool (SWAT)	542
57.7. practice: getting started with samba	544
57.8. solution: getting started with samba	545

57.1. /etc/samba/smb.conf

smbd -b

Samba configuration is done in the **smb.conf** file. The file can be edited manually, or you can use a web based interface like webmin or swat to manage it. The file is usually located in /etc/samba. You can find the exact location with **smbd -b**.

```
[root@RHEL4b ~]# smbd -b | grep CONFIGFILE
CONFIGFILE: /etc/samba/smb.conf
```

the default smb.conf

The default smb.conf file contains a lot of examples with explanations.

```
[paul@RHEL4b ~]$ ls -l /etc/samba/smb.conf
-rw-r--r-- 1 root root 10836 May 30 23:08 /etc/samba/smb.conf
```

Also on Ubuntu and Debian, smb.conf is packed with samples and explanations.

```
paul@laika:~$ ls -l /etc/samba/smb.conf
-rw-r--r-- 1 root root 10515 2007-05-24 00:21 /etc/samba/smb.conf
```

minimal smb.conf

Below is an example of a very minimalistic **smb.conf**. It allows samba to start, and to be visible to other computers (Microsoft shows computers in Network Neighborhood or My Network Places).

```
[paul@RHEL4b ~]$ cat /etc/samba/smb.conf
[global]
workgroup = WORKGROUP
[firstshare]
path = /srv/samba/public
```

net view

Below is a screenshot of the **net view** command on Microsoft Windows Server 2003 sp2. It shows how a Red Hat Enterprise Linux 5.3 and a Ubuntu 9.04 Samba server, both with a minimalistic smb.conf, are visible to Microsoft computers nearby.

```
C:\Documents and Settings\Administrator>net view
Server Name          Remark
-----
\\LAIKA              Samba 3.3.2
\\RHEL53              Samba 3.0.33-3.7.e15
\\W2003
The command completed successfully.
```

long lines in smb.conf

Some parameters in smb.conf can get a long list of values behind them. You can continue a line (for clarity) on the next by ending the line with a backslash.

```
valid users = Serena, Venus, Lindsay \  
             Kim, Justine, Sabine \  
             Amelie, Marie, Suzanne
```

curious smb.conf

Curious but true: smb.conf accepts synonyms like **create mode** and **create mask**, and (sometimes) minor spelling errors like **browsable** and **browseable**. And on occasion you can even switch words, the **guest only** parameter is identical to **only guest**. And **writable = yes** is the same as **readonly = no**.

man smb.conf

You can access a lot of documentation when typing **man smb.conf**.

```
[root@RHEL4b samba]# apropos samba  
cupsaddsmb      (8) - export printers to samba for windows clients  
lmhosts        (5) - The Samba NetBIOS hosts file  
net            (8) - Tool for administration of Samba and remote CIFS servers  
pdbedit       (8) - manage the SAM database (Database of Samba Users)  
samba         (7) - A Windows SMB/CIFS fileserver for UNIX  
smb.conf [smb] (5) - The configuration file for the Samba suite  
smbpasswd     (5) - The Samba encrypted password file  
smbstatus     (1) - report on current Samba connections  
swat         (8) - Samba Web Administration Tool  
tdbbackup     (8) - tool for backing up and ... of samba .tdb files  
[root@RHEL4b samba]#
```

57.2. /usr/bin/testparm

syntax check smb.conf

To verify the syntax of the smb.conf file, you can use **testparm**.

```
[paul@RHEL4b ~]$ testparm  
Load smb config files from /etc/samba/smb.conf  
Processing section "[firstshare]"  
Loaded services file OK.  
Server role: ROLE_STANDALONE  
Press enter to see a dump of your service definitions
```

testparm -v

An interesting option is **testparm -v**, which will output all the global options with their default value.

```
[root@RHEL52 ~]# testparm -v | head
Load smb config files from /etc/samba/smb.conf
Processing section "[pub0]"
Processing section "[global$]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions
```

```
[global]
dos charset = CP850
unix charset = UTF-8
display charset = LOCALE
workgroup = WORKGROUP
realm =
netbios name = TEACHER0
netbios aliases =
netbios scope =
server string = Samba 3.0.28-1.el5_2.1
...
```

There were about 350 default values for `smb.conf` parameters in Samba 3.0.x. This number grew to almost 400 in Samba 3.5.x.

testparm -s

The samba daemons are constantly (once every 60 seconds) checking the `smb.conf` file, so it is good practice to keep this file small. But it is also good practice to document your samba configuration, and to explicitly set options that have the same default values. The **testparm -s** option allows you to do both. It will output the smallest possible samba configuration file, while retaining all your settings. The idea is to have your samba configuration in another file (like `smb.conf.full`) and let `testparm` parse this for you. The screenshot below shows you how. First the `smb.conf.full` file with the explicitly set option `workgroup` to `WORKGROUP`.

```
[root@RHEL4b samba]# cat smb.conf.full
[global]
workgroup = WORKGROUP

# This is a demo of a documented smb.conf
# These two lines are removed by testparm -s

server string = Public Test Server

[firstshare]
path = /srv/samba/public
```

Next, we execute `testparm` with the `-s` option, and redirect stdout to the real **smb.conf** file.

```
[root@RHEL4b samba]# testparm -s smb.conf.full > smb.conf
Load smb config files from smb.conf.full
Processing section "[firstshare]"
Loaded services file OK.
```

And below is the end result. The two comment lines and the default option are no longer there.

```
[root@RHEL4b samba]# cat smb.conf
```



```
# Global parameters
[global]
server string = Public Test Server

[firstshare]
path = /srv/samba/public
[root@RHEL4b samba]#
```

57.3. /usr/bin/smbclient

smbclient looking at Samba

With **smbclient** you can see browsing and share information from your smb server. It will display all your shares, your workgroup, and the name of the Master Browser. The **-N** switch is added to avoid having to enter an empty password. The **-L** switch is followed by the name of the host to check.

```
[root@RHEL4b init.d]# smbclient -NL rhel4b
Anonymous login successful
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.0.10-1.4E.9]

Sharename      Type      Comment
-----      -
firstshare     Disk
IPC$           IPC       IPC Service (Public Test Server)
ADMIN$         IPC       IPC Service (Public Test Server)
Anonymous login successful
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.0.10-1.4E.9]

Server          Comment
-----
RHEL4B          Public Test Server
WINXP

Workgroup       Master
-----
WORKGROUP      WINXP
```

smbclient anonymous

The screenshot below uses **smbclient** to display information about a remote smb server (in this case a computer with Ubuntu 11.10).

```
root@ubull110:/etc/samba# testparm smbclient -NL 127.0.0.1
Anonymous login successful
Domain=[LINUXTR] OS=[Unix] Server=[Samba 3.5.11]

Sharename      Type      Comment
-----      -
share1         Disk
IPC$           IPC       IPC Service (Samba 3.5.11)
Anonymous login successful
Domain=[LINUXTR] OS=[Unix] Server=[Samba 3.5.11]

Server          Comment
```

```
-----  
Workgroup           Master  
-----  
LINUXTR            DEBIAN6  
WORKGROUP          UBU1110
```

smbclient with credentials

Windows versions after xp sp2 and 2003 sp1 do not accept guest access (the NT_STATUS_ACCESS_DENIED error). This example shows how to provide credentials with **smbclient**.

```
[paul@RHEL53 ~]$ smbclient -L w2003 -U administrator%stargate  
Domain=[W2003] OS=[Windows Server 2003 3790 Service Pack 2] Server=...
```

```
Sharename          Type      Comment  
-----  
C$                 Disk     Default share  
IPC$               IPC      Remote IPC  
ADMIN$             Disk     Remote Admin  
...
```

57.4. /usr/bin/smbtree

Another useful tool to troubleshoot Samba or simply to browse the SMB network is **smbtree**. In its simplest form, smbtree will do an anonymous browsing on the local subnet, displaying all SMB computers and (if authorized) their shares.

Let's take a look at two screenshots of smbtree in action (with blank password). The first one is taken immediately after booting four different computers (one MS Windows 2000, one MS Windows xp, one MS Windows 2003 and one RHEL 4 with Samba 3.0.10).

```
[paul@RHEL4b ~]$ smbtree  
Password:  
WORKGROUP  
PEGASUS  
  \\WINXP  
  \\RHEL4B                Pegasus Domain Member Server  
Error connecting to 127.0.0.1 (Connection refused)  
cli_full_connection: failed to connect to RHEL4B<20> (127.0.0.1)  
  \\HM2003  
[paul@RHEL4b ~]$
```

The information displayed in the previous screenshot looks incomplete. The browsing elections are still ongoing, the browse list is not yet distributed to all clients by the (to be elected) browser master. The next screenshot was taken about one minute later. And it shows even less.

```
[paul@RHEL4b ~]$ smbtree  
Password:  
WORKGROUP  
  \\W2000  
[paul@RHEL4b ~]$
```

So we wait a while, and then run **smbtree** again, this time it looks a lot nicer.

```
[paul@RHEL4b ~]$ smbtree
Password:
WORKGROUP
  \\W2000
PEGASUS
  \\WINXP
  \\RHEL4B
    Pegasus Domain Member Server
  \\RHEL4B\ADMIN$
    IPC Service (Pegasus Domain Member Server)
  \\RHEL4B\IPC$
    IPC Service (Pegasus Domain Member Server)
  \\RHEL4B\domaindata
    Active Directory users only
  \\HM2003
[paul@RHEL4b ~]$ smbtree --version
Version 3.0.10-1.4E.9
[paul@RHEL4b ~]$
```

I added the version number of **smbtree** in the previous screenshot, to show you the difference when using the latest version of smbtree (below a screenshot taken from Ubuntu Feisty Fawn). The latest version shows a more complete overview of machines and shares.

```
paul@laika:~$ smbtree --version
Version 3.0.24
paul@laika:~$ smbtree
Password:
WORKGROUP
  \\W2000
    \\W2000\firstshare
    \\W2000\C$
      Default share
    \\W2000\ADMIN$
      Remote Admin
    \\W2000\IPC$
      Remote IPC
PEGASUS
  \\WINXP
cli_rpc_pipe_open: cli_nt_create failed on pipe \srvsvc to machine WINXP.
Error was NT_STATUS_ACCESS_DENIED
  \\RHEL4B
    Pegasus Domain Member Server
  \\RHEL4B\ADMIN$
    IPC Service (Pegasus Domain Member Server)
  \\RHEL4B\IPC$
    IPC Service (Pegasus Domain Member Server)
  \\RHEL4B\domaindata
    Active Directory users only
  \\HM2003
cli_rpc_pipe_open: cli_nt_create failed on pipe \srvsvc to machine HM2003.
Error was NT_STATUS_ACCESS_DENIED
paul@laika:~$
```

The previous screenshot also provides useful errors on why we cannot see shared info on computers winxp and w2003. Let us try the old **smbtree** version on our RHEL server, but this time with Administrator credentials (which are the same on all computers).

```
[paul@RHEL4b ~]$ smbtree -UAdministrator%Stargate1
WORKGROUP
  \\W2000
PEGASUS
  \\WINXP
    \\WINXP\C$
      Default share
    \\WINXP\ADMIN$
      Remote Admin
    \\WINXP\share55
    \\WINXP\IPC$
      Remote IPC
  \\RHEL4B
    Pegasus Domain Member Server
  \\RHEL4B\ADMIN$
    IPC Service (Pegasus Domain Member Server)
  \\RHEL4B\IPC$
    IPC Service (Pegasus Domain Member Server)
```

```
    \\RHEL4B\domaindata      Active Directory users only
  \\HM2003
    \\HM2003\NETLOGON       Logon server share
    \\HM2003\SYSVOL        Logon server share
    \\HM2003\WSUSTemp      A network share used by Local Publishing ...
    \\HM2003\ADMIN$        Remote Admin
    \\HM2003\tools
    \\HM2003\IPC$          Remote IPC
    \\HM2003\WsusContent   A network share to be used by Local ...
    \\HM2003\C$            Default share
[paul@RHEL4b ~]$
```

As you can see, this gives a very nice overview of all SMB computers and their shares.

57.5. server string

The comment seen by the **net view** and the **smbclient** commands is the default value for the **server string** option. Simply adding this value to the global section in **smb.conf** and restarting samba will change the option.

```
[root@RHEL53 samba]# testparm -s 2>/dev/null | grep server
server string = Red Hat Server in Paris
```

After a short while, the changed option is visible on the Microsoft computers.

```
C:\Documents and Settings\Administrator>net view
Server Name          Remark
```

```
-----
\\LAIKA              Ubuntu 9.04 server in Antwerp
\\RHEL53             Red Hat Server in Paris
\\W2003
```

57.6. Samba Web Administration Tool (SWAT)

Samba comes with a web based tool to manage your samba configuration file. **SWAT** is accessible with a web browser on port 901 of the host system. To enable the tool, first find out whether your system is using the **inetd** or the **xinetd** superdaemon.

```
[root@RHEL4b samba]# ps fax | grep inet
15026 pts/0    S+   0:00          \_ grep inet
2771  ?        Ss   0:00 xinetd -stayalive -pidfile /var/run/xinetd.pid
[root@RHEL4b samba]#
```

Then edit the **inetd.conf** or change the **disable = yes** line in **/etc/xinetd.d/swat** to **disable = no**.

```
[root@RHEL4b samba]# cat /etc/xinetd.d/swat
# default: off
# description: SWAT is the Samba Web Admin Tool. Use swat \
#              to configure your Samba server. To use SWAT, \
#              connect to port 901 with your favorite web browser.
service swat
{
```

```
port          = 901
socket_type   = stream
wait          = no
only_from     = 127.0.0.1
user          = root
server        = /usr/sbin/swat
log_on_failure += USERID
disable       = no
}
[root@RHEL4b samba]# /etc/init.d/xinetd restart
Stopping xinetd:          [ OK ]
Starting xinetd:         [ OK ]
[root@RHEL4b samba]#
```

Change the **only from** value to enable swat from remote computers. This examples shows how to provide swat access to all computers in a /24 subnet.

```
[root@RHEL53 xinetd.d]# grep only /etc/xinetd.d/swat
only_from = 192.168.1.0/24
```

Be careful when using SWAT, it erases all your manually edited comments in smb.conf.

57.7. practice: getting started with samba

1. Take a backup copy of the original smb.conf, name it smb.conf.orig
2. Enable SWAT and take a look at it.
3. Stop the Samba server.
4. Create a minimalistic smb.conf.minimal and test it with testparm.
5. Use testparm -s to create /etc/samba/smb.conf from your smb.conf.minimal .
6. Start Samba with your minimal smb.conf.
7. Verify with smbclient that your Samba server works.
8. Verify that another (Microsoft) computer can see your Samba server.
9. Browse the network with net view, smbtree and with Windows Explorer.
10. Change the "Server String" parameter in smb.conf. How long does it take before you see the change (net view, smbclient, My Network Places,...) ?
11. Will restarting Samba after a change to smb.conf speed up the change ?
12. Which computer is the master browser master in your workgroup ? What is the master browser ?
13. If time permits (or if you are waiting for other students to finish this practice), then install a sniffer (wireshark) and watch the browser elections.

57.8. solution: getting started with samba

1. Take a backup copy of the original smb.conf, name it smb.conf.orig

```
cd /etc/samba ; cp smb.conf smb.conf.orig
```

2. Enable SWAT and take a look at it.

```
on Debian/Ubuntu: vi /etc/inetd.conf (remove # before swat)
```

```
on RHEL/Fedora: vi /etc/xinetd.d/swat (set disable to no)
```

3. Stop the Samba server.

```
/etc/init.d/smb stop (Red Hat)
```

```
/etc/init.d/samba stop (Debian)
```

4. Create a minimalistic smb.conf.minimal and test it with testparm.

```
cd /etc/samba ; mkdir my_smb_confs ; cd my_smb_confs
```

```
vi smb.conf.minimal
```

```
testparm smb.conf.minimal
```

5. Use testparm -s to create /etc/samba/smb.conf from your smb.conf.minimal .

```
testparm -s smb.conf.minimal > ../smb.conf
```

6. Start Samba with your minimal smb.conf.

```
/etc/init.d/smb restart (Red Hat)
```

```
/etc/init.d/samba restart (Debian)
```

7. Verify with smbclient that your Samba server works.

```
smbclient -NL 127.0.0.1
```

8. Verify that another computer can see your Samba server.

```
smbclient -NL 'ip-address' (on a Linux)
```

9. Browse the network with net view, smbtree and with Windows Explorer.

```
on Linux: smbtree
```

```
on Windows: net view (and WindowsKey + e)
```

10. Change the "Server String" parameter in smb.conf. How long does it take before you see the change (net view, smbclient, My Network Places,...) ?

```
vi /etc/samba/smb.conf
```

```
(should take only seconds when restarting samba)
```

11. Will restarting Samba after a change to smb.conf speed up the change ?

```
yes
```

12. Which computer is the master browser master in your workgroup ? What is the master browser ?

The computer that won the elections.

This machine will make the list of computers in the network

13. If time permits (or if you are waiting for other students to finish this practice), then install a sniffer (wireshark) and watch the browser elections.

On ubuntu: `sudo aptitude install wireshark`

then: `sudo wireshark, select interface`

Chapter 58. a read only file server

Table of Contents

58.1. Setting up a directory to share	548
58.2. configure the share	548
58.3. restart the server	549
58.4. verify the share	549
58.5. a note on netcat	551
58.6. practice: read only file server	552
58.7. solution: read only file server	553

58.1. Setting up a directory to share

Let's start with setting up a very simple read only file server with Samba. Everyone (even anonymous guests) will receive read access.

The first step is to create a directory and put some test files in it.

```
[root@RHEL52 ~]# mkdir -p /srv/samba/readonly
[root@RHEL52 ~]# cd /srv/samba/readonly/
[root@RHEL52 readonly]# echo "It is cold today." > winter.txt
[root@RHEL52 readonly]# echo "It is hot today." > summer.txt
[root@RHEL52 readonly]# ls -l
total 8
-rw-r--r-- 1 root root 17 Jan 21 05:49 summer.txt
-rw-r--r-- 1 root root 18 Jan 21 05:49 winter.txt
[root@RHEL52 readonly]#
```

58.2. configure the share

smb.conf [global] section

In this example the samba server is a member of WORKGROUP (the default workgroup). We also set a descriptive server string, this string is visible to users browsing the network with net view, windows explorer or smbclient.

```
[root@RHEL52 samba]# head -5 smb.conf
[global]
workgroup = WORKGROUP
server string = Public Anonymous File Server
netbios name = TEACHER0
security = share
```

You might have noticed the line with **security = share**. This line sets the default security mode for our samba server. Setting the security mode to **share** will allow clients (smbclient, any windows, another Samba server, ...) to provide a password for each share. This is one way of using the SMB/CIFS protocol. The other way (called **user mode**) will allow the client to provide a username/password combination, before the server knows which share the client wants to access.

smb.conf [share] section

The share is called pubread and the path is set to our newly created directory. Everyone is allowed access (**guest ok = yes**) and security is set to read only.

```
[pubread]
path = /srv/samba/readonly
comment = files to read
read only = yes
guest ok = yes
```

Here is a very similar configuration on Ubuntu 11.10.

```
root@ubull110:~# cat /etc/samba/smb.conf
[global]
workgroup = LINUXTR
netbios name = UBU1110
security = share
[roshare1]
path = /srv/samba/readonly
read only = yes
guest ok = yes
```

It doesn't really matter which Linux distribution you use. Below the same config on Debian 6, as good as identical.

```
root@debian6:~# cat /etc/samba/smb.conf
[global]
workgroup = LINUXTR
netbios name = DEBIAN6
security = share
[roshare1]
path = /srv/samba/readonly
read only = yes
guest ok = yes
```

58.3. restart the server

After testing with **testparm**, restart the samba server (so you don't have to wait).

```
[root@RHEL4b readonly]# service smb restart
Shutting down SMB services:           [ OK ]
Shutting down NMB services:          [ OK ]
Starting SMB services:                [ OK ]
Starting NMB services:                [ OK ]
```

58.4. verify the share

verify with smbclient

You can now verify the existence of the share with **smbclient**. Our **pubread** is listed as the fourth share.

```
[root@RHEL52 samba]# smbclient -NL 127.0.0.1
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.0.33-3.7.e15]

Sharename      Type           Comment
-----
IPC$           IPC           IPC Service (Public Anonymous File Server)
global$       Disk
pub0          Disk
pubread       Disk          files to read
Domain=[WORKGROUP] OS=[Unix] Server=[Samba 3.0.33-3.7.e15]

Server          Comment
-----
TEACHER0       Samba 3.0.33-3.7.e15
W2003EE

Workgroup      Master
-----
```

WORKGROUP

W2003EE

verify on windows

The final test is to go to a Microsoft windows computer and read a file on the Samba server. First we use the **net use** command to mount the pubread share on the driveletter k.

```
C:\>net use K: \\teacher0\pubread
The command completed successfully.
```

Then we test looking at the contents of the share, and reading the files.

```
C:\>dir k:
Volume in drive K is pubread
Volume Serial Number is 0C82-11F2

Directory of K:\

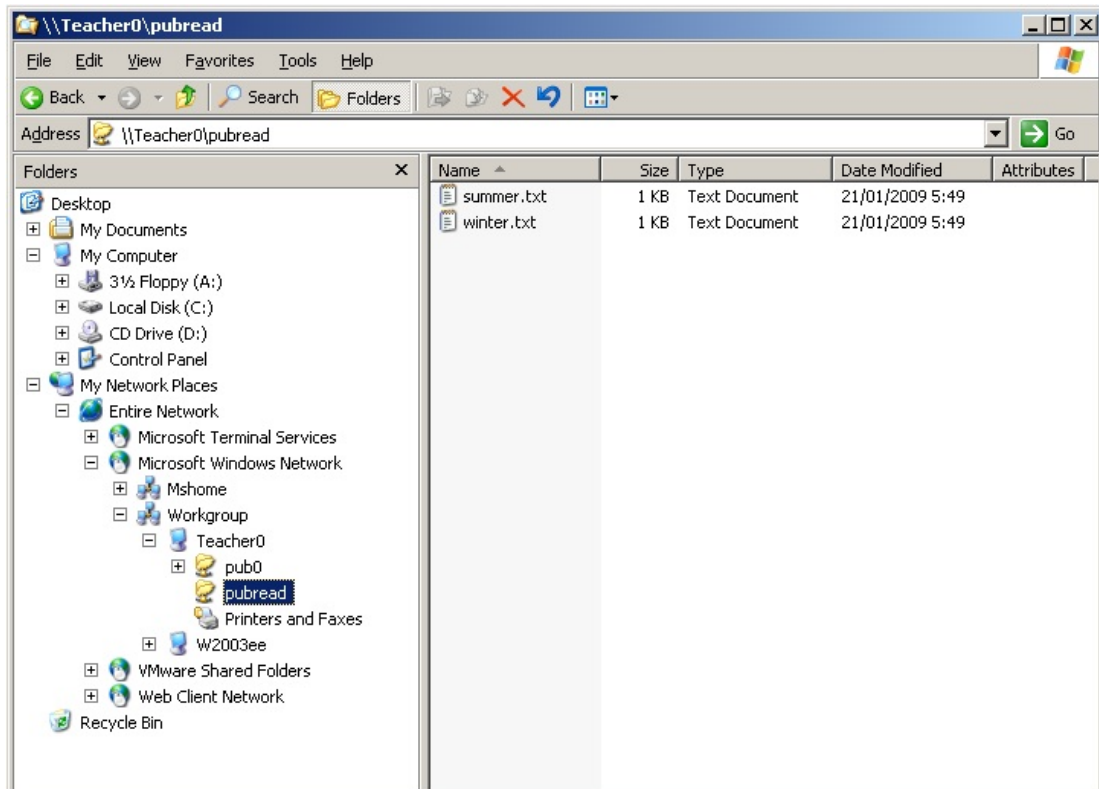
21/01/2009  05:49    <DIR>          .
21/01/2009  05:49    <DIR>          ..
21/01/2009  05:49                   17 summer.txt
21/01/2009  05:49                   18 winter.txt
                2 File(s)              35 bytes
                2 Dir(s)  13.496.242.176 bytes free
```

Just to be on the safe side, let us try writing.

```
K:\>echo very cold > winter.txt
Access is denied.
```

```
K:\>
```

Or you can use windows explorer...



58.5. a note on netcat

The Windows command line screenshot is made in a Linux console, using **netcat** as a pipe to a Windows command shell.

The way this works, is by enabling netcat to listen on the windows computer to a certain port, executing cmd.exe when a connection is received. Netcat is similar to cat, in the way that cat does nothing, only netcat does nothing over the network.

To enable this connection, type the following on the windows computer (after downloading netcat for windows).

```
nc -l -p 23 -t -e cmd.exe
```

And then connect to this machine with netcat from any Linux computer. You end up with a cmd.exe prompt inside your Linux shell.

```
paul@laika:~$ nc 192.168.1.38 23
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\>net use k: /delete
net use k: /delete
k: was deleted successfully.
```

58.6. practice: read only file server

1. Create a directory in a good location (FHS) to share files for everyone to read.
2. Make sure the directory is owned properly and is world accessible.
3. Put a textfile in this directory.
4. Share the directory with Samba.
5. Verify from your own and from another computer (smbclient, net use, ...) that the share is accessible for reading.
6. Make a backup copy of your smb.conf, name it smb.conf.ReadOnlyFileServer.

58.7. solution: read only file server

1. Create a directory in a good location (FHS) to share files for everyone to read.

choose one of these...

```
mkdir -p /srv/samba/readonly
```

```
mkdir -p /home/samba/readonly
```

```
/home/paul/readonly is wrong!!
```

```
/etc/samba/readonly is wrong!!
```

```
/readonly is wrong!!
```

2. Make sure the directory is owned properly and is world accessible.

```
chown root:root /srv/samba/readonly
```

```
chmod 755 /srv/samba/readonly
```

3. Put a textfile in this directory.

```
echo Hello World > hello.txt
```

4. Share the directory with Samba.

You `smb.conf.readonly` could look like this:

```
[global]
workgroup = WORKGROUP
server string = Read Only File Server
netbios name = STUDENTx
security = share
```

```
[readonlyX]
path = /srv/samba/readonly
comment = read only file share
read only = yes
guest ok = yes
```

test with `testparm` before going in production!

5. Verify from your own and from another computer (`smbclient`, `net use`, ...) that the share is accessible for reading.

On Linux: `smbclient -NL 127.0.0.1`

On Windows Explorer: browse to My Network Places

On Windows `cmd.exe`: `net use L: //studentx/readonly`

6. Make a backup copy of your `smb.conf`, name it `smb.conf.ReadOnlyFileServer`.

```
cp smb.conf smb.conf.ReadOnlyFileServer
```

Chapter 59. a writable file server

Table of Contents

59.1. set up a directory to share	555
59.2. share section in smb.conf	555
59.3. configure the share	555
59.4. test connection with windows	555
59.5. test writing with windows	556
59.6. How is this possible ?	556
59.7. practice: writable file server	557
59.8. solution: writable file server	558

59.1. set up a directory to share

In this second example, we will create a share where everyone can create files and write to files. Again, we start by creating a directory

```
[root@RHEL52 samba]# mkdir -p /srv/samba/writable
[root@RHEL52 samba]# chmod 777 /srv/samba/writable/
```

59.2. share section in smb.conf

There are two parameters to make a share writable. We can use **read only** or **writable**. This example shows how to use **writable** to give write access to a share.

```
writable = yes
```

And this is an example of using the **read only** parameter to give write access to a share.

```
read only = no
```

59.3. configure the share

Then we simply add a share to our file server by editing **smb.conf**. Below the check with **testparm**. (We could have changed the description of the server...)

```
[root@RHEL52 samba]# testparm
Load smb config files from /etc/samba/smb.conf
Processing section "[pubwrite]"
Processing section "[pubread]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions

[global]
netbios name = TEACHER0
server string = Public Anonymous File Server
security = SHARE

[pubwrite]
comment = files to write
path = /srv/samba/writable
read only = No
guest ok = Yes

[pubread]
comment = files to read
path = /srv/samba/readonly
guest ok = Yes
```

59.4. test connection with windows

We can now test the connection on a windows 2003 computer. We use the **net use** for this.

```
C:\>net use L: \\teacher0\pubwrite
net use L: \\teacher0\pubwrite
The command completed successfully.
```

59.5. test writing with windows

We mounted the **pubwrite** share on the L: drive in windows. Below we test that we can write to this share.

```
L:\>echo hoi > hoi.txt

L:\>dir
Volume in drive L is pubwrite
Volume Serial Number is 0C82-272A

Directory of L:\

21/01/2009  06:11    <DIR>          .
21/01/2009  06:11    <DIR>          ..
21/01/2009  06:16                6 hoi.txt
               1 File(s)                6 bytes
               2 Dir(s)  13.496.238.080 bytes free
```

59.6. How is this possible ?

Linux (or any Unix) always needs a user account to gain access to a system. The windows computer did not provide the samba server with a user account or a password. Instead, the Linux owner of the files created through this writable share is the Linux guest account (usually named nobody).

```
[root@RHEL52 samba]# ls -l /srv/samba/writable/
total 4
-rwxr--r-- 1 nobody nobody 6 Jan 21 06:16 hoi.txt
```

So this is not the cleanest solution. We will need to improve this.

59.7. practice: writable file server

1. Create a directory and share it with Samba.
2. Make sure everyone can read and write files, test writing with smbclient and from a Microsoft computer.
3. Verify the ownership of files created by (various) users.

59.8. solution: writable file server

1. Create a directory and share it with Samba.

```
mkdir /srv/samba/writable
chmod 777 /srv/samba/writable
```

the share section in smb.conf can look like this:

```
[pubwrite]
path = /srv/samba/writable
comment = files to write
read only = no
guest ok = yes
```

2. Make sure everyone can read and write files, test writing with smbclient and from a Microsoft computer.

to test writing with smbclient:

```
echo one > count.txt
echo two >> count.txt
echo three >> count.txt
smbclient //localhost/pubwrite
Password:
smb: \> put count.txt
```

3. Verify the ownership of files created by (various) users.

```
ls -l /srv/samba/writable
```

Chapter 60. samba first user account

Table of Contents

60.1. creating a samba user	560
60.2. ownership of files	560
60.3. /usr/bin/smbpasswd	560
60.4. /etc/samba/smbpasswd	560
60.5. passdb backend	561
60.6. forcing this user	561
60.7. practice: first samba user account	562
60.8. solution: first samba user account	563

60.1. creating a samba user

We will create a user for our samba file server and make this user the owner of the directory and all of its files. This anonymous user gets a clear description, but does not get a login shell.

```
[root@RHEL52 samba]# useradd -s /bin/false sambanobody
[root@RHEL52 samba]# usermod -c "Anonymous Samba Access" sambanobody
[root@RHEL52 samba]# passwd sambanobody
Changing password for user sambanobody.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

60.2. ownership of files

We can use this user as owner of files and directories, instead of using the root account. This approach is clear and more secure.

```
[root@RHEL52 samba]# chown -R sambanobody:sambanobody /srv/samba/
[root@RHEL52 samba]# ls -al /srv/samba/writable/
total 12
drwxrwxrwx 2 sambanobody sambanobody 4096 Jan 21 06:11 .
drwxr-xr-x 6 sambanobody sambanobody 4096 Jan 21 06:11 ..
-rwxr--r-- 1 sambanobody sambanobody    6 Jan 21 06:16 hoi.txt
```

60.3. /usr/bin/smbpasswd

The sambanobody user account that we created in the previous examples is not yet used by samba. It just owns the files and directories that we created for our shares. The goal of this section is to force ownership of files created through the samba share to belong to our sambanobody user. Remember, our server is still accessible to everyone, nobody needs to know this user account or password. We just want a clean Linux server.

To accomplish this, we first have to tell Samba about this user. We can do this by adding the account to **smbpasswd**.

```
[root@RHEL52 samba]# smbpasswd -a sambanobody
New SMB password:
Retype new SMB password:
Added user sambanobody.
```

60.4. /etc/samba/smbpasswd

To find out where Samba keeps this information (for now), use **smbd -b**. The **PRIVATE_DIR** variable will show you where the smbpasswd database is located.

```
[root@RHEL52 samba]# smbd -b | grep PRIVATE
PRIVATE_DIR: /etc/samba
[root@RHEL52 samba]# ls -l smbpasswd
```

```
-rw----- 1 root root 110 Jan 21 06:19 smbpasswd
```

You can use a simple `cat` to see the contents of the **smbpasswd** database. The **sambanobody** user does have a password (it is secret).

```
[root@RHEL52 samba]# cat smbpasswd
sambanobody:503:AE9 ... 9DB309C528E540978:[U ]:LCT-4976B05B:
```

60.5. passdb backend

Note that recent versions of Samba have **tdbsam** as default for the **passdb backend** parameter.

```
root@ubull110:~# testparm -v 2>/dev/null| grep 'passdb backend'

passdb backend = tdbsam
```

60.6. forcing this user

Now that Samba knows about this user, we can adjust our writable share to force the ownership of files created through it. For this we use the **force user** and **force group** options. Now we can be sure that all files in the Samba writable share are owned by the same **sambanobody** user.

Below is the renewed definition of our share in `smb.conf`.

```
[pubwrite]
path = /srv/samba/writable
comment = files to write
force user = sambanobody
force group = sambanobody
read only = no
guest ok = yes
```

When you reconnect to the share and write a file, then this **sambanobody** user will own the newly created file (and nobody needs to know the password).

60.7. practice: first samba user account

1. Create a user account for use with samba.
2. Add this user to samba's user database.
3. Create a writable shared directory and use the "force user" and "force group" directives to force ownership of files.
4. Test the working of force user with smbclient, net use and Windows Explorer.

60.8. solution: first samba user account

1. Create a user account for use with samba.

```
useradd -s /bin/false smbguest
```

```
usermod -c 'samba guest'
```

```
passwd smbguest
```

2. Add this user to samba's user database.

```
smbpasswd -a smbguest
```

3. Create a writable shared directory and use the "force user" and "force group" directives to force ownership of files.

```
[userwrite]
path = /srv/samba/userwrite
comment = everyone writes files owned by smbguest
read only = no
guest ok = yes
force user = smbguest
force group = smbguest
```

4. Test the working of force user with smbclient, net use and Windows Explorer.

```
ls -l /srv/samba/userwrite (and verify ownership)
```

Chapter 61. samba authentication

Table of Contents

61.1. creating the users on Linux	565
61.2. creating the users on samba	565
61.3. security = user	565
61.4. configuring the share	566
61.5. testing access with net use	566
61.6. testing access with smbclient	566
61.7. verify ownership	567
61.8. common problems	567
61.9. practice : samba authentication	569
61.10. solution: samba authentication	570

61.1. creating the users on Linux

The goal of this example is to set up a file share accessible to a number of different users. The users will need to authenticate with their password before access to this share is granted. We will first create three randomly named users, each with their own password. First we add these users to Linux.

```
[root@RHEL52 ~]# useradd -c "Serena Williams" serena
[root@RHEL52 ~]# useradd -c "Justine Henin" justine
[root@RHEL52 ~]# useradd -c "Martina Hingis" martina
[root@RHEL52 ~]# passwd serena
Changing password for user serena.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@RHEL52 ~]# passwd justine
Changing password for user justine.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@RHEL52 ~]# passwd martina
Changing password for user martina.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

61.2. creating the users on samba

Then we add them to the `smbpasswd` file, with the same password.

```
[root@RHEL52 ~]# smbpasswd -a serena
New SMB password:
Retype new SMB password:
Added user serena.
[root@RHEL52 ~]# smbpasswd -a justine
New SMB password:
Retype new SMB password:
Added user justine.
[root@RHEL52 ~]# smbpasswd -a martina
New SMB password:
Retype new SMB password:
Added user martina.
```

61.3. security = user

Remember that we set samba's security mode to share with the `security = share` directive in the `[global]` section ? Since we now require users to always provide a userid and password for access to our samba server, we will need to change this. Setting `security = user` will require the client to provide samba with a valid userid and password before giving access to a share.

Our [global] section now looks like this.

```
[global]
workgroup = WORKGROUP
netbios name = TEACHER0
server string = Samba File Server
security = user
```

61.4. configuring the share

We add the following [share] section to our smb.conf (and we do not forget to create the directory /srv/samba/authwrite).

```
[authwrite]
path = /srv/samba/authwrite
comment = authenticated users only
read only = no
guest ok = no
```

61.5. testing access with net use

After restarting samba, we test with different users from within Microsoft computers. The screenshots use the **net use** First serena from Windows XP.

```
C:\>net use m: \\teacher0\authwrite stargate /user:serena
The command completed successfully.
```

```
C:\>m:
```

```
M:\>echo greetings from Serena > serena.txt
```

The next screenshot is martina on a Windows 2000 computer, she succeeds in writing her files, but fails to overwrite the file from serena.

```
C:\>net use k: \\teacher0\authwrite stargate /user:martina
The command completed successfully.
```

```
C:\>k:
```

```
K:\>echo greetings from martina > Martina.txt
```

```
K:\>echo test overwrite > serena.txt
Access is denied.
```

61.6. testing access with smbclient

You can also test connecting with authentication with **smbclient**. First we test with a wrong password.

```
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U martina wrongpass
session setup failed: NT_STATUS_LOGON_FAILURE
```

Then we test with the correct password, and verify that we can access a file on the share.

```
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U martina stargate
Domain=[TEACHER0] OS=[Unix] Server=[Samba 3.0.33-3.7.el5]
smb: \> more serena.txt
getting file \serena.txt of size 14 as /tmp/smbmore.QQfmSN (6.8 kb/s)
one
two
three
smb: \> q
```

61.7. verify ownership

We now have a simple standalone samba file server with authenticated access. And the files in the shares belong to their proper owners.

```
[root@RHEL52 samba]# ls -l /srv/samba/authwrite/
total 8
-rwxr--r-- 1 martina martina  0 Jan 21 20:06 martina.txt
-rwxr--r-- 1 serena  serena  14 Jan 21 20:06 serena.txt
-rwxr--r-- 1 serena  serena   6 Jan 21 20:09 ser.txt
```

61.8. common problems

NT_STATUS_BAD_NETWORK_NAME

You can get `NT_STATUS_BAD_NETWORK_NAME` when you forget to create the target directory.

```
[root@RHEL52 samba]# rm -rf /srv/samba/authwrite/
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U martina stargate
Domain=[TEACHER0] OS=[Unix] Server=[Samba 3.0.33-3.7.el5]
tree connect failed: NT_STATUS_BAD_NETWORK_NAME
```

NT_STATUS_LOGON_FAILURE

You can get `NT_STATUS_LOGON_FAILURE` when you type the wrong password or when you type an unexisting username.

```
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U martina STARGATE
```

```
session setup failed: NT_STATUS_LOGON_FAILURE
```

usernames are (not) case sensitive

Remember that usernames on Linux are case sensitive.

```
[root@RHEL52 samba]# su - MARTINA
su: user MARTINA does not exist
[root@RHEL52 samba]# su - martina
[martina@RHEL52 ~]$
```

But usernames on Microsoft computers are not case sensitive.

```
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U martina stargate
Domain=[TEACHER0] OS=[Unix] Server=[Samba 3.0.33-3.7.el5]
smb: \> q
[root@RHEL52 samba]# smbclient //teacher0/authwrite -U MARTINA stargate
Domain=[TEACHER0] OS=[Unix] Server=[Samba 3.0.33-3.7.el5]
smb: \> q
```

61.9. practice : samba authentication

0. Make sure you have properly named backups of your smb.conf of the previous practices.
1. Create three users (on the Linux and on the samba), remember their passwords!
2. Set up a shared directory that is only accessible to authenticated users.
3. Use smbclient and a windows computer to access your share, use more than one user account (windows requires a logoff/logon for this).
4. Verify that files created by these users belong to them.
5. Try to change or delete a file from another user.

61.10. solution: samba authentication

1. Create three users (on the Linux and on the samba), remember their passwords!

```
useradd -c 'SMB user1' userx  
passwd userx
```

2. Set up a shared directory that is only accessible to authenticated users.

The shared section in smb.conf could look like this:

```
[authwrite]  
path = /srv/samba/authwrite  
comment = authenticated users only  
read only = no  
guest ok = no
```

3. Use smbclient and a windows computer to access your share, use more than one user account (windows requires a logoff/logon for this).

```
on Linux: smbclient //studentX/authwrite -U user1 password  
on windows net use p: \\studentX\authwrite password /user:user2
```

4. Verify that files created by these users belong to them.

```
ls -l /srv/samba/authwrite
```

5. Try to change or delete a file from another user.

you should not be able to change or overwrite files from others.

Chapter 62. samba securing shares

Table of Contents

62.1. security based on user name	572
62.2. security based on ip-address	573
62.3. security through obscurity	573
62.4. file system security	574
62.5. practice: securing shares	576
62.6. solution: securing shares	577

62.1. security based on user name

valid users

To restrict users per share, you can use the **valid users** parameter. In the example below, only the users listed as valid will be able to access the tennis share.

```
[tennis]
path = /srv/samba/tennis
comment = authenticated and valid users only
read only = No
guest ok = No
valid users = serena, kim, venus, justine
```

invalid users

If you are paranoia, you can also use **invalid users** to explicitly deny the listed users access. When a user is in both lists, the user has no access!

```
[tennis]
path = /srv/samba/tennis
read only = No
guest ok = No
valid users = kim, serena, venus, justine
invalid users = venus
```

read list

On a writable share, you can set a list of read only users with the **read list** parameter.

```
[football]
path = /srv/samba/football
read only = No
guest ok = No
read list = martina, roberto
```

write list

Even on a read only share, you can set a list of users that can write. Use the **write list** parameter.

```
[football]
path = /srv/samba/golf
read only = Yes
guest ok = No
write list = eddy, jan
```

62.2. security based on ip-address

hosts allow

The **hosts allow** or **allow hosts** parameter is one of the key advantages of Samba. It allows access control of shares on the ip-address level. To allow only specific hosts to access a share, list the hosts, separated by comma's.

```
allow hosts = 192.168.1.5, 192.168.1.40
```

Allowing entire subnets is done by ending the range with a dot.

```
allow hosts = 192.168.1.
```

Subnet masks can be added in the classical way.

```
allow hosts = 10.0.0.0/255.0.0.0
```

You can also allow an entire subnet with exceptions.

```
hosts allow = 10. except 10.0.0.12
```

hosts deny

The **hosts deny** or **deny hosts** parameter is the logical counterpart of the previous. The syntax is the same as for hosts allow.

```
hosts deny = 192.168.1.55, 192.168.1.56
```

62.3. security through obscurity

hide unreadable

Setting **hide unreadable** to yes will prevent users from seeing files that cannot be read by them.

```
hide unreadable = yes
```

browsable

Setting the **browseable = no** directive will hide shares from My Network Places. But it will not prevent someone from accessing the share (when the name of the share is known).

Note that **browsable** and **browseable** are both correct syntax.

```
[pubread]
path = /srv/samba/readonly
comment = files to read
read only = yes
guest ok = yes
browseable = no
```

62.4. file system security

create mask

You can use **create mask** and **directory mask** to set the maximum allowed permissions for newly created files and directories. The mask you set is an AND mask (it takes permissions away).

```
[tennis]
path = /srv/samba/tennis
read only = No
guest ok = No
create mask = 640
directory mask = 750
```

force create mode

Similar to **create mask**, but different. Where the mask from above was a logical AND, the mode you set here is a logical OR (so it adds permissions). You can use the **force create mode** and **force directory mode** to set the minimal required permissions for newly created files and directories.

```
[tennis]
path = /srv/samba/tennis
read only = No
guest ok = No
force create mode = 444
force directory mode = 550
```

security mask

The **security mask** and **directory security mask** work in the same way as **create mask** and **directory mask**, but apply only when a windows user is changing permissions using the windows security dialog box.

force security mode

The **force security mode** and **force directory security mode** work in the same way as **force create mode** and **force directory mode**, but apply only when a windows user is changing permissions using the windows security dialog box.

inherit permissions

With **inherit permissions = yes** you can force newly created files and directories to inherit permissions from their parent directory, overriding the create mask and directory mask settings.

```
[authwrite]
path = /srv/samba/authwrite
comment = authenticated users only
read only = no
guest ok = no
create mask = 600
directory mask = 555
inherit permissions = yes
```

62.5. practice: securing shares

1. Create a writable share called sales, and a readonly share called budget. Test that it works.
2. Limit access to the sales share to ann, sandra and veronique.
3. Make sure that roberto cannot access the sales share.
4. Even though the sales share is writable, ann should only have read access.
5. Even though the budget share is read only, sandra should also have write access.
6. Limit one shared directory to the 192.168.1.0/24 subnet, and another share to the two computers with ip-addresses 192.168.1.33 and 172.17.18.19.
7. Make sure the computer with ip 192.168.1.203 cannot access the budget share.
8. Make sure (on the budget share) that users can see only files and directories to which they have access.
9. Make sure the sales share is not visible when browsing the network.
10. All files created in the sales share should have 640 permissions or less.
11. All directories created in the budget share should have 750 permissions or more.
12. Permissions for files on the sales share should never be set more than 664.
13. Permissions for files on the budget share should never be set less than 500.
14. If time permits (or if you are waiting for other students to finish this practice), then combine the "read only" and "writable" statements to check which one has priority.
15. If time permits then combine "read list", "write list", "hosts allow" and "hosts deny". Which of these has priority ?

62.6. solution: securing shares

1. Create a writable share called sales, and a readonly share called budget. Test that it works.

see previous solutions on how to do this...

2. Limit access to the sales share to ann, sandra and veronique.

```
valid users = ann, sandra, veronique
```

3. Make sure that roberto cannot access the sales share.

```
invalid users = roberto
```

4. Even though the sales share is writable, ann should only have read access.

```
read list = ann
```

5. Even though the budget share is read only, sandra should also have write access.

```
write list = sandra
```

6. Limit one shared directory to the 192.168.1.0/24 subnet, and another share to the two computers with ip-addresses 192.168.1.33 and 172.17.18.19.

```
hosts allow = 192.168.1.
```

```
hosts allow = 192.168.1.33, 172.17.18.19
```

7. Make sure the computer with ip 192.168.1.203 cannot access the budget share.

```
hosts deny = 192.168.1.203
```

8. Make sure (on the budget share) that users can see only files and directories to which they have access.

```
hide unreadable = yes
```

9. Make sure the sales share is not visible when browsing the network.

```
browsable = no
```

10. All files created in the sales share should have 640 permissions or less.

```
create mask = 640
```

11. All directories created in the budget share should have 750 permissions or more.

```
force directory mode = 750
```

12. Permissions for files on the sales share should never be set more than 664.

```
security mask = 750
```

13. Permissions for files on the budget share should never be set less than 500.

```
force security directory mask = 500
```

14. If time permits (or if you are waiting for other students to finish this practice), then combine the "read only" and "writable" statements to check which one has priority.

15. If time permits then combine "read list", "write list", "hosts allow" and "hosts deny". Which of these has priority ?

Chapter 63. samba domain member

Table of Contents

63.1. changes in smb.conf	580
63.2. joining an Active Directory domain	581
63.3. winbind	582
63.4. wbinfo	583
63.5. getent	584
63.6. file ownership	584
63.7. practice : samba domain member	585

63.1. changes in smb.conf

workgroup

The **workgroup** option in the global section should match the netbios name of the Active Directory domain.

```
workgroup = STARGATE
```

security mode

Authentication will not be handled by samba now, but by the Active Directory domain controllers, so we set the **security** option to domain.

```
security = Domain
```

Linux uid's

Linux requires a user account for every user accessing its file system, we need to provide Samba with a range of uid's and gid's that it can use to create these user accounts. The range is determined with the **idmap uid** and the **idmap gid** parameters. The first Active Directory user to connect will receive Linux uid 20000.

```
idmap uid = 20000-22000  
idmap gid = 20000-22000
```

winbind use default domain

The **winbind use default domain** parameter makes sure winbind also operates on users without a domain component in their name.

```
winbind use default domain = yes
```

[global] section in smb.conf

Below is our new global section in **smb.conf**.

```
[global]
workgroup = STARGATE
security = Domain
server string = Stargate Domain Member Server
idmap uid = 20000-22000
idmap gid = 20000-22000
winbind use default domain = yes
```

realm in /etc/krb5.conf

To connect to a Windows 2003 sp2 (or later) you will need to adjust the kerberos realm in **/etc/krb5.conf** and set both lookup statements to true.

```
[libdefaults]
default_realm = STARGATE.LOCAL
dns_lookup_realm = true
dns_lookup_kdc = true
```

[share] section in smb.conf

Nothing special is required for the share section in smb.conf. Remember that we do not manually create users in smbpasswd or on the Linux (/etc/passwd). Only Active Directory users are allowed access.

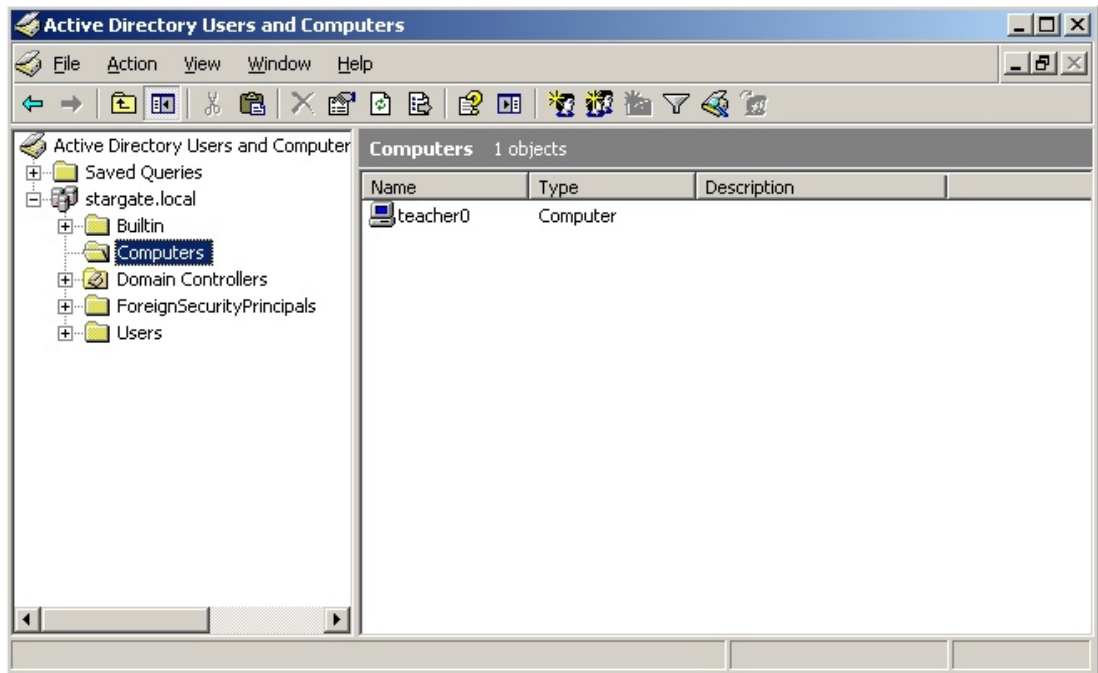
```
[domaingroup]
path = /srv/samba/domaingroup
comment = Active Directory users only
read only = No
```

63.2. joining an Active Directory domain

While the Samba server is stopped, you can use **net rpc join** to join the Active Directory domain.

```
[root@RHEL52 samba]# service smb stop
Shutting down SMB services:          [ OK ]
Shutting down NMB services:         [ OK ]
[root@RHEL52 samba]# net rpc join -U Administrator
Password:
Joined domain STARGATE.
```

We can verify in the aduc (Active Directory Users and Computers) that a computer account is created for this samba server.



63.3. winbind

adding winbind to nsswitch.conf

The **winbind daemon** is talking with the Active Directory domain.

We need to update the `/etc/nsswitch.conf` file now, so user group and host names can be resolved against the winbind daemon.

```
[root@RHEL52 samba]# vi /etc/nsswitch.conf
[root@RHEL52 samba]# grep winbind /etc/nsswitch.conf
passwd:      files winbind
group:       files winbind
hosts:       files dns winbind
```

starting samba and winbindd

Time to start Samba followed by **winbindd**.

```
[root@RHEL4b samba]# service smb start
Starting SMB services: [ OK ]
Starting NMB services: [ OK ]
[root@RHEL4b samba]# service winbind start
Starting winbindd services: [ OK ]
[root@RHEL4b samba]#
```

63.4. wbinfo

verify the trust

You can use **wbinfo -t** to verify the trust between your samba server and Active Directory.

```
[root@RHEL52 ~]# wbinfo -t
checking the trust secret via RPC calls succeeded
```

list all users

We can obtain a list of all user with the **wbinfo -u** command. The domain is not shown when the **winbind use default domain** parameter is set.

```
[root@RHEL52 ~]# wbinfo -u
TEACHER0\serena
TEACHER0\justine
TEACHER0\martina
STARGATE\administrator
STARGATE\guest
STARGATE\support_388945a0
STARGATE\pol
STARGATE\krbtgt
STARGATE\arthur
STARGATE\harry
```

list all groups

We can obtain a list of all domain groups with the **wbinfo -g** command. The domain is not shown when the **winbind use default domain** parameter is set.

```
[root@RHEL52 ~]# wbinfo -g
BUILTIN\administrators
BUILTIN\users
BATMAN\domain computers
BATMAN\domain controllers
BATMAN\schema admins
BATMAN\enterprise admins
BATMAN\domain admins
BATMAN\domain users
BATMAN\domain guests
BATMAN\group policy creator owners
BATMAN\dnsupdateproxy
```

query a user

We can use **wbinfo -a** to verify authentication of a user against Active Directory. Assuming a user account **harry** with password **stargate** is just created on the Active Directory, we get the following screenshot.

```
[root@RHEL52 ~]# wbinfo -a harry%stargate
plaintext password authentication succeeded
challenge/response password authentication succeeded
```

63.5. getent

We can use **getent** to verify that winbindd is working and actually adding the Active directory users to `/etc/passwd`.

```
[root@RHEL52 ~]# getent passwd harry
harry:*:20000:20008:harry potter:/home/BATMAN/harry:/bin/false
[root@RHEL52 ~]# getent passwd arthur
arthur:*:20001:20008:arthur dent:/home/BATMAN/arthur:/bin/false
[root@RHEL52 ~]# getent passwd bilbo
bilbo:*:20002:20008:bilbo baggins:/home/BATMAN/bilbo:/bin/false
```

If the user already exists locally, then the local user account is shown. This is because winbind is configured in `/etc/nsswitch.conf` after **files**.

```
[root@RHEL52 ~]# getent passwd paul
paul:x:500:500:Paul Cobbaut:/home/paul:/bin/bash
```

All the Active Directory users can now easily connect to the Samba share. Files created by them, belong to them.

63.6. file ownership

```
[root@RHEL4b samba]# ll /srv/samba/domaindata/
total 0
-rwxr--r-- 1 justine 20000 0 Jun 22 19:54 create_by_justine_on_winxp.txt
-rwxr--r-- 1 venus   20000 0 Jun 22 19:55 create_by_venus.txt
-rwxr--r-- 1 maria   20000 0 Jun 22 19:57 Maria.txt
```

63.7. practice : samba domain member

1. Verify that you have a working Active Directory (AD) domain.
2. Add the domain name and domain controller to `/etc/hosts`. Set the AD-DNS in `/etc/resolv.conf`.
3. Setup Samba as a member server in the domain.
4. Verify the creation of a computer account in AD for your Samba server.
5. Verify the automatic creation of AD users in `/etc/passwd` with `wbinfo` and `getent`.
6. Connect to Samba shares with AD users, and verify ownership of their files.

Chapter 64. samba domain controller

Table of Contents

64.1. about Domain Controllers	587
64.2. About security modes	587
64.3. About password backends	588
64.4. [global] section in smb.conf	588
64.5. netlogon share	589
64.6. other [share] sections	590
64.7. Users and Groups	590
64.8. tdbsam	591
64.9. about computer accounts	591
64.10. local or roaming profiles	592
64.11. Groups in NTFS acls	593
64.12. logon scripts	594
64.13. practice: samba domain controller	595

64.1. about Domain Controllers

Windows NT4

Windows NT4 works with single master replication domain controllers. There is exactly one PDC (Primary Domain Controller) in the domain, and zero or more BDC's (Backup Domain Controllers). Samba 3 has all features found in Windows NT4 PDC and BDC, and more. This includes file and print serving, domain control with single logon, logon scripts, home directories and roaming profiles.

Windows 200x

With Windows 2000 came Active Directory. AD includes multimaster replication and group policies. Samba 3 can only be a member server in Active Directory, it cannot manage group policies. Samba 4 can do this (in beta).

Samba 3

Samba 3 can act as a domain controller in its own domain. In a Windows NT4 domain, with one Windows NT4 PDC and zero or more BDC's, Samba 3 can only be a member server. The same is valid for Samba 3 in an Active Directory Domain. In short, a Samba 3 domain controller can not share domain control with Windows domain controllers.

Samba 4

Samba 4 can be a domain controller in an Active Directory domain, including managing group policies. As of this writing, Samba 4 is not released for production!

64.2. About security modes

security = share

The 'Windows for Workgroups' way of working, a client requests connection to a share and provides a password for that connection. Anyone who knows a password for a share can access that share. This security model was common in Windows 3.11, Windows 95, Windows 98 and Windows ME.

security = user

The client will send a userid + password before the server knows which share the client wants to access. This mode should be used whenever the samba server is in control of the user database. Both for standalone and samba domain controllers.

security = domain

This mode will allow samba to verify user credentials using NTLM in Windows NT4 and in all Active Directory domains. This is similar to Windows NT4 BDC's joining a native Windows 2000/3 Active Directory domain.

security = ads

This mode will make samba use Kerberos to connect to the Active Directory domain.

security = server

This mode is obsolete, it can be used to forward authentication to another server.

64.3. About password backends

The previous chapters all used the **smbpasswd** user database. For domain control we opt for the **tdbsam** password backend. Another option would be to use LDAP. Larger domains will benefit from using LDAP instead of the not so scalable tdbsam. When you need more than one Domain Controller, then the Samba team advises to not use tdbsam.

64.4. [global] section in smb.conf

Now is a good time to start adding comments in your smb.conf. First we will take a look at the naming of our domain and server in the **[global]** section, and at the domain controlling parameters.

security

The security must be set to user (which is the default). This mode will make samba control the user accounts, so it will allow samba to act as a domain controller.

```
security = user
```

os level

A samba server is the most stable computer in the network, so it should win all browser elections (**os level** above 32) to become the **browser master**

```
os level = 33
```

passdb backend

The **passdb backend** parameter will determine whether samba uses **smbpasswd**, **tdbsam** or **ldap**.

```
passdb backend = tdbsam
```

preferred master

Setting the **preferred master** parameter to yes will make the nmbd daemon force an election on startup.

```
preferred master = yes
```

domain logons

Setting the **domain logons** parameter will make this samba server a domain controller.

```
domain logons = yes
```

domain master

Setting the **domain master** parameter can cause samba to claim the **domain master browser** role for its workgroup. Don't use this parameter in a workgroup with an active NT4 PDC.

```
domain master = yes
```

[global] section

The screenshot below shows a sample [global] section for a samba domain controller.

```
[global]
# names
workgroup = SPORTS
netbios name = DCSPORTS
server string = Sports Domain Controller
# domain control parameters
security = user
os level = 33
preferred master = Yes
domain master = Yes
domain logons = Yes
```

64.5. netlogon share

Part of the microsoft definition for a domain controller is that it should have a **netlogon share**. This is the relevant part of smb.conf to create this netlogon share on Samba.

```
[netlogon]
comment = Network Logon Service
path = /srv/samba/netlogon
admin users = root
guest ok = Yes
browseable = No
```

64.6. other [share] sections

We create some sections for file shares, to test the samba server. Users can all access the general sports file share, but only group members can access their own sports share.

```
[sports]
comment = Information about all sports
path = /srv/samba/sports
valid users = @ntsports
read only = No

[tennis]
comment = Information about tennis
path = /srv/samba/tennis
valid users = @nttennis
read only = No

[football]
comment = Information about football
path = /srv/samba/football
valid users = @ntfootball
read only = No
```

64.7. Users and Groups

To be able to use users and groups in the samba domain controller, we can first set up some groups on the Linux computer.

```
[root@RHEL52 samba]# groupadd ntadmins
[root@RHEL52 samba]# groupadd ntsports
[root@RHEL52 samba]# groupadd ntfootball
[root@RHEL52 samba]# groupadd nttennis
```

This enables us to add group membership info to some new users for our samba domain. Don't forget to give them a password.

```
[root@RHEL52 samba]# useradd -m -G ntadmins Administrator
[root@RHEL52 samba]# useradd -m -G ntsports,nttennis venus
[root@RHEL52 samba]# useradd -m -G ntsports,nttennis kim
[root@RHEL52 samba]# useradd -m -G ntsports,nttennis jelena
[root@RHEL52 samba]# useradd -m -G ntsports,ntfootball figo
```

```
[root@RHEL52 samba]# useradd -m -G ntsports,ntfootball ronaldo
[root@RHEL52 samba]# useradd -m -G ntsports,ntfootball pfaff
```

It is always safe to verify creation of users, groups and passwords in `/etc/passwd`, `/etc/shadow` and `/etc/group`.

```
[root@RHEL52 samba]# tail -11 /etc/group
ntadmins:x:507:Administrator
ntsports:x:508:venus,kim,jelena,figo,ronaldo,pfaff
ntfootball:x:509:figo,ronaldo,pfaff
nttennis:x:510:venus,kim,jelena
Administrator:x:511:
venus:x:512:
kim:x:513:
jelena:x:514:
figo:x:515:
ronaldo:x:516:
pfaff:x:517:
```

64.8. tdbsam

Next we must make these users known to samba with the `smbpasswd` tool. When you add the first user to **tdbsam**, the file `/etc/samba/passdb.tdb` will be created.

```
[root@RHEL52 samba]# smbpasswd -a root
New SMB password:
Retype new SMB password:
tdbsam_open: Converting version 0 database to version 3.
Added user root.
```

Adding all the other users generates less output, because `tdbsam` is already created.

```
[root@RHEL4b samba]# smbpasswd -a root
New SMB password:
Retype new SMB password:
Added user root.
```

64.9. about computer accounts

Every NT computer (Windows NT, 2000, XP, Vista) can become a member of a domain. Joining the domain (by right-clicking on My Computer) means that a computer account will be created in the domain. This computer account also has a password (but you cannot know it) to prevent other computers with the same name from accidentally becoming member of the domain. The computer account created by Samba is visible in the `/etc/passwd` file on Linux. Computer accounts appear as a normal user account, but end their name with a dollar sign. Below a screenshot of the windows 2003 computer account, created by Samba 3.

```
[root@RHEL52 samba]# tail -5 /etc/passwd
jelena:x:510:514::/home/jelena:/bin/bash
figo:x:511:515::/home/figo:/bin/bash
ronaldo:x:512:516::/home/ronaldo:/bin/bash
pfaff:x:513:517::/home/pfaff:/bin/bash
w2003ee$:x:514:518::/home/nobody:/bin/false
```

To be able to create the account, you will need to provide credentials of an account with the permission to create accounts (by default only root can do this on Linux). And we will have to tell Samba how to do this, by adding an **add machine script** to the global section of smb.conf.

```
add machine script = /usr/sbin/useradd -s /bin/false -d /home/nobody %u
```

You can now join a Microsoft computer to the sports domain (with the root user). After reboot of the Microsoft computer, you will be able to logon with Administrator (password Stargate1), but you will get an error about your roaming profile. We will fix this in the next section.

When joining the samba domain, you have to enter the credentials of a Linux account that can create users (usually only root can do this). If the Microsoft computer complains with **The parameter is incorrect**, then you possibly forgot to add the **add machine script**.

64.10. local or roaming profiles

For your information, if you want to force local profiles instead of roaming profiles, then simply add the following two lines to the global section in smb.conf.

```
logon home =
logon path =
```

Microsoft computers store a lot of User Metadata and application data in a user profile. Making this profile available on the network will enable users to keep their Desktop and Application settings across computers. User profiles on the network are called **roaming profiles** or **roving profiles**. The Samba domain controller can manage these profiles. First we need to add the relevant section in smb.conf.

```
[Profiles]
comment = User Profiles
path = /srv/samba/profiles
readonly = No
profile acls = Yes
```

Besides the share section, we also need to set the location of the profiles share (this can be another Samba server) in the global section.

```
logon path = \\%L\Profiles\%U
```

The **%L** variable is the name of this Samba server, the **%U** variable translates to the username. After adding a user to `smbpasswd` and letting the user log on and off, the profile of the user will look like this.

```
[root@RHEL4b samba]# ll /srv/samba/profiles/Venus/
total 568
drwxr-xr-x  4 Venus Venus  4096 Jul  5 10:03 Application Data
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 Cookies
drwxr-xr-x  3 Venus Venus  4096 Jul  5 10:03 Desktop
drwxr-xr-x  3 Venus Venus  4096 Jul  5 10:03 Favorites
drwxr-xr-x  4 Venus Venus  4096 Jul  5 10:03 My Documents
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 NetHood
-rwxr--r--  1 Venus Venus 524288 Jul  5  2007 NTUSER.DAT
-rwxr--r--  1 Venus Venus  1024 Jul  5  2007 NTUSER.DAT.LOG
-rw-r--r--  1 Venus Venus   268 Jul  5 10:03 ntuser.ini
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 PrintHood
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 Recent
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 SendTo
drwxr-xr-x  3 Venus Venus  4096 Jul  5 10:03 Start Menu
drwxr-xr-x  2 Venus Venus  4096 Jul  5 10:03 Templates
```

64.11. Groups in NTFS acls

We have users on Unix, we have groups on Unix that contain those users.

```
[root@RHEL4b samba]# grep nt /etc/group
...
ntadmins:x:506:Administrator
ntsports:x:507:Venus,Serena,Kim,Figo,Pfaff
nttennis:x:508:Venus,Serena,Kim
ntfootball:x:509:Figo,Pfaff
[root@RHEL4b samba]#
```

We already added Venus to the `tdbsam` with `smbpasswd`.

```
smbpasswd -a Venus
```

Does this mean that Venus can access the tennis and the sports shares ? Yes, all access works fine on the Samba server. But the `nttennis` group is not available on the windows machines. To make the groups available on windows (like in the `ntfs` security tab of files and folders), we have to map unix groups to windows groups. To do this, we use the **net groupmap** command.

```
[root@RHEL4b samba]# net groupmap add ntgroup="tennis" unixgroup=nttennis type=d
No rid or sid specified, choosing algorithmic mapping
Successfully added group tennis to the mapping db
[root@RHEL4b samba]# net groupmap add ntgroup="football" unixgroup=ntfootball type=d
No rid or sid specified, choosing algorithmic mapping
Successfully added group football to the mapping db
[root@RHEL4b samba]# net groupmap add ntgroup="sports" unixgroup=ntsports type=d
No rid or sid specified, choosing algorithmic mapping
```

```
Successfully added group sports to the mapping db
[root@RHEL4b samba]#
```

Now you can use the Samba groups on all NTFS volumes on members of the domain.

64.12. logon scripts

Before testing a logon script, make sure it has the proper carriage returns that DOS files have.

```
[root@RHEL4b netlogon]# cat start.bat
net use Z: \\DCSPORTS0\SPORTS
[root@RHEL4b netlogon]# unix2dos start.bat
unix2dos: converting file start.bat to DOS format ...
[root@RHEL4b netlogon]#
```

Then copy the scripts to the netlogon share, and add the following parameter to `smb.conf`.

```
logon script = start.bat
```


64.13. practice: samba domain controller

1. Setup Samba as a domain controller.
2. Create the shares salesdata, salespresentations and meetings. Salesdata must be accessible to all sales people and to all managers. SalesPresentations is only for all sales people. Meetings is only accessible to all managers. Use groups to accomplish this.
3. Join a Microsoft computer to your domain. Verify the creation of a computer account in /etc/passwd.
4. Setup and verify the proper working of roaming profiles.
5. Find information about home directories for users, set them up and verify that users receive their home directory mapped under the H:-drive in MS Windows Explorer.
6. Use a couple of samba domain groups with members to set acls on ntfs. Verify that it works!
7. Knowing that the %m variable contains the computername, create a separate log file for every computer(account).
8. Knowing that %s contains the client operating system, include a smb.%s.conf file that contains a share. (The share will only be visible to clients with that OS).
9. If time permits (or if you are waiting for other students to finish this practice), then combine "valid users" and "invalid users" with groups and usernames with "hosts allow" and "hosts deny" and make a table of which get priority over which.

Chapter 65. a brief look at samba 4

Table of Contents

65.1. Samba 4 alpha 6	598
-----------------------------	-----

65.1. Samba 4 alpha 6

A quick view on Samba 4 alpha 6 (January 2009). You can also follow this guide <http://wiki.samba.org/index.php/Samba4/HOWTO>

Remove old Samba from Red Hat

```
yum remove samba
```

set a fix ip address (Red Hat has an easy GUI)

download and untar

```
samba.org, click 'download info', choose mirror, dl samba4 latest alpha
```

once untarred, enter the directory and read the howto4.txt

```
cd samba-4.0.0alpha6/
```

```
more howto4.txt
```

first we have to configure, compile and install samba4

```
cd source4/
```

```
./configure
```

```
make
```

```
make install
```

Then we can use the provision script to setup our realm. I used booi.schot as domain name (instead of example.com).

```
./setup/provision --realm=BOOI.SCHOT --domain=BOOI --adminpass=stargate \  
--server-role='domain controller'
```

i added a simple share for testing

```
vi /usr/local/samba/etc/smb.conf
```

then i started samba

```
cd /usr/local/samba/sbin/
```

```
./samba
```

I tested with smbclient, it works

```
smbclient //localhost/test -Uadministrator%stargate
```

I checked that bind (and bind-chroot) were installed (yes), so copied the srv records

```
cp booi.schot.zone /var/named/chroot/etc/
```

then appended to named.conf

```
cat named.conf >> /var/named/chroot/etc/named.conf
```

I followed these steps in the howto4.txt

```
vi /etc/init.d/named [added two export lines right after start()]
chmod a+r /usr/local/samba/private/dns.keytab
cp krb5.conf /etc/
vi /var/named/chroot/etc/named.conf
--> remove a lot, but keep allow-update { any; };
```

restart bind (named!), then tested dns with dig, this works (stripped screenshot!)

```
[root@RHEL52 private]# dig _ldap._tcp.dc._msdcs.booi.schot SRV @localhost

; (1 server found)
;; global options: printcmd
;; Got answer:
;; -HEADER- opcode: QUERY, status: NXDOMAIN, id: 58186
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;_ldap._tcp.dc._msdcs.booi.schot. IN SRV

;; AUTHORITY SECTION:
.      10800 IN SOA A.ROOT-SERVERS.NET....

;; Query time: 54 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Tue Jan 27 20:57:05 2009
;; MSG SIZE rcvd: 124

[root@RHEL52 private]#
```

made sure /etc/resolv.conf points to himself

```
[root@RHEL52 private]# cat /etc/resolv.conf
search booi.schot
nameserver 127.0.0.1
```

start windows 2003 server, enter the samba4 as DNS!

ping the domain, if it doesn't work, then add your redhats hostname and your realm to windows/system32/drivers/etc/hosts

join the windows computer to the domain

reboot the windows

log on with administrator stargate

start run dsa.msc to manage samba4

create an OU, a user and a GPO, test that it works

Part XVII. dns server

Chapter 66. introduction to DNS

Table of Contents

66.1. about dns	602
66.2. dns namespace	604
66.3. caching only servers	609
66.4. authoritative dns servers	611
66.5. primary and secondary	611
66.6. zone transfers	611
66.7. master and slave	612
66.8. SOA record	612
66.9. full or incremental zone transfers	613
66.10. DNS cache	614
66.11. forward lookup zone example	615
66.12. Practice: caching only DNS server	616
66.13. Practice: caching only with forwarder	619
66.14. Practice: primary authoritative server	621
66.15. Practice: reverse DNS	623
66.16. Practice: a DNS slave server	624

Every computer on the internet is connected to a huge worldwide tree of **dns** servers. Most organisations have more than one **dns server**, and even Personal Area Networks have a **built-in dns server** in a small modem or router.

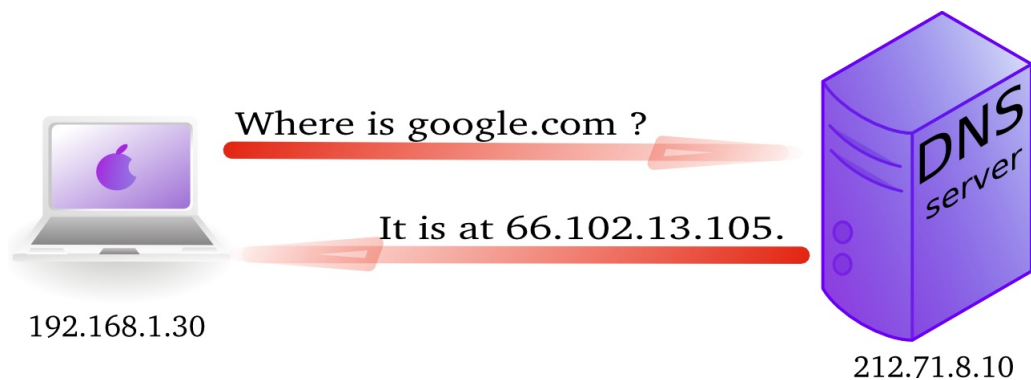
In this chapter we will explain what **dns** actually is and how to set it up using Linux.

66.1. about dns

name to ip-address resolution

The **domain name system** or **dns** is a service on a tcp/ip network that enables clients to translate names into ip-addresses. It is much more than that, but let's keep it simple for now.

When you use a browser to go to a website, then you type the name of that website in the url bar. But for your computer to actually communicate with the web server hosting said website, your computer needs the ip-address of that web server. That is where **dns** comes in.



In wireshark you can use the **dns** filter to see this traffic.

No. .	Time	Source	Destination	Protocol	Info
4560	11.467767	192.168.1.30	212.71.8.10	DNS	Standard query A google.com
4569	11.487774	212.71.8.10	192.168.1.30	DNS	Standard query response A 66.102.13.105

history

In the Seventies, only a few hundred computers were connected to the internet. To resolve names, computers had a flat file that contained a table to resolve hostnames to ip-addresses. This local file was downloaded from **hosts.txt** on an ftp server in Stanford.

In 1984 **Paul Mockapetris** created **dns**, a distributed treelike hierarchical database that will be explained in detail in these chapters.

Today, **dns** or **domain name system** is a worldwide distributed hierarchical database controlled by **ICANN**. Its primary function is to resolve names to ip addresses, and to point to internet servers providing **smtp** or **ldap** services.

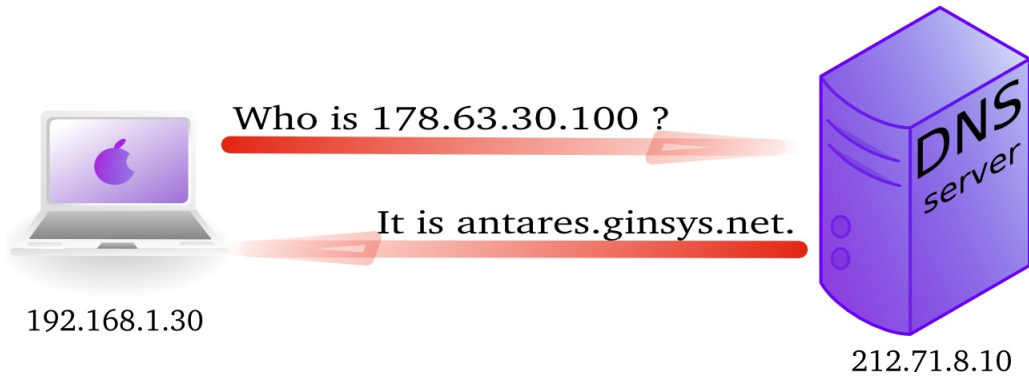
The old **hosts.txt** file is still active today on most computer systems under the name **/etc/hosts**. We will discuss this file later, as it can influence name resolution.

forward and reverse lookup queries

The question a client asks a dns server is called a **query**. When a client queries for an ip-address, this is called a **forward lookup query** (as seen in the previous drawing).

The reverse, a query for the name of a host, is called a **reverse lookup query**.

Below a picture of a **reverse lookup query**.



Here is a screenshot of a **reverse lookup query** in **nslookup**.

```
paul@ubu1010:~$ nslookup
> set type=PTR
> 178.63.30.100
Server: 212.71.8.10
Address: 212.71.8.10#53

Non-authoritative answer:
100.30.63.178.in-addr.arpa name = antares.ginsys.net.
```

This is what a reverse lookup looks like when sniffing with **wireshark**.

No. .	Time	Source	Destination	Protocol	Info
280	172.307847	192.168.1.30	212.71.8.10	DNS	Standard query PTR 100.30.63.178.in-addr.arpa
281	172.321299	212.71.8.10	192.168.1.30	DNS	Standard query response PTR antares.ginsys.net

/etc/resolv.conf

A client computer needs to know the ip-address of the **dns server** to be able to send queries to it. This is either provided by a **dhcp server** or manually entered.

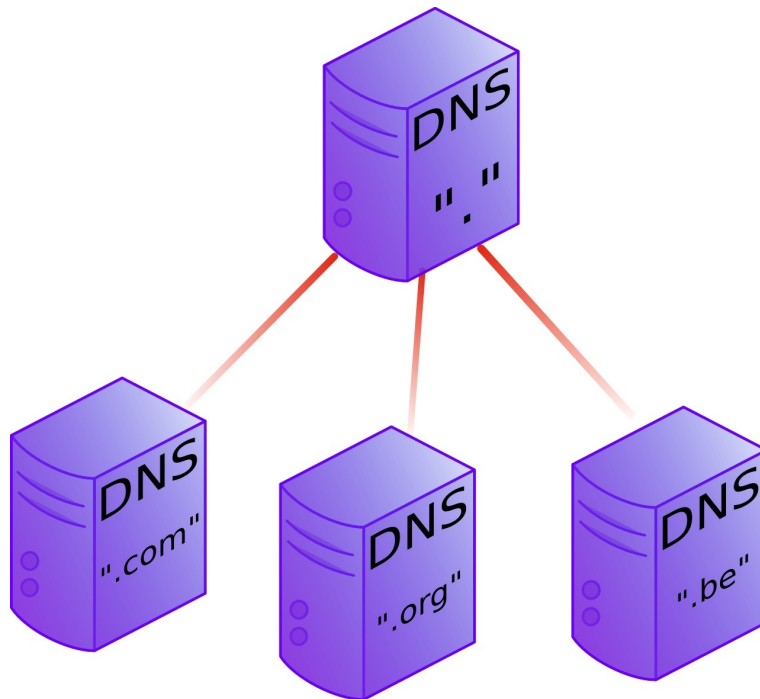
Linux clients keep this information in the **/etc/resolv.conf** file.

```
paul@ubu1010:~$ cat /etc/resolv.conf
nameserver 212.71.8.10
```

66.2. dns namespace

hierarchy

The **dns namespace** is hierarchical tree structure, with the **root servers** (aka dot-servers) at the top. The **root servers** are usually represented by a dot.



Below the **root-servers** are the **Top Level Domains** or **tld's**.

There are more **tld's** than shown in the picture. Currently about 200 countries have a **tld**. And there are several general **tld's** like .com, .edu, .org, .gov, .net, .mil, .int and more recently also .aero, .info, .museum, ...

root servers

There are thirteen **root servers** on the internet, they are named **A** to **M**. Journalists often refer to these servers as **the master servers of the internet**, because if these servers go down, then nobody can (use names to) connect to websites.

The root servers are not thirteen physical machines, they are many more. For example the **F** root server consists of 46 physical machines that all behave as one (using anycast).

<http://root-servers.org>
<http://f.root-servers.org>
http://en.wikipedia.org/wiki/Root_nameserver.

root hints

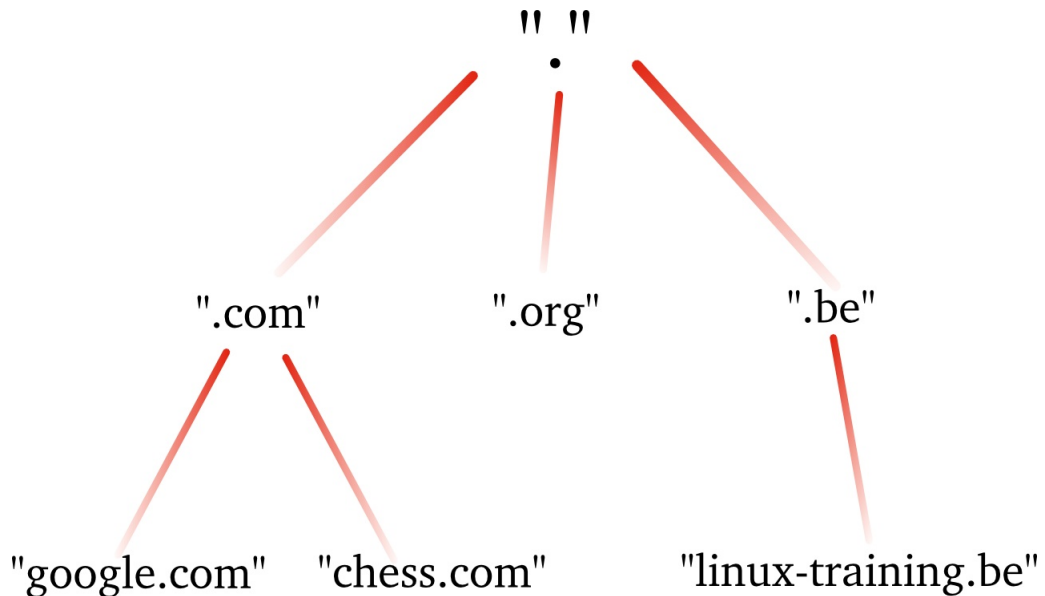
Every **dns server software** will come with a list of **root hints** to locate the **root servers**.

```
root@gwen:~# grep ' A ' /etc/bind/db.root
A.ROOT-SERVERS.NET.      3600000      A      198.41.0.4
B.ROOT-SERVERS.NET.      3600000      A      192.228.79.201
C.ROOT-SERVERS.NET.      3600000      A      192.33.4.12
D.ROOT-SERVERS.NET.      3600000      A      128.8.10.90
E.ROOT-SERVERS.NET.      3600000      A      192.203.230.10
F.ROOT-SERVERS.NET.      3600000      A      192.5.5.241
G.ROOT-SERVERS.NET.      3600000      A      192.112.36.4
H.ROOT-SERVERS.NET.      3600000      A      128.63.2.53
I.ROOT-SERVERS.NET.      3600000      A      192.36.148.17
J.ROOT-SERVERS.NET.      3600000      A      192.58.128.30
K.ROOT-SERVERS.NET.      3600000      A      193.0.14.129
L.ROOT-SERVERS.NET.      3600000      A      199.7.83.42
M.ROOT-SERVERS.NET.      3600000      A      202.12.27.33
```

domains

One level below the **top level domains** are the **domains**. Domains can have subdomains (also called child domains).

This picture shows **dns domains** like google.com, chess.com, linux-training.be (there are millions more).



DNS domains are registered at the **tld** servers, the **tld** servers are registered at the **dot servers**.

top level domains

Below the root level are the **top level domains** or **tld's**. Originally there were only seven defined:

Table 66.1. the first top level domains

year	TLD	purpose
1985	.arpa	Reverse lookup via in-addr.arpa
1985	.com	Commercial Organizations
1985	.edu	US Educational Institutions
1985	.gov	US Government Institutions
1985	.mil	US Military
1985	.net	Internet Service Providers, Internet Infrastructure
1985	.org	Non profit Organizations
1988	.int	International Treaties like nato.int

Country **tld's** were defined for individual countries, like **.uk** in 1985 for Great Britain (yes really), **.be** for Belgium in 1988 and **.fr** for France in 1986. See RFC 1591 for more info.

In 1998 seven new general purpose **tld's** where chosen, they became active in the 21st century.

Table 66.2. new general purpose tld's

year	TLD	purpose
2002	.aero	aviation related
2001	.biz	businesses
2001	.coop	for co-operatives
2001	.info	informative internet resources
2001	.museum	for museums
2001	.name	for all kinds of names, pseudonyms and labels...
2004	.pro	for professionals

Many people were surprised by the choices, claiming not much use for them and wanting a separate **.xxx** domain (introduced in 2011) for adult content, and **.kidz** a save haven for children. In the meantime more useless **tld's** were create like **.travel** (for travel agents) and **.tel** (for internet communications) and **.jobs** (for jobs sites).

fully qualified domain name

The **fully qualified domain name** or **fqdn** is the combination of the **hostname** of a machine appended with its **domain name**.

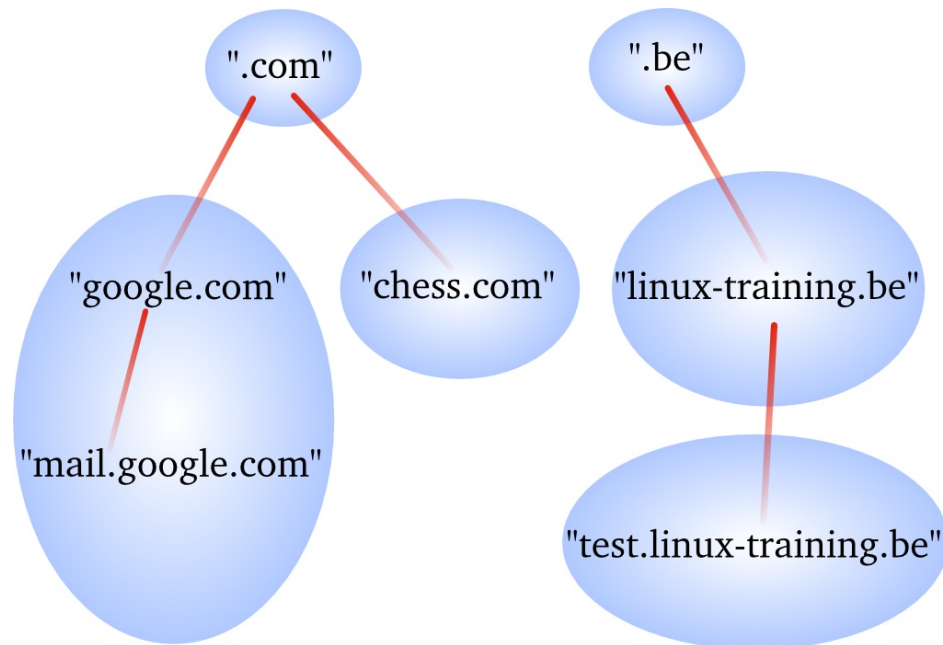
If for example a system is called **gwen** and it is in the domain **linux-training.be**, then the fqdn of this system is **gwen.linux-training.be**.

On Linux systems you can use the **hostname** and **domainname** commands to verify this information.

```
root@gwen:~# hostname
gwen
root@gwen:~# domainname
linux-training.be
root@gwen:~# hostname --fqdn
gwen.linux-training.be
```

dns zones

A **zone** (aka a **zone of authority**) is a portion of the DNS tree that covers one domain name or child domain name. The picture below represents zones as blue ovals. Some zones will contain delegate authority over a child domain to another zone.



A **dns server** can be **authoritative** over 0, 1 or more **dns zones**. We will see more details later on the relation between a **dns server** and a **dns zone**.

A **dns zone** consists of **records**, also called **resource records**. We will list some of those **resource records** on the next page.

dns records

A record

The **A record**, which is also called a **host record** contains the ipv4-address of a computer. When a DNS client queries a DNS server for an A record, then the DNS server will resolve the hostname in the query to an ip-address. An **AAAA record** is similar but contains an ipv6 address instead of ipv4.

PTR record

A **PTR record** is the reverse of an A record. It contains the name of a computer and can be used to resolve an ip-address to a hostname.

NS record

A **NS record** or **nameserver record** is a record that points to a DNS name server (in this zone). You can list all your name servers for your DNS zone in distinct NS records.

glue A record

An A record that maps the name of an NS record to an ip address is said to be a **glue record**.

SOA record

The SOA record of a zone contains meta information about the zone itself. The contents of the SOA record is explained in detail in the section about zone transfers. There is exactly one SOA record for each zone.

CNAME record

A **CNAME record** maps a hostname to a hostname, creating effectively an alias for an existing hostname. The name of the mail server is often aliased to **mail** or **smtp**, and the name of a web server to **www**.

MX record

The **MX** record points to an **smtp server**. When you send an email to another domain, then your mail server will need the MX record of the target domain's mail server.

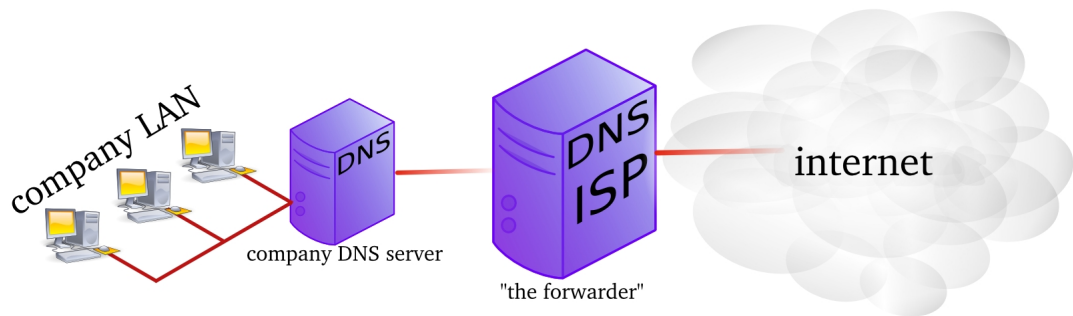
66.3. caching only servers

A **dns server** that is set up without **authority** over a **zone**, but that is connected to other name servers and caches the queries is called a **caching only name server**. Caching only name servers do not have a **zone database** with resource records. Instead they connect to other name servers and cache that information.

There are two kinds of caching only name servers. Those with a **forwarder**, and those that use the **root servers**.

caching only server with forwarder

A **caching only server** with a **forwarder** is a DNS server that will get all its information from the **forwarder**. The **forwarder** must be a **dns server** for example the **dns server** of an **internet service provider**.



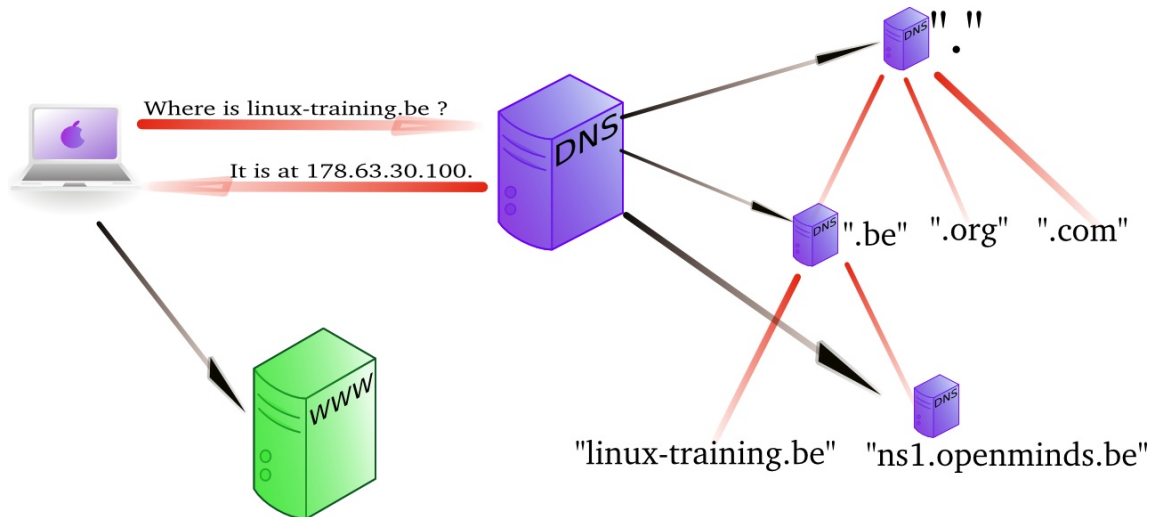
This picture shows a **dns server** on the company LAN that has set the **dns server** from their **isp** as a **forwarder**. If the ip address of the **isp dns server** is 212.71.8.10, then the following lines would occur in the **named.conf** file of the company **dns server**:

```
forwarders {  
    212.71.8.10;  
};
```

caching only server without forwarder

A caching only server without forwarder will have to get information elsewhere. When it receives a query from a client, then it will consult one of the **root servers**. The **root server** will refer it to a **tld** server, which will refer it to another **dns** server. That last server might know the answer to the query, or may refer to yet another server. In the end, our hard working **dns** server will find an answer and report this back to the client.

In the picture below, the clients asks for the ip address of linux-training.be. Our caching only server will contact the root server, and be referred to the .be server. It will then contact the .be server and be referred to one of the name servers of Openminds. One of these name servers (in this cas ns1.openminds.be) will answer the query with the ip-address of linux-training.be. When our caching only server reports this to the client, then the client can connect to this website.

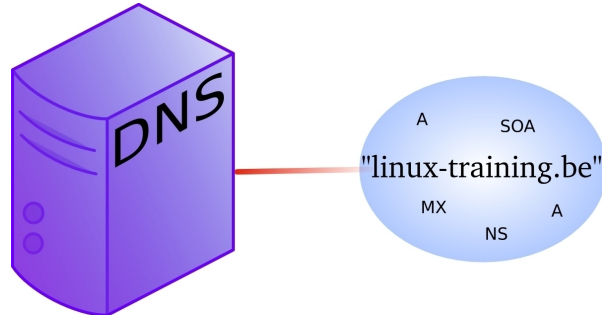


iterative or recursive query

A **recursive query** is a DNS query where the client that is submitting the query expects a complete answer (Like the fat red arrow above going from the Macbook to the DNS server). An **iterative query** is a DNS query where the client does not expect a complete answer (the three black arrows originating from the DNS server in the picture above). Iterative queries usually take place between name servers. The root name servers do not respond to recursive queries.

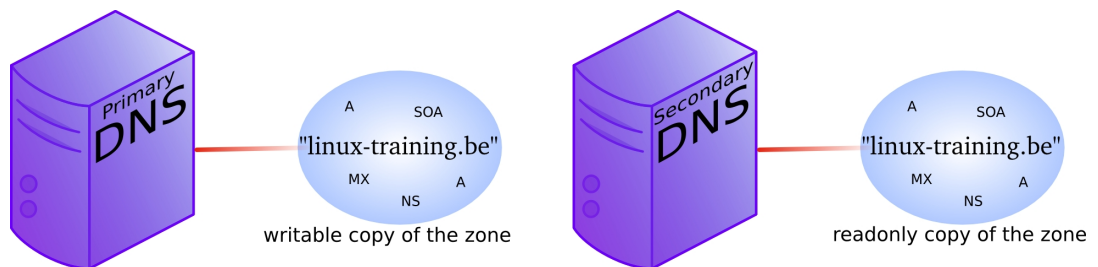
66.4. authoritative dns servers

A DNS server that is controlling a zone, is said to be the **authoritative** DNS server for that zone. Remember that a **zone** is a collection of **resource records**.



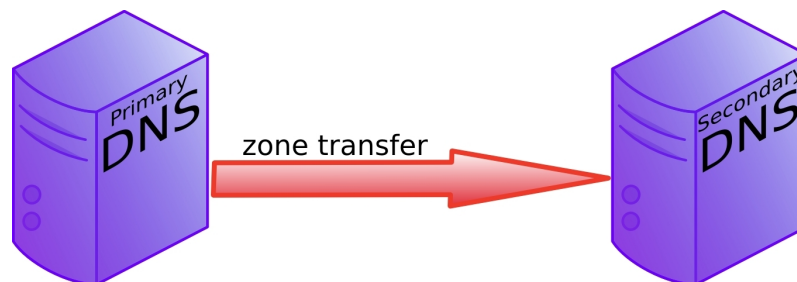
66.5. primary and secondary

When you set up the first **authoritative** dns server for a zone, then this is called the **primary dns server**. This server will have a readable and writable copy of the **zone database**. For reasons of fault tolerance, performance or load balancing you may decide to set up another **dns server** with authority over that zone. This is called a **secondary** dns server.



66.6. zone transfers

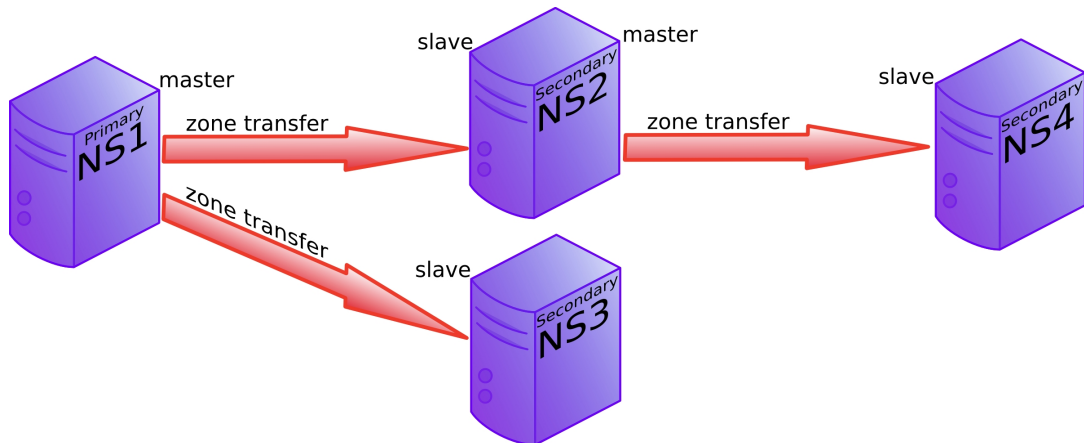
The slave server receives a copy of the zone database from the master server using a **zone transfer**. Zone transfers are requested by the slave servers at regular intervals. Those intervals are defined in the **soa record**.



66.7. master and slave

When adding a **secondary dns server** to a zone, then you will configure this server as a **slave server** to the **primary server**. The primary server then becomes the **master server** of the slave server.

Often the **primary dns server** is the **master** server of all slaves. Sometimes a **slave server** is **master server** for a second line slave server. In the picture below ns1 is the primary dns server and ns2, ns3 and ns4 are secondaries. The master for slaves ns2 and ns3 is ns1, but the master for ns4 is ns2.



66.8. SOA record

The **soa record** contains a **refresh** value. If this is set to 30 minutes, then the slave server will request a copy of the zone file every 30 minutes. There is also a **retry** value. The retry value is used when the master server did not reply to the last zone transfer request. The value for **expiry time** says how long the slave server will answer to queries, without receiving a zone update.

Below an example of how to use nslookup to query the **soa record** of a zone (linux-training.be).

```

root@debian6:~# nslookup
> set type=SOA
> server ns1.openminds.be
> linux-training.be
Server:          ns1.openminds.be
Address:         195.47.215.14#53

linux-training.be
  origin = ns1.openminds.be
  mail addr = hostmaster.openminds.be
  serial = 2321001133
  refresh = 14400
  retry = 3600
  expire = 604800
  minimum = 3600
  
```

Zone transfers only occur when the zone database was updated (meaning when one or more resource records were added, removed or changed on the master server). The

slave server will compare the **serial number** of its own copy of the SOA record with the serial number of its master's SOA record. When both serial numbers are the same, then no update is needed (because no records were added, removed or deleted). When the slave has a lower serial number than its master, then a zone transfer is requested.

Below a zone transfer captured in wireshark.

Time	Source	Destination	Protocol	Info
1 0.000000	192.168.1.37	192.168.1.35	DNS	Standard query SOA cobbaut.paul
2 0.008502	192.168.1.35	192.168.1.37	DNS	Standard query response SOA ns.cobbaut.paul
3 0.014672	192.168.1.37	192.168.1.35	TCP	33713 > domain [SYN] Seq=0 Win=5840 Len=0 MS
4 0.015215	192.168.1.35	192.168.1.37	TCP	domain > 33713 [SYN, ACK] Seq=0 Ack=1 Win=57
5 0.015307	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=1 Ack=1 Win=5856 Le
6 0.015954	192.168.1.37	192.168.1.35	TCP	[TCP segment of a reassembled PDU]
7 0.018359	192.168.1.35	192.168.1.37	TCP	domain > 33713 [ACK] Seq=1 Ack=3 Win=5792 Le
8 0.018411	192.168.1.37	192.168.1.35	DNS	Standard query IXFR cobbaut.paul
9 0.018823	192.168.1.35	192.168.1.37	TCP	domain > 33713 [ACK] Seq=1 Ack=77 Win=5792 L
10 0.019784	192.168.1.35	192.168.1.37	DNS	Standard query response SOA ns.cobbaut.paul
11 0.019821	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=77 Ack=295 Win=6912
12 0.020618	192.168.1.37	192.168.1.35	TCP	33713 > domain [FIN, ACK] Seq=77 Ack=295 Win
13 0.021011	192.168.1.35	192.168.1.37	TCP	domain > 33713 [FIN, ACK] Seq=295 Ack=78 Win
14 0.021040	192.168.1.37	192.168.1.35	TCP	33713 > domain [ACK] Seq=78 Ack=296 Win=6912

66.9. full or incremental zone transfers

When a zone transfer occurs, this can be either a full zone transfer or an incremental zone transfer. The decision depends on the size of the transfer that is needed to completely update the zone on the slave server. An incremental zone transfer is preferred when the total size of changes is smaller than the size of the zone database. Full zone transfers use the **axfr** protocol, incremental zone transfer use the **ixfr** protocol.

66.10. DNS cache

DNS is a caching protocol.

When a client queries its local DNS server, and the local DNS server is not authoritative for the query, then this server will go looking for an authoritative name server in the DNS tree. The local name server will first query a root server, then a **tld** server and then a domain server. When the local name server resolves the query, then it will relay this information to the client that submitted the query, and it will also keep a copy of these queries in its cache. So when a(nother) client submits the same query to this name server, then it will retrieve this information from its cache.

For example, a client queries for the A record on `www.linux-training.be` to its local server. This is the first query ever received by this local server. The local server checks that it is not authoritative for the `linux-training.be` domain, nor for the **.be tld**, and it is also not a root server. So the local server will use the root hints to send an **iterative** query to a root server.

The root server will reply with a reference to the server that is authoritative for the `.be` domain (root DNS servers do not resolve fqdn's, and root servers do not respond to recursive queries).

The local server will then send an iterative query to the authoritative server for the **.be tld**. This server will respond with a reference to the name server that is authoritative for the `linux-training.be` domain.

The local server will then send the query for `www.linux-training.be` to the authoritative server (or one of its slave servers) for the `linux-training.be` domain. When the local server receives the ip-address for `www.linux-training.be`, then it will provide this information to the client that submitted this query.

Besides caching the A record for `www.linux-training.be`, the local server will also cache the NS and A record for the `linux-training.be` name server and the `.be` name server.

66.11. forward lookup zone example

The way to set up zones in `/etc/named.conf` is to create a zone entry with a reference to another file located in `/var/named`.

Here is an example of such an entry in `/etc/named.conf`:

```
zone "classdemo.local" IN {
    type master;
    file "classdemo.local.zone";
    allow-update { none; };
};
```

To create the zone file, the easy method is to copy an existing zone file (this is easier than writing from scratch).

```
[root@RHEL4b named]# cd /var/named/
[root@RHEL4b named]# pwd
/var/named
[root@RHEL4b named]# cp localhost.zone classdemo.local.zone
[root@RHEL4b named]#
```

Here is an example of a zone file.

```
[root@RHEL4b named]# cat classdemo.local.zone
$TTL      86400
$ORIGIN   classdemo.local.
@         IN SOA  rhel4b.classdemo.local.  admin.classdemo.local. (
                                2007083100      ; serial
                                3H                ; refresh
                                900               ; retry
                                1W                ; expiry
                                1D )              ; minimum

                                IN NS           rhel4b.classdemo.local.
                                IN MX          10 mail.classdemo.local.
                                IN A           192.168.1.191

rhel4b    IN      A       192.168.1.191
mail      IN      A       192.168.1.191
www       IN      A       192.168.1.191
ftp       IN      A       192.168.1.191
server2   IN      A       192.168.1.1
```

66.12. Practice: caching only DNS server

1a. installing DNS software on Debian/Ubuntu

```
root@ubul010srv:~# dpkg -l | grep bind9
ii  bind9-host      1:9.7.1.dfsg.P2-2ubuntu0.2  Version of 'host' bundled with BIND 9.X
ii  libbind9-60    1:9.7.1.dfsg.P2-2ubuntu0.2  BIND9 Shared Library used by BIND
root@ubul010srv:~# aptitude install bind9
The following NEW packages will be installed:
  bind9 bind9utils{a}
0 packages upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 433kB of archives. After unpacking 1,352kB will be used.
Do you want to continue? [Y/n/?]

... output truncated ...
```

```
* Starting domain name service... bind9 [ OK ]
```

```
root@ubul010srv:~# dpkg -l | grep bind9
ii bind9          1:9.7.1.dfsg.P2-2ubuntu0.2  Internet Domain Name Server
ii bind9-host    1:9.7.1.dfsg.P2-2ubuntu0.2  Version of 'host' bundled with BIND 9.X
ii bind9utils    1:9.7.1.dfsg.P2-2ubuntu0.2  Utilities for BIND
ii libbind9-60  1:9.7.1.dfsg.P2-2ubuntu0.2  BIND9 Shared Library used by BIND
root@ubul010srv:~#
```

1b. installing DNS software on RHEL/Fedora

```
[root@fedora14 ~]# rpm -qa | grep bind
samba-winbind-clients-3.5.8-74.fc14.i686
bind-utils-9.7.3-1.fc14.i686
PackageKit-device-rebind-0.6.12-2.fc14.i686
bind-libs-9.7.3-1.fc14.i686
[root@fedora14 ~]# yum install bind
Loaded plugins: langpacks, presto, refresh-packagekit
Adding en_US to language list
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package bind.i686 32:9.7.3-1.fc14 set to be installed
--> Finished Dependency Resolution
```

```
...output truncated
```

```
Running Transaction
  Installing      : 32:bind-9.7.3-1.fc14.i686                1/1
```

```
Installed:
  bind.i686 32:9.7.3-1.fc14
```

```
Complete!
[root@fedora14 ~]# rpm -qa | grep bind
samba-winbind-clients-3.5.8-74.fc14.i686
bind-utils-9.7.3-1.fc14.i686
PackageKit-device-rebind-0.6.12-2.fc14.i686
bind-libs-9.7.3-1.fc14.i686
bind-9.7.3-1.fc14.i686
[root@fedora14 ~]#
```

2. Discover the default configuration files. Can you define the purpose of each file ?

2a. On Fedora:

```
[root@fedora14 ~]# ls -ld /etc/named*
drwxr-x---. 2 root named 4096 Feb 18 16:07 /etc/named
-rw-r-----. 1 root named 1008 Jul 19 2010 /etc/named.conf
-rw-r--r---. 1 root named 2544 Feb 18 16:07 /etc/named.iscdlv.key
-rw-r-----. 1 root named 931 Jun 21 2007 /etc/named.rfc1912.zones
-rw-r--r---. 1 root named 487 Jul 19 2010 /etc/named.root.key
[root@fedora14 ~]# ls -l /var/named/
total 28
drwxrwx---. 2 named named 4096 Feb 18 16:07 data
drwxrwx---. 2 named named 4096 Feb 18 16:07 dynamic
-rw-r-----. 1 root named 1892 Feb 18 2008 named.ca
-rw-r-----. 1 root named 152 Dec 15 2009 named.empty
-rw-r-----. 1 root named 152 Jun 21 2007 named.localhost
-rw-r-----. 1 root named 168 Dec 15 2009 named.loopback
drwxrwx---. 2 named named 4096 Feb 18 16:07 slaves
```

2. On Ubuntu:

```
root@ubul010srv:~# ls -l /etc/bind
total 52
-rw-r--r-- 1 root root 601 2011-02-23 16:22 bind.keys
-rw-r--r-- 1 root root 237 2011-02-23 16:22 db.0
-rw-r--r-- 1 root root 271 2011-02-23 16:22 db.127
-rw-r--r-- 1 root root 237 2011-02-23 16:22 db.255
-rw-r--r-- 1 root root 353 2011-02-23 16:22 db.empty
-rw-r--r-- 1 root root 270 2011-02-23 16:22 db.local
-rw-r--r-- 1 root root 2994 2011-02-23 16:22 db.root
-rw-r--r-- 1 root bind 463 2011-02-23 16:22 named.conf
-rw-r--r-- 1 root bind 490 2011-02-23 16:22 named.conf.default-zones
-rw-r--r-- 1 root bind 165 2011-02-23 16:22 named.conf.local
-rw-r--r-- 1 root bind 572 2011-02-23 16:22 named.conf.options
-rw-r----- 1 bind bind 77 2011-05-15 17:52 rndc.key
-rw-r--r-- 1 root root 1317 2011-02-23 16:22 zones.rfc1918
```

3. Setup caching only dns server. This is normally the default setup. A caching-only name server will look up names for you and cache them. Most tutorials will tell you to add a **forwarder**, so we first try without this!

```
root@ubul010srv:/var/log# nslookup
> server 192.168.1.37
Default server: 192.168.1.37
Address: 192.168.1.37#53
>
> slashdot.org
Server: 192.168.1.37
Address: 192.168.1.37#53

Non-authoritative answer:
Name: slashdot.org
Address: 216.34.181.45
```

Hey this seems to work without a forwarder. Using a sniffer you can find out what really happens (since the server is not using a cache, not using your dns-server (from /etc/resolv.conf). So where is this information coming from, and what can you learn from sniffing this dns traffic ?

4. Explain in detail what happens when you enable a caching only dns server without forwarder. This wireshark screenshot can help, but you learn more by sniffing the traffic yourself! I will choose two volunteers to explain this in front of the class.

66.13. Practice: caching only with forwarder

5. Add a local dns-server as a forwarder (at my home this is 192.168.1.1, probably different ip in a classroom!).

```
root@ubul010srv:~# grep -A2 forwarder /etc/bind/named.conf.options | t\
ail -3
forwarders {
    192.168.1.1;
};
root@ubul010srv:~# /etc/init.d/bind9 restart
* Stopping domain name service... bind9          [ OK ]
* Starting domain name service... bind9          [ OK ]
root@ubul010srv:~#
```

6. Explain the purpose of adding the forwarder. What is our DNS server doing when it receives a query ? Again the wireshark screenshot can help, you should see something similar.

```
root@ubul010srv:~# nslookup
> server
Default server: 192.168.1.4
Address: 192.168.1.4#53
> server 192.168.1.37
Default server: 192.168.1.37
Address: 192.168.1.37#53
>
> cobbaut.be
Server: 192.168.1.37
Address: 192.168.1.37#53

Non-authoritative answer:
Name: cobbaut.be
Address: 88.151.243.8
```

The screenshot shows a Wireshark capture of a DNS transaction. The filter is set to 'dns'. Two packets are visible:

No. .	Time	Source	Destination	Protocol	Info
278	13.741725	192.168.1.37	192.168.1.1	DNS	Standard query A cobbaut.be
285	13.759925	192.168.1.1	192.168.1.37	DNS	Standard query response A 88.151.243.8

The details pane for the selected packet (No. 278) shows:

- Frame 278 (81 bytes on wire, 81 bytes captured)
- Ethernet II, Src: 8c:7b:9d:d6:df:f2 (8c:7b:9d:d6:df:f2), Dst: ZygateCo_aa:68:f0 (00:02:cf:aa:68:f0)
- Internet Protocol, Src: 192.168.1.37 (192.168.1.37), Dst: 192.168.1.1 (192.168.1.1)
- User Datagram Protocol, Src Port: 44677 (44677), Dst Port: domain (53)
- Domain Name System (query)
 - Transaction ID: 0xf488
 - Flags: 0x0100 (Standard query)
 - Questions: 1
 - Answer RRs: 0
 - Authority RRs: 0
 - Additional RRs: 1
 - Queries
 - cobbaut.be: type A, class IN
 - Additional records

7. What happens when you query for the same domain name more than once ?

8. Why does it say "non-authoritative answer" ? When is a dns server authoritative ?

9. You can also use **dig** instead of **nslookup**.

```
dig @192.168.1.37 linux-training.be
```

10. How can we avoid having to set the server in dig or nslookup ?

```
root@ubul010srv:~# cat /etc/resolv.conf
nameserver 127.0.0.1
```

11. When you use **dig** for the first time for a domain, where is the answer coming from ? And the second time ? How can you tell ?

66.14. Practice: primary authoritative server

1. Instead of only caching the information from other servers, we will now make our server authoritative for our own domain.

2. I choose the new TLD **.paul** and the domain **cobbaut.paul** and put the information in **/etc/bind/named.conf.local**.

```
root@ubul010srv:/etc/bind# grep -C1 cobbaut named.conf.local
```

```
zone "cobbaut.paul" {
    type master;
    file "/etc/bind/db.cobbaut.paul";
};
```

3. Also add a zone database file, similar to this one (add some A records for testing). Set the **Refresh** and **Retry** values not too high so you can sniff this traffic (this example makes the slave server contact the master every 300 seconds).

```
root@ubul010srv:/etc/bind# cat db.cobbaut.paul
;
; BIND data file for domain cobbaut.paul
;
$TTL 604800
@ IN SOA ns.cobbaut.paul. root.cobbaut.paul. (
                20110516      ; Serial
                300          ; Refresh
                200          ; Retry
                2419200      ; Expire
                604800 )      ; Negative Cache TTL
;
@           IN      NS       ns.cobbaut.paul.
ns          IN      A        192.168.1.37
ubul010srv  IN      A        192.168.1.37
anya       IN      A        192.168.1.1
mac        IN      A        192.168.1.30
root@ubul010srv:/etc/bind#
```

4. Restart the DNS server and check your zone in the error log.

```
root@ubul010srv:/etc/bind# grep cobbaut /var/log/daemon.log
May 16 00:33:49 ubul010srv named[25449]: zone cobbaut.paul/IN: loaded\
serial 20110516
```

5. Use dig or nslookup (or even ping) to test your A records.

```
root@ubul010srv:/etc/bind# ping mac.cobbaut.paul
PING mac.cobbaut.paul (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_req=1 ttl=64 time=2.28 ms
64 bytes from 192.168.1.30: icmp_req=1 ttl=64 time=2.31 ms (DUP!)
^C
--- mac.cobbaut.paul ping statistics ---
1 packets transmitted, 1 received, +1 duplicates, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.282/2.296/2.310/0.014 ms
root@ubul010srv:/etc/bind# dig anya.cobbaut.paul

; <<>> DiG 9.7.1-P2 <<>> anya.cobbaut.paul
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38237
```

```
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; QUESTION SECTION:
;anya.cobbaut.paul. IN A

;; ANSWER SECTION:
anya.cobbaut.paul. 604800 IN A 192.168.1.1

;; AUTHORITY SECTION:
cobbaut.paul. 604800 IN NS ns.cobbaut.paul.

;; ADDITIONAL SECTION:
ns.cobbaut.paul. 604800 IN A 192.168.1.37

;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon May 16 00:38:22 2011
;; MSG SIZE rcvd: 84

root@ubul010srv:/etc/bind#
```

6. Our primary server appears to be up and running. Note the information here:

```
server os : Ubuntu 10.10
ip : 192.168.1.37
domain name: cobbaut.paul
server name: ns.cobbaut.paul
```

66.15. Practice: reverse DNS

1. We can add ip to name resolution to our dns-server using a reverse dns zone.
2. Start by adding a .arpa zone to /etc/bind/named.conf.local like this (we set notify to no to avoid sending of notify messages to other name servers):

```
root@ubul010srv:/etc/bind# grep -A4 arpa named.conf.local
zone "1.168.192.in-addr.arpa" {
    type master;
    notify no;
    file "/etc/bind/db.192";
};
```

3. Also create a zone database file for this reverse lookup zone.

```
root@ubul010srv:/etc/bind# cat db.192
;
; BIND reverse data file for 192.168.1.0/24 network
;
$TTL 604800
@ IN SOA ns.cobbaut.paul root.cobbaut.paul. (
    20110516 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative Cache TTL
;
@ IN NS ns.
37 IN PTR ns.cobbaut.paul.
1 IN PTR anya.cobbaut.paul.
30 IN PTR mac.cobbaut.paul.
root@ubul010srv:/etc/bind#
```

4. Test with nslookup or dig:

```
root@ubul010srv:/etc/bind# dig 1.168.192.in-addr.arpa AXFR
```

66.16. Practice: a DNS slave server

1. A slave server transfers zone information over the network from a master server (a slave can also be a master). A primary server maintains zone records in its local file system. As an exercise, and to verify the work of all students, set up a slave server of all the master servers in the classroom.

2. Before configuring the slave server, we have to allow transfers from our zone to this server. Remember that this is not very secure since transfers are in clear text and limited to an ip address. This example follows our demo from above. The ip of my slave server is 192.168.1.31, yours is probably different.

```
root@ubul010srv:/etc/bind# grep -A2 cobbaut named.conf.local
zone "cobbaut.paul" {
    type master;
    file "/etc/bind/db.cobbaut.paul";
    allow-transfer { 192.168.1.31; };
};
root@ubul010srv:/etc/bind#
```

3. My slave server is running Fedora 14. Bind configuration files are only a little different. Below the addition of a slave zone to this server, note the ip address (192.168.1.37) of my master dns server for the cobbaut.paul zone.

```
[root@fedora14 etc]# grep cobbaut -A2 named.conf
zone "cobbaut.paul" {
    type slave;
    file "/var/named/slaves/db.cobbaut.paul";
    masters { 192.168.1.37; };
};
[root@fedora14 etc]#
```

4. You might need to add the ip-address of the server on Fedora to allow queries other than from localhost.

```
[root@fedora14 etc]# grep 127 named.conf
listen-on port 53 { 127.0.0.1; 192.168.1.31; };
```

5. Restarting bind on the slave server should transfer the zone database file:

```
[root@fedora14 etc]# ls -l /var/named/slaves/
total 4
-rw-r--r--. 1 named named 387 May 16 03:23 db.cobbaut.paul
[root@fedora14 etc]#
```

Chapter 67. advanced DNS

Table of Contents

67.1. DNS round robin	626
67.2. DNS delegation	627
67.3. DNS load balancing	628
67.4. DNS notify	628
67.5. testing IXFR and AXFR	628
67.6. DDNS integration with DHCP	628
67.7. reverse is forward in-addr.arpa	629
67.8. ipv6	629
67.9. split-horizon dns	629
67.10. DNS security : file corruption	629
67.11. DNS security : zone transfers	629
67.12. DNS security : zone transfers, ip spoofing	630
67.13. DNS security : queries	630
67.14. DNS security : chrooted bind	630
67.15. DNS security : DNSSEC	630
67.16. DNS security : root	631

67.1. DNS round robin

When you create multiple A records for the same name, then **bind** will do a round robin of the order in which the records are returned. This allows the use of DNS as a load balancer between hosts, since clients will usually take the first ip-address offered.

This is what it looks like in the **zone configuration file**.

```
faith IN A 192.168.1.20
faith IN A 192.168.1.22
```

Below a screenshot of nslookup querying a load balanced A record. Notice the order of ip-addresses returned.

```
> server 192.168.1.35
Default server: 192.168.1.35
Address: 192.168.1.35#53
> faith.cobbaut.paul
Server: 192.168.1.35
Address: 192.168.1.35#53

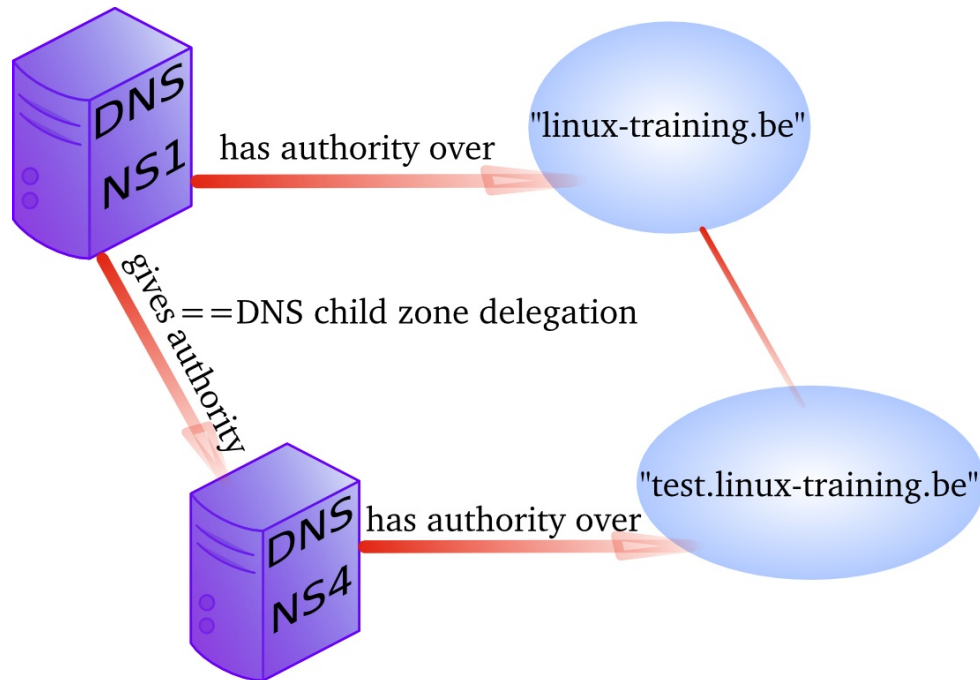
Name: faith.cobbaut.paul
Address: 192.168.1.20
Name: faith.cobbaut.paul
Address: 192.168.1.22
> faith.cobbaut.paul
Server: 192.168.1.35
Address: 192.168.1.35#53

Name: faith.cobbaut.paul
Address: 192.168.1.22
Name: faith.cobbaut.paul
Address: 192.168.1.20
> faith.cobbaut.paul
Server: 192.168.1.35
Address: 192.168.1.35#53

Name: faith.cobbaut.paul
Address: 192.168.1.20
Name: faith.cobbaut.paul
Address: 192.168.1.22
```


67.2. DNS delegation

You can **delegate** a child domain to another DNS server. The child domain then becomes a new zone, with authority at the new dns server.



This is a screenshot of the zone database file with delegation.

```
root@ubu1010srv:/etc/bind# cat db.linear-training.be
$TTL 3d ; default ttl set to three days
$ORIGIN linux-training.be.
@      IN SOA  ns1.linear-training.be. paul.linear-training.be. (
20110524
300
300
10000
20000
)
IN NS  ns1.linear-training.be.
IN NS  ns2.linear-training.be.
IN NS  ns3.linear-training.be.
IN MX  10 smtp.openminds.be.
ns1 IN A 192.168.1.35
ns2 IN A 192.168.1.36
ns3 IN A 192.168.1.37
www IN A 192.168.1.35
mac IN A 192.168.1.30

$ORIGIN office.linear-training.be.
@ IN NS ns4.office.linear-training.be.
; or replace those two lines with:
; office.linear-training.com IN NS ns4.office.linear-training.be

IN NS ns1.linear-training.be. ; in case this is a slave
ns4 IN A 192.168.1.33 ; the glue record
; ns4.office.linear-training.be A 192.168.1.33 ; also ok!
```

67.3. DNS load balancing

Not as above. When you have more than one DNS server authoritative for a zone, you can spread queries amongst all server. One way to do this is by creating NS records for all servers that participate in the load balancing of external queries.

You could also configure different name servers on internal clients.

67.4. DNS notify

The original design of DNS in rfc 1034 and rfc 1035 implemented a **refresh** time in the **SOA** record to configure a time loop for slaves to query their master server. This can result in a lot of useless pull requests, or in a significant lag between updates.

For this reason **dns notify (rfc 1996)** was designed. The server will now notify slaves whenever there is an update. By default this feature is activated in **bind**.

Notify can be disabled as in this screenshot.

```
zone "1.168.192.in-addr.arpa" {
    type master;
    notify no;
    file "/etc/bind/db.192";
};
```

67.5. testing IXFR and AXFR

Full zone transfers (AXFR) are initiated when you restart the bind server, or when you manually update the zone database file directly. With **nsupdate** you can update a zone database and initiate an incremental zone transfer.

You need DDNS allowed for **nsupdate** to work.

```
root@ubul010srv:/etc/bind# nsupdate
> server 127.0.0.1
> update add mac14.linux-training.be 86400 A 192.168.1.23
> send
update failed: REFUSED
```

67.6. DDNS integration with DHCP

Some organizations like to have all their client computers in DNS. This can be cumbersome to maintain. Luckily **rfc 2136** describes integration of DHCP servers with a DNS server. Whenever DHCP acknowledges a client ip configuration, it can notify DNS with this clients ip-address and name. This is called **dynamic updates** or DDNS.

67.7. reverse is forward in-addr.arpa

Reverse lookup is actually implemented as a forward lookup in the **in-addr.arpa** domain. This domain has 256 child domains (from 0.in-addr.arpa to 255.in-addr.arpa), with each child domain having again 256 child domains. And this twice more to a structure of over four billion (2 to the power 32) domains.

67.8. ipv6

With rfc 3596 came ipv6 extensions for DNS. There is the AAAA record for ipv6 hosts on the network, and there is the **ip6.int** domain for reverse lookup (having 16 child domains from 0.ip6.int to f.ip6.int, each of those having again 16 child domains...and this 16 times.

67.9. split-horizon dns

You can use the **view clause** in **bind** to give different results to different clients.

```
view "antwerp" {
match-clients { 172.16.42/24; }; // the network in Antwerp
zone "cobbaut.paul" {
    type master;
    file "/etc/bind/db.cobbaut.paul.antwerp"; // www=172.16.42.9
};

view "brussels" {
match-clients { 172.16.33/24; }; // the Brussels network
zone "cobbaut.paul" {
    type master;
    file "/etc/bind/db.cobbaut.paul.brussels"; // www=172.16.33.4
};
};
```

67.10. DNS security : file corruption

To mitigate file corruption on the **zone files** and the **bind configuration** files protect them with Unix permissions and take regular backups.

67.11. DNS security : zone transfers

Limit zone transfers to certain ip addresses instead of to **any**. Nevermind that ip-addresses can be spoofed, still use this.

67.12. DNS security : zone transfers, ip spoofing

You could setup DNSSEC (which is not the easiest to maintain) and with rfc 2845(tsig?) and with rfc 2930(tkey, but this is open to brute force), or you could disable all zone transfers and use a script with ssh to copy them manually.

67.13. DNS security : queries

Allow recursion only from the local network, and iterative queries from outside only when necessary. This can be configured on master and slave servers.

```
view "internal" {
match-clients { 192.168.42/24; };
recursion yes;
...
};

view "external" {
match-clients { any; };
recursion no;
...
};
```

Or allow only queries from the local network.

```
options {
    allow-query { 192.168.42.0/24; localhost; };
};

zone "cobbaut.paul" {
    allow-query { any; };
};
```

Or only allow recursive queries from internal clients.

```
options {
    allow-recursion { 192.168.42.0/24; localhost; };
};
```

67.14. DNS security : chrooted bind

Most Linux distributions allow an easy setup of bind in a **chrooted** environment.

67.15. DNS security : DNSSEC

DNSSEC uses public/private keys to secure communications, this is described in rfc's 4033, 4034 and 4035.

67.16. DNS security : root

Do not run bind as root. Do not run any application daemon as root.

Part XVIII. dhcp server

Chapter 68. introduction to dhcp

Table of Contents

68.1. four broadcasts	634
68.2. picturing dhcp	635
68.3. installing a dhcp server	636
68.4. dhcp server on Red Hat	636
68.5. dhcp options	636
68.6. client reservations	636
68.7. example config files	637
68.8. older example config files	637
68.9. advanced dhcp	639
68.10. Practice: DHCP and DDNS	640

Dynamic Host Configuration Protocol (or short **dhcp**) is a standard tcp/ip protocol that distributes ip configurations to clients. **dhcp** is defined in **rfc 2131** (before that it was defined as an update to **bootp** in rfc 1531/1541).

The alternative to **dhcp** is manually entering the ip configuration on each client computer.

68.1. four broadcasts

dhcp works with layer 2 broadcasts. A **dhcp** client that starts, will send a **dhcp discover** on the network. All **dhcp servers** (that have a lease available) will respond with a **dhcp offer**. The client will choose one of those offers and will send a **dhcp request** containing the chosen offer. The **dhcp server** usually responds with a **dhcp ack**(knowledge).

In wireshark it looks like this.

Filter: bootp

No.	Time	Source	Destination	Protocol	Info
40383	1687.343978	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transac
40385	1687.647466	192.168.1.200	255.255.255.255	DHCP	DHCP Offer - Transac
40386	1687.647535	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transac
40387	1687.653918	192.168.1.200	255.255.255.255	DHCP	DHCP ACK - Transac

Client IP address: 0.0.0.0 (0.0.0.0)
 Your (client) IP address: 192.168.1.158 (192.168.1.158)
 Next server IP address: 0.0.0.0 (0.0.0.0)
 Relay agent IP address: 0.0.0.0 (0.0.0.0)
 Client MAC address: CadmusCo_5e:38:76 (08:00:27:5e:38:76)
 Client hardware address padding: 00000000000000000000
 Server host name not given
 Boot file name not given
 Magic cookie: (OK)
 ▷ Option: (t=53,l=1) DHCP Message Type = DHCP ACK
 ▷ Option: (t=54,l=4) DHCP Server Identifier = 192.168.1.200
 ▷ Option: (t=51,l=4) IP Address Lease Time = 6 hours
 ▷ Option: (t=81,l=24) Client Fully Qualified Domain Name
 ▷ Option: (t=1,l=4) Subnet Mask = 255.255.255.0
 ▷ Option: (t=15,l=15) Domain Name = "classdemo.local"
 ▷ Option: (t=3,l=4) Router = 192.168.1.1
 ▷ Option: (t=6,l=4) Domain Name Server = 192.168.1.1
 End Option

```

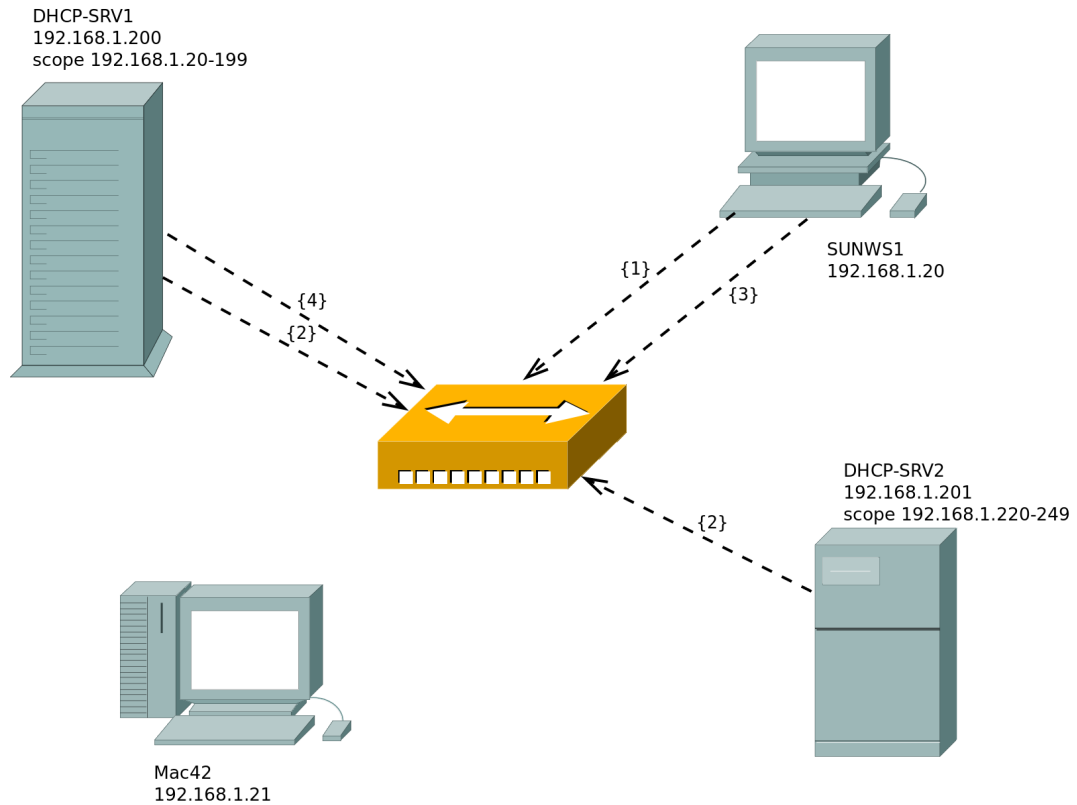
0120  a8 01 c8 33 04 00 00 54 60 51 18 03 02 02 77 32  ...3...1 0...w2
0130  30 30 33 2e 63 6c 61 73 73 64 65 6d 6f 2e 6c 6f  003.clas sdemo.lo
0140  63 61 6c 01 04 ff ff ff 00 0f 0f 63 6c 61 73 73  cal..... ...class
0150  64 65 6d 6f 2e 6c 6f 63 61 6c 03 04 c0 a8 01 01  demo.loc al.....
0160  06 04 c0 a8 01 01 ff  .....
  
```

Text item (), 6 bytes Packets: 42437 Displayed: 93 Marked: 0 Profile: Default

When this procedure is finished, then the client is allowed to use that ip-configuration until the end of its lease time.

68.2. picturing dhcp

Here we have a small network with two **dhcp servers** named DHCP-SRV1 and DHCP-SRV2 and two clients (SunWS1 and Mac42). All computers are connected by a hub or switch (pictured in the middle). All four computers have a cable to the hub (cables not pictured).



1. The client SunWS1 sends a **dhcp discover** on the network. All computers receive this broadcast.
2. Both **dhcp servers** answer with a **dhcp offer**. DHCP-SRV1 is a **dedicated dhcp server** and is faster in sending a **dhcp offer** than DHCP-SRV2 (who happens to also be a file server).
3. The client chooses the offer from DHCP-SRV1 and sends a **dhcp request** on the network.
4. DHCP-SRV1 answers with a **dhcp ack** (short for acknowledge).

All four broadcasts (or five when you count both offers) can be layer 2 ethernet broadcast to mac address **ff:ff:ff:ff:ff:ff** and a layer 3 ip broadcast to 255.255.255.255.

The same story can be read in **rfc 2131**.

68.3. installing a dhcp server

On Debian/Ubuntu

```
debian5:~# aptitude install dhcp3-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Reading task descriptions... Done
The following NEW packages will be installed:
  dhcp3-server
```

You get a configuration file with many examples.

```
debian5:~# ls -l /etc/dhcp3/dhcpd.conf
-rw-r--r-- 1 root root 3551 2011-04-10 21:23 /etc/dhcp3/dhcpd.conf
```

68.4. dhcp server on Red Hat

After installing we get a `/etc/dhcpd.conf` that points us to an example file named `dhcpd.conf.sample`.

```
[root@localhost ~]# cat /etc/dhcpd.conf
#
# DHCP Server Configuration file.
#   see /usr/share/doc/dhcp*/dhcpd.conf.sample
```

So we copy the sample and adjust it for our real situation. We name the copy `/etc/dhcpd.conf`.

```
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.140 192.168.1.159
  option routers          192.168.1.1;
  option subnet-mask     255.255.255.0;
  option domain-name     "classdemo.local";
  option domain-name-servers 192.168.1.1;
  default-lease-time    21600;
}
```

68.5. dhcp options

Options can be set on the global, scope, client-reservation level.

```
option subnet-mask 255.255.255.0;
option domain-name "linux-training.be";
option domain-name-servers "ns1.openminds.be";
option routers 192.168.42.1;
```

68.6. client reservations

You can reserve an ip configuration for a client using the mac address.

```
host pc42 {
hardware ethernet 11:22:33:44:55:66;
fixed-address 192.168.42.42;
}
```

You can add individual options to this reservation.

```
host pc42 {
hardware ethernet 11:22:33:44:55:66;
fixed-address 192.168.42.42;
option domain-name "linux-training.be";
option routers 192.168.42.1;
}
```

68.7. example config files

Below you see several sections of `/etc/dhcp/dhcpd.conf` on a **Debian 6** server.

```
# NetSec Antwerp Network
subnet 192.168.1.0 netmask 255.255.255.0 {
  range 192.168.1.20 192.168.1.199;
  option domain-name-servers ns1.netsec.local;
  option domain-name "netsec.local";
  option routers 192.168.1.1;
  option broadcast-address 192.168.1.255;
  default-lease-time 7200;
  max-lease-time 7200;
}
```

Above the general configuration for the network, with a pool of 180 addresses.

Below two client reservations:

```
#
# laptops
#

host mac {
  hardware ethernet 00:26:bb:xx:xx:xx;
  fixed-address mac.netsec.local;
}

host vmac {
  hardware ethernet 8c:7b:9d:xx:xx:xx;
  fixed-address vmac.netsec.local;
}
```

68.8. older example config files

For `dhcpd.conf` on Fedora with dynamic updates for a DNS domain.

```
[root@fedora14 ~]# cat /etc/dhcp/dhcpd.conf
authoritative;
include "/etc/rndc.key";

log-facility local6;

server-identifier    fedora14;
```

```
ddns-domainname "office.linux-training.be";
ddns-update-style interim;
ddns-updates on;
update-static-leases on;

option domain-name "office.linux-training.be";
option domain-name-servers 192.168.42.100;
option ip-forwarding off;

default-lease-time 1800;
max-lease-time 3600;

zone office.linux-training.be {
    primary 192.168.42.100;
}

subnet 192.168.4.0 netmask 255.255.255.0 {
    range 192.168.4.24 192.168.4.40;
}
```

Allowing any updates in the zone database (part of the named.conf configuration)

```
zone "office.linux-training.be" {
    type master;
    file "/var/named/db.office.linux-training.be";
    allow-transfer { any; };
    allow-update { any; };
};
```

Allowing secure key updates in the zone database (part of the named.conf configuration)

```
zone "office.linux-training.be" {
    type master;
    file "/var/named/db.office.linux-training.be";
    allow-transfer { any; };
    allow-update { key mykey; };
};
```

Sample key file contents:

```
[root@fedora14 ~]# cat /etc/rndc.key
key "rndc-key" {
    algorithm hmac-md5;
    secret "4Ykd58uIeUr3Ve6ad1qTfQ==";
};
```

Generate your own keys with **dnssec-keygen**.

How to include a key in a config file:

```
include "/etc/bind/rndc.key";
```

Also make sure that **bind** can write to your db.zone file (using `chmod/chown`). For Ubuntu this can be in `/etc/bind`, for Fedora in `/var/named`.

68.9. advanced dhcp

80/20 rule

DHCP servers should not be a single point of failure. Let us discuss redundant dhcp server setups (todo later).

relay agent

To avoid having to place a dhcp server on every segment, we can use **dhcp relay agents**.

rogue dhcp servers

Rogue dhcp servers are a problem without a solution. For example accidental connection of a (believed to be simple) hub/switch to a network with an internal dhcp server.

dhcp and ddns

DHCP can dynamically update DNS when it configures a client computer. DDNS can be used with or without secure keys.

When set up properly records can be added automaticall to the zone file:

```
root@fedora14~# tail -2 /var/named/db.office.linux-training.be
ubu1010srv          A      192.168.42.151
                   TXT    "00dfbb15e144a273c3cf2d6ae933885782"
```

68.10. Practice: DHCP and DDNS

1. Make sure you have a unique fixed ip address for your DNS and DHCP server (easier on the same machine).
2. Install DHCP and browse the explanation in the default configuration file `/etc/dhcp/dhcpd.conf` or `/etc/dhcp3/dhcpd.conf`.
3. Decide on a valid scope and activate it.
4. Test with a client that your DHCP server works.
5. Use wireshark to capture the four broadcasts when a client receives an ip (for the first time).
6. Use wireshark to capture a DHCPNAK and a DHCPrelease.
7. Reserve a configuration for a particular client (using mac address).
8. Configure your DHCP/DNS server(s) with a proper hostname and domainname (`/etc/hosts`, `/etc/hostname`, `/etc/sysconfig/network` on Fedora/RHEL, `/etc/resolv.conf` ...). You may need to disable NetworkManager on *buntu-desktops.
9. Make sure your DNS server still works, and is master over (at least) one domain.

There are several ways to do steps 10-11-12. Google is your friend in exploring DDNS with keys, with key-files or without keys.
10. Configure your DNS server to allow dynamic updates from your DHCP server.
11. Configure your DHCP server to send dynamic updates to your DNS server.
12. Test the working of Dynamic DNS.

Part XIX. dhcp server

Part XX. iptables firewall

Chapter 69. introduction to routers

Table of Contents

69.1. terminology	643
69.2. packet forwarding	644

69.1. terminology

router or firewall

A router is a device that connects two networks. A **firewall** is a device that besides acting as a **router**, also contains (and implements) rules to determine whether packets are allowed to travel from one network to another. A firewall can be configured to block access based on networks, hosts, protocols and ports. Firewalls can also change the contents of packets while forwarding them.

packet forwarding

Packet forwarding means allowing packets to go from one network to another. When a multihomed host is connected to two different networks, and it allows packets to travel from one network to another through its two network interfaces, it is said to have enabled **packet forwarding**.

packet filtering

Packet filtering is very similar to packet forwarding, but every packet is individually tested against rules that decide on allowing or dropping the packet. The rules are stored by iptables.

stateful

A **stateful** firewall is an advancement over stateless firewalls that inspect every individual packet. A stateful firewall will keep a table of active connections, and is knowledgeable enough to recognise when new connections are part of an active session. Linux iptables is a stateful firewall.

NAT (network address translation)

A **NAT** device is a router that is also changing the source and/or target ip-address in packets. It is typically used to connect multiple computers in a private address

range (rfc 1918) with the (public) internet. A NAT can hide private addresses from the internet.

It is important to understand that people and vendors do not always use the right term when referring to a certain type of NAT. Be sure you talk about the same thing. We can distinguish several types of NAT.

PAT (port address translation)

NAT often includes PAT. A **PAT** device is a router that is also changing the source and/or target tcp/udp port in packets. PAT is Cisco terminology and is used by **SNAT**, **DNAT**, **masquerading** and **port forwarding** in Linux. RFC 3022 calls it **NAPT** and defines the NAT/PAT combo as "traditional NAT". A device sold to you as a NAT-device will probably do NAT and PAT.

SNAT (source network address translation)

A **SNAT** device is changing the source ip-address when a packet passes our NAT. SNAT configuration with iptables includes a fixed target source address.

masquerading

Masquerading is a form of SNAT that will hide the (private) source ip-addresses of your private network using a public ip-address. Masquerading is common on dynamic internet interfaces (broadband modem/routers). Masquerade configuration with iptables uses a dynamic target source address.

DNAT (destination network address translation)

A **DNAT** device is changing the destination ip-address when a packet passes our NAT.

port forwarding

When static DNAT is set up in a way that allows outside connections to enter our private network, then we call it **port forwarding**.

69.2. packet forwarding

about packet forwarding

Packet forwarding means allowing packets to go from one network to another. When a multihomed host is connected to two different networks, and it allows packets to

travel from one network to another through its two network interfaces, it is said to have enabled packet forwarding.

/proc/sys/net/ipv4/ip_forward

Whether a host is forwarding packets is defined in **/proc/sys/net/ipv4/ip_forward**. The following screenshot shows how to enable packet forwarding on Linux.

```
[root@RHEL5 ~]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

The next command shows how to disable packet forwarding.

```
[root@RHEL5 ~]# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Use cat to check if packet forwarding is enabled.

```
[root@RHEL5 ~]# cat /proc/sys/net/ipv4/ip_forward
```

/etc/sysctl.conf

By default, most Linux computers are not configured for automatic packet forwarding. To enable packet forwarding whenever the system starts, change the **net.ipv4.ip_forward** variable in **/etc/sysctl.conf** to the value 1.

```
[root@RHEL5 ~]# grep ip_forward /etc/sysctl.conf
net.ipv4.ip_forward = 0
```

Practice: packet forwarding

1. Set up two dsl (Damn Small Linux) machines, one on vmnet1, the other on vmnet8. Make sure they both get an ip-address in the correct subnet. These two machines will be 'left' and 'right' from the 'router'.
2. Set up a RHEL server with two network cards, one on vmnet1, the other on vmnet8. This computer will be the 'router'. Complete the table below with the relevant names, ip-addresses and mac-addresses.

Table 69.1. Packet Forwarding Exercise

	left:	router:		right:
MAC				
IP				

3. How can you verify whether the RHEL will allow packet forwarding by default or not ? Test that you can ping from the RHEL to the two dsl machines, and from the two dsl machines to the RHEL. Use **arp -a** to make sure you are connected with the correct MAC addresses.

4. Ping from one dsl to the other. Enable and/or disable packet forwarding on the RHEL server and verify what happens to the ping between the two dsl machines. If you do not succeed in pinging between the two dsl machines (on different subnets), then use a sniffer like wireshark or tcpdump to discover the problem.

5. Use wireshark or tcpdump -xx to answer the following questions. Does the source MAC change when a packet passes through the filter ? And the destination MAC ? What about source and destination IP-addresses ?

Solution: packet forwarding

1. Set up two dsl (Damn Small Linux) machines, one on vmnet1, the other on vmnet8. Make sure they both get an ip-address in the correct subnet. These two machines will be 'left' and 'right' from the 'router'.

The configuration of the dsl machines can be similar to the following two screenshots. Both machines must be in a different subnet (here 192.168.187.0/24 and 172.16.122.0/24)

```
root@ttypl[root]# ifconfig eth0 | grep -A1 eth0
eth0 Link encap:Ethernet HWaddr 00:0C:29:08:F4:C1
      inet addr:192.168.187.130 Bcast:192.168.187.255 Mask:255.255.255.0
root@ttypl[root]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
192.168.187.0 * 255.255.255.0 U 0 0 0 eth0
default 192.168.187.128 0.0.0.0 UG 0 0 0 eth0
root@ttypl[root]#
```

```
root@ttypl[root]# ifconfig eth0 | grep -A1 eth0
eth0 Link encap:Ethernet HWaddr 00:0C:29:6E:1A:AA
      inet addr:172.16.122.129 Bcast:172.16.122.255 Mask:255.255.255.0
root@ttypl[root]# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
172.16.122.0 * 255.255.255.0 U 0 0 0 eth0
default 172.16.122.128 0.0.0.0 UG 0 0 0 eth0
root@ttypl[root]#
```

2. Set up a RHEL server with two network cards, one on vmnet1, the other on vmnet8. This computer will be the 'router'.

The 'router' can be set up like this screenshot shows.

```
[root@RHEL5 ~]# ifconfig | grep -A1 eth
eth1 Link encap:Ethernet HWaddr 00:0C:29:8C:90:49
      inet addr:192.168.187.128 Bcast:192.168.187.255 Mask:255.255.255.0
--
eth2 Link encap:Ethernet HWaddr 00:0C:29:8C:90:53
      inet addr:172.16.122.128 Bcast:172.16.122.255 Mask:255.255.255.0
[root@RHEL5 ~]#
```

Your setup may use different ip and mac addresses than the ones in the table below. This table serves as a reference for the screenshots from this solution to the practice.

Table 69.2. Packet Forwarding Solution

left: dsl	router: RHEL5		right: dsl
00:0c:29:08:f4:c1	00:0c:29:8c:90:49	00:0c:29:8c:90:53	00:0c:29:6e:1a:aa
192.168.187.130	192.168.187.128	172.16.122.128	172.16.122.129

3. How can you verify whether the RHEL will allow packet forwarding by default or not ? Test that you can ping from the RHEL to the two dsl machines, and from the two dsl machines to the RHEL. Use **arp -a** to make sure you are connected with the correct MAC addresses.

This can be done with "**grep ip_forward /etc/sysctl.conf**" (1 is enabled, 0 is disabled).

```
[root@RHEL5 ~]# grep ip_for /etc/sysctl.conf
net.ipv4.ip_forward = 0
```

4. Ping from one dsl to the other. Enable and/or disable packet forwarding on the RHEL server and verify what happens to the ping between the two dsl machines. If you do not succeed in pinging between the two dsl machines (on different subnets), then use a sniffer like ethereal or tcpdump to discover the problem.

Did you forget to add a default gateway to the dsl machines ? Use **route add default gw 'ip-address'**.

You should be able to ping when packet forwarding is enabled (and both default gateways are properly configured). The ping will not work when packet forwarding is disabled or when gateways are not configured correctly.

5. Use wireshark or tcpdump -xx to answer the following questions. Does the source MAC change when a packet passes through the filter ? And the destination MAC ? What about source and destination IP-addresses ?

Both MAC addresses are changed when passing the router. The screenshots below show tcpdump -xx output on the router. The first one is taken on the eth1(vmnet1) interface in the 192.168.187.0/24 network, the second one is from the other interface (eth2 on vmnet8 in 172.16.122.0/24). The first six bytes are the destination MAC, the next six are the source.

```
[root@RHEL5 ~]# tcpdump -xx -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
04:18:23.817854 IP 192.168.187.130 > 172.16.122.129: ICMP echo request...
 0x0000: 000c 298c 9049 000c 2908 f4c1 0800 4500
 0x0010: 0054 0000 4000 4001 97ec c0a8 bb82 ac10
 0x0020: 7a81 0800 3b28 a717 0006 8059 d148 d614
 0x0030: 0300 0809 0a0b 0c0d 0e0f 1011 1213 1415
 0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
 0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
04:18:23.817962 IP 172.16.122.129 > 192.168.187.130: ICMP echo reply...
 0x0000: 000c 2908 f4c1 000c 298c 9049 0800 4500
 0x0010: 0054 d364 0000 3f01 0588 ac10 7a81 c0a8
 0x0020: bb82 0000 4328 a717 0006 8059 d148 d614
 0x0030: 0300 0809 0a0b 0c0d 0e0f 1011 1213 1415
 0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
 0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
```

```
[root@RHEL5 ~]# tcpdump -xx -i eth2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2, link-type EN10MB (Ethernet), capture size 96 bytes
04:18:33.904697 IP 192.168.187.130 > 172.16.122.129: ICMP echo request...
 0x0000: 000c 296e 1aaa 000c 298c 9053 0800 4500
 0x0010: 0054 0000 4000 3f01 98ec c0a8 bb82 ac10
 0x0020: 7a81 0800 2320 a717 0008 8a59 d148 e41a
 0x0030: 0300 0809 0a0b 0c0d 0e0f 1011 1213 1415
 0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
 0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
04:18:33.944514 IP 172.16.122.129 > 192.168.187.130: ICMP echo reply...
 0x0000: 000c 298c 9053 000c 296e 1aaa 0800 4500
 0x0010: 0054 d366 0000 4001 0486 ac10 7a81 c0a8
 0x0020: bb82 0000 2b20 a717 0008 8a59 d148 e41a
 0x0030: 0300 0809 0a0b 0c0d 0e0f 1011 1213 1415
 0x0040: 1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
 0x0050: 2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
```

Chapter 70. Firewall: iptables

Table of Contents

70.1. about iptables	649
70.2. packet filtering	649
70.3. network address translation	654

70.1. about iptables

Iptables is a user-space application that allows a user to configure the Linux kernel's Netfilter. By default there are three tables in the kernel that contain sets of rules. The filter table is used for packet filtering, the NAT table for address translation and the mangle table for special-purpose processing of packets. Series of rules in each table are called a **chain**.

The following screenshot shows how to stop and start iptables.

```
[root@RHEL5 ~]# /etc/init.d/iptables stop
[root@RHEL5 ~]# /etc/init.d/iptables start
[root@RHEL5 ~]#
```

70.2. packet filtering

about packet filtering

Packet filtering is a bit more than packet forwarding. Packet forwarding only uses a routing table to make decisions, the kernel now also uses a list of rules. So with packet filtering, the kernel will inspect each packet and decide based on iptables rules to allow or drop a packet.

filter table

The filter table in iptables has three chains (sets of rules). The INPUT chain is used for any packet coming into the system. The OUTPUT chain is for any packet leaving the system. And the FORWARD chain is for packets that are forwarded (routed) through the system.

The screenshot below shows how to list the filter table and all its rules.

```
[root@RHEL5 ~]# iptables -t filter -nL
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
```

```
Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
[root@RHEL5 ~]#
```

As you can see, all three chains in the filter table are set to ACCEPT everything. ACCEPT is the default behaviour.

Changing default policy rules

To start, let's set the default policy for all three chains to drop everything. Note that you might lose your connection when typing this over ssh ;-).

```
[root@RHEL5 ~]# iptables -P INPUT DROP
[root@RHEL5 ~]# iptables -P FORWARD DROP
[root@RHEL5 ~]# iptables -P OUTPUT DROP
```

Next, we allow the server to use its own loopback device (this allows the server to access its services running on localhost). We first append a rule to the INPUT chain to allow (ACCEPT) traffic from the lo (loopback) interface, then we do the same to allow packets to leave the system through the loopback interface.

```
[root@RHEL5 ~]# iptables -A INPUT -i lo -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o lo -j ACCEPT
```

Looking at the filter table again (omitting -t filter because it is the default table).

```
[root@RHEL5 ~]# iptables -nL
Chain INPUT (policy DROP)
target     prot opt source                destination
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0

Chain FORWARD (policy DROP)
target     prot opt source                destination

Chain OUTPUT (policy DROP)
target     prot opt source                destination
ACCEPT     all  --  0.0.0.0/0             0.0.0.0/0
```

Allowing ssh over eth0

This example show how to add two rules to allow ssh access to your system from outside.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT
```


The filter table will look something like this screenshot (note that `-v` is added for more verbose output).

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
  pkts bytes target prot opt in     out     source        destination
     0     0 ACCEPT all  --  lo      *        0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT tcp  --  eth0    *        0.0.0.0/0    0.0.0.0/0  tcp dpt:22

Chain FORWARD (policy DROP 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source        destination

Chain OUTPUT (policy DROP 3 packets, 228 bytes)
  pkts bytes target prot opt in     out     source        destination
     0     0 ACCEPT all  --  *       lo      0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT tcp  --  *       eth0    0.0.0.0/0    0.0.0.0/0  tcp spt:22
[root@RHEL5 ~]#
```

Allowing access from a subnet

This example shows how to allow access from any computer in the 10.1.1.0/24 network, but only through eth1. There is no port (application) limitation here.

```
[root@RHEL5 ~]# iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```

Together with the previous examples, the policy is expanding.

```
[root@RHEL5 ~]# iptables -nvL
Chain INPUT (policy DROP 7 packets, 609 bytes)
  pkts bytes target prot opt in     out     source        destination
     0     0 ACCEPT all  --  lo      *        0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT tcp  --  eth0    *        0.0.0.0/0    0.0.0.0/0  tcp dpt:22
     0     0 ACCEPT tcp  --  eth1    *        10.1.1.0/24  0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
  pkts bytes target prot opt in     out     source        destination

Chain OUTPUT (policy DROP 3 packets, 228 bytes)
  pkts bytes target prot opt in     out     source        destination
     0     0 ACCEPT all  --  *       lo      0.0.0.0/0    0.0.0.0/0
     0     0 ACCEPT tcp  --  *       eth0    0.0.0.0/0    0.0.0.0/0  tcp spt:22
     0     0 ACCEPT tcp  --  *       eth1    0.0.0.0/0    10.1.1.0/24
```

iptables save

Use **iptables save** to automatically implement these rules when the firewall is (re)started.

```
[root@RHEL5 ~]# /etc/init.d/iptables save
Saving firewall rules to /etc/sysconfig/iptables:          [ OK ]
[root@RHEL5 ~]#
```

scripting example

You can write a simple script for these rules. Below is an example script that implements the firewall rules that you saw before in this chapter.

```
#!/bin/bash
# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X

# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# allow ssh over eth0 from outside to system
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT

# allow any traffic from 10.1.1.0/24 to system
iptables -A INPUT -i eth1 -s 10.1.1.0/24 -p tcp -j ACCEPT
iptables -A OUTPUT -o eth1 -d 10.1.1.0/24 -p tcp -j ACCEPT
```

Allowing ICMP(ping)

When you enable iptables, you will get an '**Operation not permitted**' message when trying to ping other hosts.

```
[root@RHEL5 ~]# ping 192.168.187.130
PING 192.168.187.130 (192.168.187.130) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
ping: sendmsg: Operation not permitted
```

The screenshot below shows you how to setup iptables to allow a ping from or to your machine.

```
[root@RHEL5 ~]# iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
[root@RHEL5 ~]# iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT
```

The previous two lines do not allow other computers to route ping messages through your router, because it only handles INPUT and OUTPUT. For routing of ping, you will need to enable it on the FORWARD chain. The following command enables routing of icmp messages between networks.

```
[root@RHEL5 ~]# iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
```

Practice: packet filtering

1. Make sure you can ssh to your router-system when iptables is active.
2. Make sure you can ping to your router-system when iptables is active.
3. Define one of your networks as 'internal' and the other as 'external'. Configure the router to allow visits to a website (http) to go from the internal network to the external network (but not in the other direction).
4. Make sure the internal network can ssh to the external, but not the other way around.

Solution: packet filtering

A possible solution, where dsl is the internal and dsr is the external network.

```
#!/bin/bash

# first cleanup everything
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X

# default drop
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# allow loopback device
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# question 1: allow ssh over eth0
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT

# question 2: Allow icmp(ping) anywhere
iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type any -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type any -j ACCEPT

# question 3: allow http from internal(dsl) to external(dsr)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 80 -j ACCEPT

# question 4: allow ssh from internal(dsl) to external(dsr)
iptables -A FORWARD -i eth1 -o eth2 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth2 -o eth1 -p tcp --sport 22 -j ACCEPT

# allow http from external(dsr) to internal(dsl)
# iptables -A FORWARD -i eth2 -o eth1 -p tcp --dport 80 -j ACCEPT
# iptables -A FORWARD -i eth1 -o eth2 -p tcp --sport 80 -j ACCEPT

# allow rpcinfo over eth0 from outside to system
```

```
# iptables -A INPUT -i eth2 -p tcp --dport 111 -j ACCEPT
# iptables -A OUTPUT -o eth2 -p tcp --sport 111 -j ACCEPT
```

70.3. network address translation

about NAT

A NAT device is a router that is also changing the source and/or target ip-address in packets. It is typically used to connect multiple computers in a private address range with the (public) internet. A NAT can hide private addresses from the internet.

NAT was developed to mitigate the use of real ip addresses, to allow private address ranges to reach the internet and back, and to not disclose details about internal networks to the outside.

The nat table in iptables adds two new chains. PREROUTING allows altering of packets before they reach the INPUT chain. POSTROUTING allows altering packets after they exit the OUTPUT chain.

Use **iptables -t nat -nL** to look at the NAT table. The screenshot below shows an empty NAT table.

```
[root@RHEL5 ~]# iptables -t nat -nL
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
[root@RHEL5 ~]#
```

SNAT (Source NAT)

The goal of source nat is to change the source address inside a packet before it leaves the system (e.g. to the internet). The destination will return the packet to the NAT-device. This means our NAT-device will need to keep a table in memory of all the packets it changed, so it can deliver the packet to the original source (e.g. in the private network).

Because SNAT is about packets leaving the system, it uses the POSTROUTING chain.

Here is an example SNAT rule. The rule says that packets coming from 10.1.1.0/24 network and exiting via eth1 will get the source ip-address set to 11.12.13.14. (Note that this is a one line command!)

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
```

Of course there must exist a proper iptables filter setup to allow the packet to traverse from one network to the other.

SNAT example setup

This example script uses a typical nat setup. The internal (eth0) network has access via SNAT to external (eth1) webservers (port 80).

```
#!/bin/bash
#
# iptables script for simple classic nat websurfing
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -p tcp \
--dport 80 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -d 10.1.1.0/24 -p tcp \
--sport 80 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j SNAT \
--to-source 11.12.13.14
echo 1 > /proc/sys/net/ipv4/ip_forward
```

IP masquerading

IP masquerading is very similar to SNAT, but is meant for dynamic interfaces. Typical example are broadband 'router/modems' connected to the internet and receiving a different ip-address from the isp, each time they are cold-booted.

The only change needed to convert the SNAT script to a masquerading is one line.

```
iptables -t nat -A POSTROUTING -o eth1 -s 10.1.1.0/24 -j MASQUERADE
```

DNAT (Destination NAT)

DNAT is typically used to allow packets from the internet to be redirected to an internal server (in your DMZ) and in a private address range that is inaccessible directly from the internet.

This example script allows internet users to reach your internal (192.168.1.99) server via ssh (port 22).

```
#!/bin/bash
```

```
#
# iptables script for DNAT
# eth0 is internal network, eth1 is internet
#
echo 0 > /proc/sys/net/ipv4/ip_forward
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
iptables -A FORWARD -i eth0 -o eth1 -s 10.1.1.0/24 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 -j ACCEPT
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 22 \
-j DNAT --to-destination 10.1.1.99
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Part XXI. apache and squid

Chapter 71. introduction to apache

Table of Contents

71.1. about apache	658
71.2. is apache installed ?	658
71.3. is apache running ?	659
71.4. apache configuration	659
71.5. virtual hosts	660
71.6. aliases and redirects	661
71.7. securing directories with htpasswd and .htaccess	661
71.8. more on .htaccess	662
71.9. traffic	662
71.10. practice: apache	662

71.1. about apache

According to NetCraft (http://news.netcraft.com/archives/web_server_survey.html) about seventy percent of all web servers are running on Apache. Some people say that the name is derived from a **patchy** web server, because of all the patches people wrote for the NCSA httpd server.

71.2. is apache installed ?

To verify whether Apache is installed, use the proper tools (rpm, dpkg, ...) and grep for apache or httpd.

This Red Hat Enterprise 4 Server has apache installed.

```
[paul@rhel4 ~]$ rpm -qa | grep -i httpd
httpd-2.0.52-25.ent
httpd-manual-2.0.52-25.ent
system-config-httpd-1.3.1-1
httpd-devel-2.0.52-25.ent
httpd-suexec-2.0.52-25.ent
```

This Ubuntu also has apache installed.

```
paul@laika:~$ dpkg -l | grep apache
ii  apache2                2.2.3-3.2build1      Next generation, scalable, ...
ii  apache2-mpm-prefork    2.2.3-3.2build1      Traditional model for Apach...
ii  apache2-utils          2.2.3-3.2build1      utility programs for webser...
ii  apache2.2-common       2.2.3-3.2build1      Next generation, scalable, ...
ii  libapache2-mod-php5    5.2.1-0ubuntu1.2     server-side, HTML-embedded ...
```


71.3. is apache running ?

This is how apache looks when it is installed on Red Hat Enterprise Linux 4, running named as **httpd**.

```
[root@RHELv4u3 ~]# /etc/init.d/httpd status
httpd is stopped
[root@RHELv4u3 ~]# service httpd start
Starting httpd: [ OK ]
[root@RHELv4u3 ~]# ps -C httpd
PID TTY          TIME CMD
4573 ?            00:00:00 httpd
4576 ?            00:00:00 httpd
4577 ?            00:00:00 httpd
4578 ?            00:00:00 httpd
4579 ?            00:00:00 httpd
4580 ?            00:00:00 httpd
4581 ?            00:00:00 httpd
4582 ?            00:00:00 httpd
4583 ?            00:00:00 httpd
[root@RHELv4u3 ~]#
```

And here is Apache running on Ubuntu, named as **apache2**.

```
root@laika:~# ps -C apache2
PID TTY          TIME CMD
6170 ?            00:00:00 apache2
6248 ?            00:00:01 apache2
6249 ?            00:00:01 apache2
6250 ?            00:00:00 apache2
6251 ?            00:00:01 apache2
6252 ?            00:00:01 apache2
7520 ?            00:00:01 apache2
8943 ?            00:00:01 apache2
root@laika:~# /etc/init.d/apache2 status
* Usage: /etc/init.d/apache2 {start|stop|restart|reload|force-reload}
root@laika:~#
```

To verify that apache is running, open a web browser on the web server, and browse to <http://localhost>. An Apache test page should be shown. The <http://localhosts/manual> url will give you an extensive Apache manual. The second test is to connect to your Apache from another computer.

71.4. apache configuration

Configuring Apache changed a bit the past couple of years. But it still takes place in **/etc/httpd** or **/etc/apache**.

```
[root@RHELv4u3 ~]# cd /etc/httpd/
[root@RHELv4u3 httpd]# ll
total 32
lrwxrwxrwx  1 root root   25 Jan 24 09:28 build -> ../../usr/lib/httpd/build
drwxr-xr-x  7 root root 4096 Jan 24 08:48 conf
```

```
drwxr-xr-x  2 root root 4096 Jan 24 09:29 conf.d
lrwxrwxrwx  1 root root   19 Jan 24 08:48 logs -> ../../var/log/httpd
lrwxrwxrwx  1 root root   27 Jan 24 08:48 modules -> ../../usr/lib/httpd/modules
lrwxrwxrwx  1 root root   13 Jan 24 08:48 run -> ../../var/run
[root@RHELv4u3 httpd]#
```

The main configuration file for the Apache server on RHEL is `/etc/httpd/conf/httpd.conf`, on Ubuntu it is `/etc/apache2/apache2.conf`. The file explains itself, and contains examples for how to set up virtual hosts or configure access.

71.5. virtual hosts

Virtual hosts can be defined by ip-address, by port or by name (host record). (The new way of defining virtual hosts is through separate config files in the conf.d directory.) Below is a very simple virtual host definition.

```
[root@rhel4 conf]# tail /etc/httpd/conf/httpd.conf
#
# This is a small test website
#
<VirtualHost testsite.local:80>
ServerAdmin webmaster@testsite.local
DocumentRoot /var/www/html/testsite/
ServerName testsite.local
ErrorLog logs/testsite.local-error_log
CustomLog logs/testsite.local-access_log common
</VirtualHost>
[root@rhel4 conf]#
```

Should you put this little index.html file in the directory mentioned in the above screenshot, then you can access this humble website.

```
[root@rhel4 conf]# cat /var/www/html/testsite/index.html
<html>
  <head><title>Test Site</title></head>
  <body>
    <p>This is the test site.</p>
  </body>
</html>
```

Below is a sample virtual host configuration. This virtual hosts overrules the default Apache **ErrorDocument** directive.

```
<VirtualHost 83.217.76.245:80>
ServerName cobbaut.be
ServerAlias www.cobbaut.be
DocumentRoot /home/paul/public_html
ErrorLog /home/paul/logs/error_log
CustomLog /home/paul/logs/access_log common
ScriptAlias /cgi-bin/ /home/paul/cgi-bin/
<Directory /home/paul/public_html>
  Options Indexes IncludesNOEXEC FollowSymLinks
  allow from all
```

```
</Directory>
ErrorDocument 404 http://www.cobbaut.be/cobbaut.php
</VirtualHost>
```

71.6. aliases and redirects

Apache supports aliases for directories, like this example shows.

```
Alias /paul/ "/home/paul/public_html/"
```

Similarly, content can be redirected to another website or web server.

```
Redirect permanent /foo http://www.foo.com/bar
```

71.7. securing directories with htpasswd and .htaccess

You can secure files and directories in your website with a userid/password. First, enter your website, and use the **htpasswd** command to create a **.htpasswd file** that contains a userid and an (encrypted) password.

```
[root@rhel4 testsite]# htpasswd -c .htpasswd pol
New password:
Re-type new password:
Adding password for user pol
[root@rhel4 testsite]# cat .htpasswd
pol:x5vZlyw1V6KXE
[root@rhel4 testsite]#
```

You can add users to this file, just don't use the -c switch again.

```
[root@rhel4 testsite]# htpasswd .htpasswd kim
New password:
Re-type new password:
Adding password for user kim
[root@rhel4 testsite]# cat .htpasswd
pol:x5vZlyw1V6KXE
kim:6/RbvugwsgOI6
[root@rhel4 testsite]#
```

You have now defined two users. Next create a subdirectory that you want to protect with these two accounts. And put the following .htaccess file in that subdirectory.

```
[root@rhel4 kimonly]# pwd
/var/www/html/testsite/kimonly
[root@rhel4 kimonly]# cat .htaccess
AuthUserFile /var/www/html/testsite/.htpasswd
AuthGroupFile /dev/null
AuthName "test access title"
AuthType Basic
```

```
<Limit GET POST>
require valid-user
</Limit>
[root@rhel4 kimonly]#
```

Finally, don't forget to verify that `AllowOverride` is set to `All` in the general Apache configuration file.

```
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
AllowOverride All
```

From now on, when a user accesses a file in that subdirectory, that user will have to provide a `userid/password` combo that is defined in your `.htpasswd`.

71.8. more on .htaccess

You can do much more with `.htaccess`. One example is to use `.htaccess` to prevent people from certain domains to access your website. Like in this case, where a number of referer spammers are blocked from the website.

```
paul@lounge:~/cobbaut.be$ cat .htaccess
# Options +FollowSymlinks
RewriteEngine On
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-adipex.fw.nu.*$ [OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-levitra.asso.ws.*$ [NC,OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-tramadol.fw.nu.*$ [NC,OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-viagra.lookin.at.*$ [NC,OR]
...
RewriteCond %{HTTP_REFERER} ^http://(www\.)?www.healthinsurancehelp.net.*$ [NC]
RewriteRule .* - [F,L]
paul@lounge:~/cobbaut.be$
```

71.9. traffic

Apache keeps a log of all visitors. The **webalizer** is often used to parse this log into nice html statistics.

71.10. practice: apache

1. Verify that Apache is installed and running.
2. Browse to the Apache HTML manual from another computer.
3. Create a virtual hosts that listens to port 8247.

4. Create a virtual hosts that listens on another ip-address.
5. Test from another computer that all virtual hosts work.
6. Protect a subdirectory of a website with `.htpasswd` and `.htaccess`.

Chapter 72. introduction to squid

Table of Contents

72.1. about proxy servers	664
72.2. squid proxy server	665

72.1. about proxy servers

usage

A **proxy server** is a server that caches the internet. Clients connect to the proxy server with a request for an internet server. The proxy server will connect to the internet server on behalf of the client. The proxy server will also cache the pages retrieved from the internet server. A proxy server may provide pages from his cache to a client, instead of connecting to the internet server to retrieve the (same) pages.

A proxy server has two main advantages. It improves web surfing speed when returning cached data to clients, and it reduces the required bandwidth (cost) to the internet.

Smaller organizations sometimes put the proxy server on the same physical computer that serves as a NAT to the internet. In larger organizations, the proxy server is one of many servers in the DMZ.

When web traffic passes via a proxy server, it is common practice to configure the proxy with extra settings for access control. Access control in a proxy server can mean user account access, but also website(url), ip-address or dns restrictions.

open proxy servers

You can find lists of open proxy servers on the internet that enable you to surf anonymously. This works when the proxy server connects on your behalf to a website, without logging your ip-address. But be careful, these (listed) open proxy servers could be created in order to eavesdrop upon their users.

squid

This chapter is an introduction to the **squid** proxy server (<http://www.squid-cache.org>). The version used is 2.5.

```
[root@RHEL4 ~]# rpm -qa | grep squid
squid-2.5.STABLE6-3.4E.12
[root@RHEL4 ~]#
```

72.2. squid proxy server

`/etc/squid/squid.conf`

Squid's main configuration file is `/etc/squid/squid.conf`. The file explains every parameter in great detail. It can be a good idea to start by creating a backup of this file.

```
[root@RHEL4 /etc/squid/]# cp squid.conf squid.conf.original
```

`/var/spool/squid`

The **squid** proxy server stores its cache by default in `/var/spool/squid`. This setting is configurable in `/etc/squid/squid.conf`.

```
[root@RHEL4 ~]# grep "^# cache_dir" /etc/squid/squid.conf
# cache_dir ufs /var/spool/squid 100 16 256
```

It is possible that in a default setup where squid has never run, that the `/var/spool/squid` directories do not exist.

```
[root@RHEL4 ~]# ls -al /var/spool/squid
ls: /var/spool/squid: No such file or directory
```

Running **squid -z** will create the necessary squid directories.

```
[root@RHEL4 ~]# squid -z
2008/09/22 14:07:47| Creating Swap Directories
[root@RHEL4 ~]# ls -al /var/spool/squid
total 80
drwxr-x---  18 squid squid 4096 Sep 22 14:07 .
drwxr-xr-x  26 root  root  4096 May 30  2007 ..
drwxr-xr-x 258 squid squid 4096 Sep 22 14:07 00
drwxr-xr-x 258 squid squid 4096 Sep 22 14:07 01
drwxr-xr-x 258 squid squid 4096 Sep 22 14:07 02
...
```

port 3128 or port 8080

By default the squid proxy server will bind to port 3128 to listen to incoming requests.

```
[root@RHEL4 ~]# grep "default port" /etc/squid/squid.conf
#       The default port number is 3128.
```

Many organizations use port 8080 instead.

```
[root@RHEL4 ~]# grep 8080 /etc/squid/squid.conf
http_port 8080
```

/var/log/squid

The standard log file location for squid is **/var/log/squid**.

```
[root@RHEL4 ~]# grep "/var/log" /etc/squid/squid.conf
# cache_access_log /var/log/squid/access.log
# cache_log /var/log/squid/cache.log
# cache_store_log /var/log/squid/store.log
```

access control

The default squid setup only allows localhost access. To enable access for a private network range, look for the "INSERT YOUR OWN RULE(S) HERE..." sentence in squid.conf and add two lines similar to the screenshot below.

```
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS

acl company_network src 192.168.1.0/24
http_access allow company_network
```

Restart the squid server, and now the local private network can use the proxy cache.

testing squid

First, make sure that the server running squid has access to the internet.

```
[root@RHEL4 ~]# wget -q http://linux-training.be/index.html
[root@RHEL4 ~]# ls -l index.html
-rw-r--r-- 1 root root 2269 Sep 18 13:18 index.html
[root@RHEL4 ~]#
```

Then configure a browser on a client to use the proxy server. OR you could set the HTTP_PROXY (sometimes http_proxy) variable to point command line programs to the proxy.

```
[root@fedora ~]# export HTTP_PROXY=http://192.168.1.39:8080
[root@ubuntu ~]# export http_proxy=http://192.168.1.39:8080
```

Testing a client machine can then be done with wget (wget -q is used to simplify the screenshot).


```
[root@RHEL5 ~]# > /etc/resolv.conf
[root@RHEL5 ~]# wget -q http://www.linux-training.be/index.html
[root@RHEL5 ~]# ls -l index.html
-rw-r--r-- 1 root root 2269 Sep 18 2008 index.html
[root@RHEL5 ~]#
```

name resolution

You need name resolution working on the squid server, but you don't need name resolution on the clients.

```
[paul@RHEL5 ~]$ wget http://grep.be
--14:35:44-- http://grep.be
Resolving grep.be... failed: Temporary failure in name resolution.
[paul@RHEL5 ~]$ export http_proxy=http://192.168.1.39:8080
[paul@RHEL5 ~]$ wget http://grep.be
--14:35:49-- http://grep.be/
Connecting to 192.168.1.39:8080... connected.
Proxy request sent, awaiting response... 200 OK
Length: 5390 (5.3K) [text/html]
Saving to: `index.html.1'

100%[=====>] 5,390      --.-K/s   in 0.1s

14:38:29 (54.8 KB/s) - `index.html' saved [5390/5390]

[paul@RHEL5 ~]$
```

Part XXII. introducing git

Chapter 73. git

Table of Contents

73.1. practice: git	670
---------------------------	-----

This chapter is an introduction to using **git** on the command line. The **git repository** is hosted by **github**, but you are free to choose another server (or create your own).

There are many excellent online tutorials for **git**. This list can save you one Google query:

<http://gitimmersion.com/>
<http://git-scm.com/book>

73.1. practice: git

1. Create a project on github to host a script that you wrote. Have at least two other people improve the script.

Part XXIII. ipv6

Chapter 74. Introduction to ipv6

Table of Contents

74.1. about ipv6	673
74.2. network id and host id	673
74.3. host part generation	673
74.4. ipv4 mapped ipv6 address	674
74.5. link local addresses	674
74.6. unique local addresses	674
74.7. globally unique unicast addresses	674
74.8. 6to4	675
74.9. ISP	675
74.10. non routable addresses	675
74.11. ping6	675
74.12. Belgium and ipv6	676
74.13. other websites	676
74.14. 6to4 gateways	678
74.15. ping6 and dns	678
74.16. ipv6 and tcp/http	678
74.17. ipv6 PTR record	678
74.18. 6to4 setup on Linux	678

74.1. about ipv6

The **ipv6** protocol is designed to replace **ipv4**. Where **ip version 4** supports a maximum of four billion unique addresses, **ip version 6** expands this to **four billion times four billion times four billion times four billion** unique addresses. This is more than 100.000.000.000.000.000.000 ipv6 addresses per square cm on our planet. That should be enough, even if every cell phone, every coffee machine and every pair of socks gets an address.

Technically speaking ipv6 uses 128-bit addresses (instead of the 32-bit from ipv4). 128-bit addresses are **huge** numbers. In decimal it would amount up to 39 digits, in hexadecimal it looks like this:

```
fe80:0000:0000:0000:0a00:27ff:fe8e:8aa8
```

Luckily ipv6 allows us to omit leading zeroes. Our address from above then becomes:

```
fe80:0:0:0:a00:27ff:fe8e:8aa8
```

When a 16-bit block is zero, it can be written as **::**. Consecutive 16-bit blocks that are zero can also be written as **::**. So our address can from above can be shortened to:

```
fe80::a00:27ff:fe8e:8aa8
```

This **::** can only occur once! The following is not a valid ipv6 address:

```
fe80::20:2e4f::39ac
```

The ipv6 **localhost** address is **0000:0000:0000:0000:0000:0000:0000:0001**, which can be abbreviated to **::1**.

```
paul@debian5:~/github/lt/images$ /sbin/ifconfig lo | grep inet6
inet6 addr: ::1/128 Scope:Host
```

74.2. network id and host id

One of the few similarities between ipv4 and ipv6 is that addresses have a host part and a network part determined by a subnet mask. Using the **cidr** notation this looks like this:

```
fe80::a00:27ff:fe8e:8aa8/64
```

The above address has 64 bits for the host id, theoretically allowing for 4 billion times four billion hosts.

The localhost address looks like this with cidr:

```
::1/128
```

74.3. host part generation

The host part of an automatically generated (stateless) ipv6 address contains part of the hosts mac address:

```
paul@debian5:~$ /sbin/ifconfig | head -3
eth3      Link encap:Ethernet  HWaddr 08:00:27:ab:67:30
          inet addr:192.168.1.29  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feab:6730/64 Scope:Link
```

Some people are concerned about privacy here...

74.4. ipv4 mapped ipv6 address

Some applications use ipv4 addresses embedded in an ipv6 address. (Yes there will be an era of migration with both ipv4 and ipv6 in use.) The ipv6 address then looks like this:

```
::ffff:192.168.1.42/96
```

Indeed a mix of decimal and hexadecimal characters...

74.5. link local addresses

ipv6 addresses starting with **fe8.** can only be used on the local segment (replace the dot with an hexadecimal digit). This is the reason you see **Scope:Link** behind the address in this screenshot. This address serves only the **local link**.

```
paul@deb503:~$ /sbin/ifconfig | grep inet6
inet6 addr: fe80::a00:27ff:fe8e:8aa8/64 Scope:Link
inet6 addr: ::1/128 Scope:Host
```

These **link local** addresses all begin with **fe8.**

Every ipv6 enabled nic will get an address in this range.

74.6. unique local addresses

The now obsolete system of **site local addresses** similar to ipv4 private ranges is replaced with a system of globally unique local ipv6 addresses. This to prevent duplicates when joining of networks within **site local** ranges.

All **unique local** addresses start with **fd.**

74.7. globally unique unicast addresses

Since **ipv6** was designed to have multiple ip addresses per interface, the **global ipv6 address** can be used next to the **link local address**.

These **globally unique** addresses all begin with **2...** or **3...** as the first 16-bits.

74.8. 6to4

6to4 is defined in rfc's 2893 and 3056 as one possible way to transition between ipv4 and ipv6 by creating an ipv6 tunnel.

It encodes an ipv4 address in an ipv6 address that starts with **2002**. For example 192.168.1.42/24 will be encoded as:

```
2002:c0a8:12a:18::1
```

You can use the command below to convert any ipv4 address to this range.

```
paul@ubul010:~$ printf "2002:%02x%02x:%02x%02x:%04x::1\n" `echo 192.168.1.42/24 \
|tr "/" " "`
2002:c0a8:012a:0018::1
```

74.9. ISP

Should you be so lucky to get an ipv6 address from an **isp**, then it will start with **2001**:

74.10. non routable addresses

Comparable to **example.com** for DNS, the following ipv6 address ranges are reserved for examples, and not routable on the internet.

```
3fff:ffff::/32
2001:0db8::/32
```

74.11. ping6

Use **ping6** to test connectivity between ipv6 hosts. You need to specify the interface (there is no routing table for 'random' generated ipv6 link local addresses).

```
[root@fedora14 ~]# ping6 -I eth0 fe80::a00:27ff:fe3c:4346
PING fe80::a00:27ff:fe3c:4346(fe80::a00:27ff:fe3c:4346) from fe80::a00:27ff:fe3c:4346 et
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=1 ttl=64 time=0.586 ms
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=2 ttl=64 time=3.95 ms
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=3 ttl=64 time=1.53 ms
```

Below a multicast ping6 that receives replies from three ip6 hosts on the same network.

```
[root@fedora14 ~]# ping6 -I eth0 ff02::1
PING ff02::1(ff02::1) from fe80::a00:27ff:fe3c:4346 eth0: 56 data bytes
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=1 ttl=64 time=0.598 ms
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=1 ttl=64 time=1.87 ms (DUP!)
64 bytes from fe80::8e7b:9dff:fed6:dff2: icmp_seq=1 ttl=64 time=535 ms (DUP!)
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=2 ttl=64 time=0.106 ms
64 bytes from fe80::8e7b:9dff:fed6:dff2: icmp_seq=2 ttl=64 time=1.79 ms (DUP!)
64 bytes from fe80::a00:27ff:fe3c:4346: icmp_seq=2 ttl=64 time=2.48 ms (DUP!)
```

74.12. Belgium and ipv6

A lot of information on ipv6 in Belgium can be found at www.ipv6council.be.

Sites like ipv6.belgium.be, www.bipt.be and www.bricozone.be are enabled for ipv6. Some Universities also: fundp.ac.be (Namur) and ulg.ac.be (Liege).

74.13. other websites

Other useful websites for testing ipv6 are:

test-ipv6.com
ipv6-test.com

Going to the ipv6-test.com website will test whether you have a valid accessible ipv6 address.



The image shows two screenshots from the ipv6-test.com website. The top screenshot displays the results of an IPv6 test: "Your internet connection is IPv6 capable", followed by the IPv6 address "2002:51a5:657d::1". Below this, it shows the Telenet logo with the Belgian flag and "BE", and states "Address type is 6to4" and "6to4 mapping to IPv4 address 81.165.101.125". The bottom screenshot displays the results of an IPv4 test: "Your internet connection is IPv4 capable", followed by the IPv4 address "81.165.101.125" and the URL "d51A5657D.access.telenet.be".

Going to the test-ipv6.com website will also test whether you have a valid accessible ipv6 address.

Test your IPv6 connectivity.

Summary | Tests Run | Technical Info | Share Results

Take Screenshot

- i** Your IPv4 address on the public Internet appears to be: `6to4`
- i** Your IPv6 address on the public Internet appears to be: `6to4`
- ✓** [World IPv6 day](#) is June 8th, 2011.
- ✓** Congratulations! You appear to have both IPv4 and IPv6 Internet working. If a publisher publishes to IPv6, your browser will connect using IPv6. Note: Your browser appears to prefer IPv4 over IPv6 when given the choice. This may in the future affect the accuracy of sites who guess at your location.
- i** You appear to be using a public 6to4 gateway; your router may be providing this to you automatically. Such public gateways have no service level agreements; you may see performance problems using such. Better would be to get a native IPv6 address from your ISP. [More info](#)
- i** Your DNS server (possibly run by your ISP) appears to have no access to the IPv6 Internet, or is not configured to use it. This may in the future restrict your ability to reach IPv6-only sites. [More info](#)

Your readiness scores

7/10 for your IPv4 stability and readiness, when publishers offer both IPv4 and IPv6

7/10 for your IPv6 stability and readiness, when publishers are forced to go IPv6 only

Click to see [test data](#)

74.14. 6to4 gateways

To access ipv4 only websites when on ipv6 you can use sixxs.net (more specifically <http://www.sixxs.net/tools/gateway/>) as a gateway.

For example use <http://www.slashdot.org.sixxs.org/> instead of <http://slashdot.org>

74.15. ping6 and dns

Below a screenshot of a **ping6** from behind a 6to4 connection.

81.165.101.125	195.130.131.4	DNS	Standard query AAAA ipv6-test.com
195.130.131.4	81.165.101.125	DNS	Standard query response AAAA 2001:41d0:2:67d1::7e57:1
2002:51a5:657d::1	2001:41d0:2:67d1::7e57:1	ICMPv6	Echo request
2001:41d0:2:67d1::7e57:1	2002:51a5:657d::1	ICMPv6	Echo reply
2002:51a5:657d::1	2001:41d0:2:67d1::7e57:1	ICMPv6	Echo request
2001:41d0:2:67d1::7e57:1	2002:51a5:657d::1	ICMPv6	Echo reply

74.16. ipv6 and tcp/http

Below a screenshot of a tcp handshake and http connection over ipv6.

Source	Destination	Protocol	Info
2002:51a5:657d::1	2001:41d0:2:67d1::7e57:1	TCP	38036 > http [SYN] Seq=0 Win=5648 L
2001:41d0:2:67d1::7e57:1	2002:51a5:657d::1	TCP	http > 38036 [SYN, ACK] Seq=0 Ack=1
2002:51a5:657d::1	2001:41d0:2:67d1::7e57:1	TCP	38036 > http [ACK] Seq=1 Ack=1 Win=
2002:51a5:657d::1	2001:41d0:2:67d1::7e57:1	HTTP	GET /json/addrinfo.php?PHPSESSID=19
2001:41d0:2:67d1::7e57:1	2002:51a5:657d::1	TCP	http > 38036 [ACK] Seq=1 Ack=708 Wi
2001:41d0:2:67d1::7e57:1	2002:51a5:657d::1	HTTP	HTTP/1.1 200 OK (text/javascript)

74.17. ipv6 PTR record

As seen in the DNS chapter, ipv6 PTR records are in the ip6.net domain, and have 32 generations of child domains.

```

Frame 46 (132 bytes on wire, 132 bytes captured)
  Ethernet II, Src: Apple_5d:2e:52 (00:26:bb:5d:2e:52), Dst: Riverdel_cf:6a:10 (00:30:b8:cf:6a:10)
  Internet Protocol, Src: 81.165.101.125 (81.165.101.125), Dst: 195.130.131.4 (195.130.131.4)
  User Datagram Protocol, Src Port: 34361 (34361), Dst Port: domain (53)
  Domain Name System (query)
    [Response In: 47]
    Transaction ID: 0xcfe3
    Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  Queries
    1.0.0.0.7.5.e.7.0.0.0.0.0.0.0.0.1.d.7.6.2.0.0.0.0.d.1.4.1.0.0.2.ip6.arpa: type PTR, class IN

```

74.18. 6to4 setup on Linux

Below a transcript of a 6to4 setup on Linux.

Thanks to <http://www.anyweb.co.nz/tutorial/v6Linux6to4> and <http://mirrors.bieringer.de/Linux+IPv6-HOWTO/> and <http://tldp.org/>!

```
root@mac:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:26:bb:5d:2e:52
          inet addr:81.165.101.125  Bcast:255.255.255.255  Mask:255.255.248.0
          inet6 addr: fe80::226:bbff:fe5d:2e52/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5926044 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2985892 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4274849823 (4.2 GB)  TX bytes:237002019 (237.0 MB)
          Interrupt:43 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:598 errors:0 dropped:0 overruns:0 frame:0
          TX packets:598 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:61737 (61.7 KB)  TX bytes:61737 (61.7 KB)

root@mac:~# sysctl -w net.ipv6.conf.default.forwarding=1
net.ipv6.conf.default.forwarding = 1
root@mac:~# ip tunnel add tun6to4 mode sit remote any local 81.165.101.125
root@mac:~# ip link set dev tun6to4 mtu 1472 up
root@mac:~# ip link show dev tun6to4
10: tun6to4: <NOARP,UP,LOWER_UP> mtu 1472 qdisc noqueue state UNKNOWN
    link/sit 81.165.101.125 brd 0.0.0.0
root@mac:~# ip -6 addr add dev tun6to4 2002:51a5:657d:0::1/64
root@mac:~# ip -6 addr add dev eth0 2002:51a5:657d:1::1/64
root@mac:~# ip -6 addr add dev eth0 fdcb:43c1:9c18:1::1/64
root@mac:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:26:bb:5d:2e:52
          inet addr:81.165.101.125  Bcast:255.255.255.255  Mask:255.255.248.0
          inet6 addr: fe80::226:bbff:fe5d:2e52/64 Scope:Link
          inet6 addr: fdcb:43c1:9c18:1::1/64 Scope:Global
          inet6 addr: 2002:51a5:657d:1::1/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5927436 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2986025 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4274948430 (4.2 GB)  TX bytes:237014619 (237.0 MB)
          Interrupt:43 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:598 errors:0 dropped:0 overruns:0 frame:0
          TX packets:598 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:61737 (61.7 KB)  TX bytes:61737 (61.7 KB)

tun6to4   Link encap:IPv6-in-IPv4
          inet6 addr: ::81.165.101.125/128 Scope:Compat
          inet6 addr: 2002:51a5:657d::1/64 Scope:Global
          UP RUNNING NOARP  MTU:1472  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
root@mac:~# ip -6 route add 2002::/16 dev tun6to4
root@mac:~# ip -6 route add ::/0 via ::192.88.99.1 dev tun6to4 metric 1
root@mac:~# ip -6 route show
::/96 via :: dev tun6to4 metric 256 mtu 1472 advmss 1412 hoplimit 0
2002:51a5:657d::/64 dev tun6to4 proto kernel metric 256 mtu 1472 advmss 1412 hoplimit 0
2002:51a5:657d:1::/64 dev eth0 proto kernel metric 256 mtu 1500 advmss 1440 hoplimit 0
2002::/16 dev tun6to4 metric 1024 mtu 1472 advmss 1412 hoplimit 0
fdcb:43c1:9c18:1::/64 dev eth0 proto kernel metric 256 mtu 1500 advmss 1440 hoplimit 0
fe80::/64 dev eth0 proto kernel metric 256 mtu 1500 advmss 1440 hoplimit 0
fe80::/64 dev tun6to4 proto kernel metric 256 mtu 1472 advmss 1412 hoplimit 0
default via ::192.88.99.1 dev tun6to4 metric 1 mtu 1472 advmss 1412 hoplimit 0
root@mac:~# ping6 ipv6-test.com
PING ipv6-test.com(ipv6-test.com) 56 data bytes
64 bytes from ipv6-test.com: icmp_seq=1 ttl=57 time=42.4 ms
64 bytes from ipv6-test.com: icmp_seq=2 ttl=57 time=43.0 ms
64 bytes from ipv6-test.com: icmp_seq=3 ttl=57 time=43.5 ms
64 bytes from ipv6-test.com: icmp_seq=4 ttl=57 time=43.9 ms
64 bytes from ipv6-test.com: icmp_seq=5 ttl=57 time=45.6 ms
^C
--- ipv6-test.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 42.485/43.717/45.632/1.091 ms
```

Part XXIV. mysql database

Chapter 75. introduction to sql using mysql

Table of Contents

75.1. installing mysql	683
75.2. accessing mysql	684
75.3. mysql databases	686
75.4. mysql tables	688
75.5. mysql records	690
75.6. joining two tables	694
75.7. mysql triggers	695

mysql is a database server that understands Structured Query Language (**SQL**). MySQL was developed by the Swedish Company **MySQL AB**. The first release was in 1995. In 2008 MySQL AB was bought by Sun Microsystems (which is now owned by Oracle).

mysql is very popular for websites in combination with **php** and **apache** (the **m** in **lamp** servers), but **mysql** is also used in organizations with huge databases like Facebook, Flickr, Google, Nokia, Wikipedia and Youtube.

This chapter will teach you **sql** by creating and using small databases, tables, queries and a simple trigger in a local **mysql** server.

75.1. installing mysql

On Debian/Ubuntu you can use **aptitude install mysql-server** to install the **mysql server** and **client**.

```
root@ubul204~# aptitude install mysql-server
The following NEW packages will be installed:
  libdbd-mysql-perl{a} libdbi-perl{a} libhtml-template-perl{a}
  libnet-daemon-perl{a} libplrpc-perl{a} mysql-client-5.5{a}
  mysql-client-core-5.5{a} mysql-server mysql-server-5.5{a}
  mysql-server-core-5.5{a}
0 packages upgraded, 10 newly installed, 0 to remove and 1 not upgraded.
Need to get 25.5 MB of archives. After unpacking 88.4 MB will be used.
Do you want to continue? [Y/n/?]
```

During the installation you will be asked to provide a password for the **root mysql user**, remember this password (or use **hunter2** like i do).

To verify the installed version, use **dpkg -l** on Debian/Ubuntu. This screenshot shows version 5.0 installed.

```
root@ubul204~# dpkg -l mysql-server | tail -1 | tr -s ' ' | cut -c-72
ii mysql-server 5.5.24-0ubuntu0.12.04.1 MySQL database server (metapacka
```

Issue **rpm -q** to get version information about MySQL on Red Hat/Fedora/CentOS.

```
[paul@RHEL52 ~]$ rpm -q mysql-server
mysql-server-5.0.45-7.el5
```

You will need at least version 5.0 to work with **triggers**.

75.2. accessing mysql

Linux users

The installation of **mysql** creates a user account in **/etc/passwd** and a group account in **/etc/group**.

```
kevin@ubul204:~$ tail -1 /etc/passwd
mysql:x:120:131:MySQL Server,,,:/nonexistent:/bin/false
kevin@ubul204:~$ tail -1 /etc/group
mysql:x:131:
```

The mysql daemon **mysqld** will run with the credentials of this user and group.

```
root@ubul204~# ps -eo uid,user,gid,group,comm | grep mysqld
  120 mysql          131 mysql      mysqld
```

mysql client application

You can now use mysql from the commandline by just typing **mysql -u root -p** and you'll be asked for the password (of the **mysql root** account). In the screenshot below the user typed **exit** to exit the mysql console.

```
root@ubul204~# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.24-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
```

You could also put the password in clear text on the command line, but that would not be very secure. Anyone with access to your bash history would be able to read your mysql root password.

```
root@ubul204~# mysql -u root -phunter2
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

~/.my.cnf

You can save configuration in your home directory in the hidden file **.my.cnf**. In the screenshot below we put the root user and password in **.my.cnf**.

```
kevin@ubul204:~$ pwd
/home/kevin
kevin@ubul204:~$ cat .my.cnf
[client]
user=root
password=hunter2
kevin@ubul204:~$
```

This enables us to log on as the **root mysql** user just by typing **mysql**.

```
kevin@ubul204:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 56
Server version: 5.5.24-0ubuntu0.12.04.1 (Ubuntu)
```

the mysql command line client

You can use the **mysql** command to take a look at the databases, and to execute SQL queries on them. The screenshots below show you how.

Here we execute the command **show databases**. Every command must be terminated by a delimiter. The default delimiter is **;** (the semicolon).

```
mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| mysql                   |
| performance_schema     |
| test                    |
+-----+
4 rows in set (0.00 sec)
```

We will use this prompt in the next sections.

75.3. mysql databases

listing all databases

You can use the **mysql** command to take a look at the databases, and to execute SQL queries on them. The screenshots below show you how. First, we log on to our MySQL server and execute the command **show databases** to see which databases exist on our mysql server.

```
kevin@ubu1204:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 57
Server version: 5.5.24-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| mysql             |
| performance_schema |
| test              |
+-----+
4 rows in set (0.00 sec)
```

creating a database

You can create a new database with the **create database** command.

```
mysql> create database famouspeople;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| famouspeople      |
| mysql             |
| performance_schema |
| test              |
+-----+
5 rows in set (0.00 sec)
```

using a database

Next we tell **mysql** to use one particular database with the **use \$database** command. This screenshot shows how to make wikidb the current database (in use).

```
mysql> use famouspeople;  
Database changed  
mysql>
```

access to a database

To give someone access to a mysql database, use the **grant** command.

```
mysql> grant all on famouspeople.* to kevin@localhost IDENTIFIED BY "hunter2";  
Query OK, 0 rows affected (0.00 sec)
```

deleting a database

When a database is no longer needed, you can permanently remove it with the **drop database** command.

```
mysql> drop database demodb;  
Query OK, 1 row affected (0.09 sec)
```

backup and restore a database

You can take a backup of a database, or move it to another computer using the **mysql** and **mysqldump** commands. In the screenshot below, we take a backup of the wikidb database on the computer named laika.

```
mysqldump -u root famouspeople > famouspeople.backup.20120708.sql
```

Here is a screenshot of a database restore operation from this backup.

```
mysql -u root famouspeople < famouspeople.backup.20120708.sql
```

75.4. mysql tables

listing tables

You can see a list of tables in the current database with the **show tables;** command. Our **famouspeople** database has no tables yet.

```
mysql> use famouspeople;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

creating a table

The **create table** command will create a new table.

This screenshot shows the creation of a country table. We use the **countrycode** as a **primary key** (all country codes are uniquely defined). Most country codes are two or three letters, so a **char** of three uses less space than a **varchar** of three. The **country name** and the name of the capital are both defined as **varchar**. The population can be seen as an **integer**.

```
mysql> create table country (
  -> countrycode char(3) NOT NULL,
  -> countryname varchar(70) NOT NULL,
  -> population int,
  -> countrycapital varchar(50),
  -> primary key (countrycode)
  -> );
Query OK, 0 rows affected (0.19 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_famouspeople |
+-----+
| country                  |
+-----+
1 row in set (0.00 sec)
```

```
mysql>
```

You are allowed to type the **create table** command on one long line, but administrators often use multiple lines to improve readability.

```
mysql> create table country ( countrycode char(3) NOT NULL, countryname\
  varchar(70) NOT NULL, population int, countrycapital varchar(50), prim\
  ary key (countrycode) );
Query OK, 0 rows affected (0.18 sec)
```

describing a table

To see a description of the structure of a table, issue the **describe \$tablename** command as shown below.

```
mysql> describe country;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| countrycode   | char(3)       | NO   | PRI | NULL    |       |
| countryname   | varchar(70)   | NO   |     | NULL    |       |
| population    | int(11)       | YES  |     | NULL    |       |
| countrycapital| varchar(50)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

removing a table

To remove a table from a database, issue the **drop table \$tablename** command as shown below.

```
mysql> drop table country;
Query OK, 0 rows affected (0.00 sec)
```

75.5. mysql records

creating records

Use **insert** to enter data into the table. The screenshot shows several insert statements that insert values depending on the position of the data in the statement.

```
mysql> insert into country values ('BE','Belgium','11000000','Brussels');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> insert into country values ('DE','Germany','82000000','Berlin');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> insert into country values ('JP','Japan','128000000','Tokyo');
Query OK, 1 row affected (0.05 sec)
```

Some administrators prefer to use uppercase for **sql** keywords. The mysql client accepts both.

```
mysql> INSERT INTO country VALUES ('FR','France','64000000','Paris');
Query OK, 1 row affected (0.00 sec)
```

Note that you get an error when using a duplicate **primary key**.

```
mysql> insert into country values ('DE','Germany','82000000','Berlin');
ERROR 1062 (23000): Duplicate entry 'DE' for key 'PRIMARY'
```

viewing all records

Below an example of a simple **select** query to look at the contents of a table.

```
mysql> select * from country;
+-----+-----+-----+-----+
| countrycode | countryname | population | countrycapital |
+-----+-----+-----+-----+
| BE          | Belgium     | 11000000  | Brussels       |
| CN          | China       | 1400000000| Beijing        |
| DE          | Germany     | 82000000  | Berlin         |
| FR          | France      | 64000000  | Paris          |
| IN          | India       | 1300000000| New Delhi      |
| JP          | Japan       | 128000000 | Tokyo          |
| MX          | Mexico      | 113000000 | Mexico City    |
| US          | United States | 313000000 | Washington     |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```


updating records

Consider the following **insert** statement. The capital of Spain is not Barcelona, it is Madrid.

```
mysql> insert into country values ('ES','Spain','48000000','Barcelona');
Query OK, 1 row affected (0.08 sec)
```

Using an **update** statement, the record can be updated.

```
mysql> update country set countrycapital='Madrid' where countrycode='ES';
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

We can use a **select** statement to verify this change.

```
mysql> select * from country;
+-----+-----+-----+-----+
| countrycode | countryname | population | countrycapital |
+-----+-----+-----+-----+
| BE          | Belgium     | 11000000  | Brussels       |
| CN          | China       | 1400000000| Beijing        |
| DE          | Germany     | 82000000  | Berlin         |
| ES          | Spain       | 48000000  | Madrid         |
| FR          | France      | 64000000  | Paris          |
| IN          | India       | 1300000000| New Delhi      |
| JP          | Japan       | 128000000 | Tokyo          |
| MX          | Mexico      | 113000000 | Mexico City    |
| US          | United States | 313000000 | Washington     |
+-----+-----+-----+-----+
9 rows in set (0.00 sec)
```

viewing selected records

Using a **where** clause in a **select** statement, you can specify which record(s) you want to see.

```
mysql> SELECT * FROM country WHERE countrycode='ES';
+-----+-----+-----+-----+
| countrycode | countryname | population | countrycapital |
+-----+-----+-----+-----+
| ES          | Spain       | 48000000  | Madrid         |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Another example of the **where** clause.

```
mysql> select * from country where countryname='Spain';
+-----+-----+-----+-----+
| countrycode | countryname | population | countrycapital |
+-----+-----+-----+-----+
| ES          | Spain       | 48000000  | Madrid         |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

primary key in where clause ?

The **primary key** of a table is a field that uniquely identifies every record (every row) in the table. when using another field in the **where** clause, it is possible to get multiple rows returned.

```
mysql> insert into country values ('EG','Egypt','82000000','Cairo');
Query OK, 1 row affected (0.33 sec)
```

```
mysql> select * from country where population='82000000';
+-----+-----+-----+-----+
| countrycode | countryname | population | countrycapital |
+-----+-----+-----+-----+
| DE          | Germany    | 82000000  | Berlin         |
| EG          | Egypt      | 82000000  | Cairo          |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

ordering records

We know that **select** allows us to see all records in a table. Consider this table.

```
mysql> select countryname,population from country;
+-----+-----+
| countryname | population |
+-----+-----+
| Belgium     | 11000000  |
| China       | 1400000000|
| Germany     | 82000000  |
| Egypt       | 82000000  |
| Spain       | 48000000  |
| France      | 64000000  |
| India       | 1300000000|
| Japan       | 128000000 |
| Mexico      | 113000000 |
| United States | 313000000 |
+-----+-----+
10 rows in set (0.00 sec)
```

Using the **order by** clause, we can change the order in which the records are presented.

```
mysql> select countryname,population from country order by countryname;
+-----+-----+
| countryname | population |
+-----+-----+
| Belgium     | 11000000  |
| China       | 1400000000|
| Egypt       | 82000000  |
| France      | 64000000  |
| Germany     | 82000000  |
| India       | 1300000000|
| Japan       | 128000000 |
| Mexico      | 113000000 |
| Spain       | 48000000  |
| United States | 313000000 |
+-----+-----+
10 rows in set (0.00 sec)
```

grouping records

Consider this table of people. The screenshot shows how to use the **avg** function to calculate an average.

```
mysql> select * from people;
+-----+-----+-----+-----+
| Name          | Field    | birthyear | countrycode |
+-----+-----+-----+-----+
| Barack Obama  | politics | 1961      | US          |
| Deng Xiaoping | politics | 1904      | CN          |
| Guy Verhofstadt | politics | 1953      | BE          |
| Justine Henin | tennis  | 1982      | BE          |
| Kim Clijsters | tennis  | 1983      | BE          |
| Li Na         | tennis  | 1982      | CN          |
| Liu Yang      | astronaut | 1978      | CN          |
| Serena Williams | tennis  | 1981      | US          |
| Venus Williams | tennis  | 1980      | US          |
+-----+-----+-----+-----+
9 rows in set (0.00 sec)

mysql> select Field,AVG(birthyear) from people;
+-----+-----+
| Field    | AVG(birthyear) |
+-----+-----+
| politics | 1967.1111111111 |
+-----+-----+
1 row in set (0.00 sec)
```

Using the **group by** clause, we can have an average per field.

```
mysql> select Field,AVG(birthyear) from people group by Field;
+-----+-----+
| Field    | AVG(birthyear) |
+-----+-----+
| astronaut | 1978           |
| politics  | 1939.3333333333 |
| tennis    | 1981.6         |
+-----+-----+
3 rows in set (0.00 sec)
```

deleting records

You can use the **delete** to permanently remove a record from a table.

```
mysql> delete from country where countryname='Spain';
Query OK, 1 row affected (0.06 sec)

mysql> select * from country where countryname='Spain';
Empty set (0.00 sec)
```

75.6. joining two tables

inner join

With an **inner join** you can take values from two tables and combine them in one result. Consider the country and the people tables from the previous section when looking at this screenshot of an **inner join**.

```
mysql> select Name,Field,countryname
-> from country
-> inner join people on people.countrycode=country.countrycode;
```

Name	Field	countryname
Barack Obama	politics	United States
Deng Xiaoping	politics	China
Guy Verhofstadt	politics	Belgium
Justine Henin	tennis	Belgium
Kim Clijsters	tennis	Belgium
Li Na	tennis	China
Liu Yang	astronaut	China
Serena Williams	tennis	United States
Venus Williams	tennis	United States

9 rows in set (0.00 sec)

This **inner join** will show only records with a match on **countrycode** in both tables.

left join

A **left join** is different from an **inner join** in that it will take all rows from the left table, regardless of a match in the right table.

```
mysql> select Name,Field,countryname from country left join people on people.countrycode
```

Name	Field	countryname
Guy Verhofstadt	politics	Belgium
Justine Henin	tennis	Belgium
Kim Clijsters	tennis	Belgium
Deng Xiaoping	politics	China
Li Na	tennis	China
Liu Yang	astronaut	China
NULL	NULL	Germany
NULL	NULL	Egypt
NULL	NULL	Spain
NULL	NULL	France
NULL	NULL	India
NULL	NULL	Japan
NULL	NULL	Mexico
Barack Obama	politics	United States
Serena Williams	tennis	United States
Venus Williams	tennis	United States

16 rows in set (0.00 sec)

You can see that some countries are present, even when they have no matching records in the **people** table.

75.7. mysql triggers

using a before trigger

Consider the following **create table** command. The last field (**amount**) is the multiplication of the two fields named **unitprice** and **unitcount**.

```
mysql> create table invoices (  
-> id char(8) NOT NULL,  
-> customerid char(3) NOT NULL,  
-> unitprice int,  
-> unitcount smallint,  
-> amount int );  
Query OK, 0 rows affected (0.00 sec)
```

We can let mysql do the calculation for that by using a **before trigger**. The screenshot below shows the creation of a trigger that calculates the amount by multiplying two fields that are about to be inserted.

```
mysql> create trigger total_amount before INSERT on invoices  
-> for each row set new.amount = new.unitprice * new.unitcount ;  
Query OK, 0 rows affected (0.02 sec)
```

Here we verify that the trigger works by inserting a new record, without providing the total amount.

```
mysql> insert into invoices values ('20090526','ABC','199','10','');  
Query OK, 1 row affected (0.02 sec)
```

Looking at the record proves that the trigger works.

```
mysql> select * from invoices;  
+-----+-----+-----+-----+-----+  
| id      | customerid | unitprice | unitcount | amount |  
+-----+-----+-----+-----+-----+  
| 20090526 | ABC      | 199      | 10      | 1990  |  
+-----+-----+-----+-----+-----+  
1 row in set (0.00 sec)
```

removing a trigger

When a **trigger** is no longer needed, you can delete it with the **drop trigger** command.

```
mysql> drop trigger total_amount;  
Query OK, 0 rows affected (0.00 sec)
```

Part XXV. selinux

Chapter 76. introduction to SELinux(draft)

Table of Contents

76.1. about selinux	697
76.2. selinux modes	698
76.3. activating selinux	698
76.4. getenforce	698
76.5. setenforce	698
76.6. sestatus	699
76.7. logging	699
76.8. DAC or MAC	700
76.9. ls -Z	700
76.10. /selinux	700
76.11. /etc/selinux/config	701
76.12. identity	701
76.13. type (or domain)	701
76.14. role	702
76.15. security context	702
76.16. transition	702
76.17. policy	703
76.18. extended attributes	703
76.19. process security context	703
76.20. chcon	703
76.21. a practical example	704

76.1. about selinux

Security Enhanced Linux or **SELinux** is a set of modifications developed by the United States National Security Agency (NSA) to provide a variety of security policies for Linux. SELinux was released as open source at the end of 2000. Since kernel version 2.6 it is an integrated part of Linux.

SELinux offers security! SELinux can control what kind of access users have to files and processes. Even when a file received **chmod 777**, SELinux can still prevent users from accessing it (unix file permissions are checked first!). SELinux does this by placing users in **roles** that represent a security context. Administrators have very strict control on access permissions granted to roles.

SELinux is present in the latest versions of Red Hat Enterprise Linux, Debian, Fedora, Ubuntu, Yellow Dog Linux and Hardened Gentoo. There is currently (2008) limited support in Suse and Slackware.

76.2. selinux modes

selinux knows three modes: enforcing, permissive and disabled. The **enforcing** mode will enforce policies, and may deny access based on **selinux rules**. The **permissive** mode will not enforce policies, but can still log actions that would have been denied in **enforcing** mode. The **disabled** mode disables **selinux**.

76.3. activating selinux

On RHEL you can use the GUI tool to activate **selinux**, on Debian there is the **selinux-activate** command. Activation requires a reboot.

```
root@deb503:~# selinux-activate
Activating SE Linux
Searching for GRUB installation directory ... found: /boot/grub
Searching for default file ... found: /boot/grub/default
Testing for an existing GRUB menu.lst file ... found: /boot/grub/menu.lst
Searching for splash image ... none found, skipping ...
Found kernel: /boot/vmlinuz-2.6.26-2-686
Updating /boot/grub/menu.lst ... done

SE Linux is activated. You may need to reboot now.
```

76.4. getenforce

Use **getenforce** to verify whether selinux is **enforced**, **disabled** or **permissive**.

```
[root@rhel55 ~]# getenforce
Permissive
```

The **/selinux/enforce** file contains 1 when enforcing, and 0 when permissive mode is active.

```
root@fedora13 ~# cat /selinux/enforce
1root@fedora13 ~#
```

76.5. setenforce

You can use **setenforce** to switch between the **Permissive** or the **Enforcing** state once **selinux** is activated..

```
[root@rhel55 ~]# setenforce Enforcing
[root@rhel55 ~]# getenforce
Enforcing
[root@rhel55 ~]# setenforce Permissive
```



```
[root@rhel155 ~]# getenforce
Permissive
```

76.6. sestatus

You can see the current **selinux** status and policy with the **sestatus** command.

```
[root@rhel155 ~]# sestatus
SELinux status:                enabled
SELinuxfs mount:              /selinux
Current mode:                  permissive
Mode from config file:        permissive
Policy version:                21
Policy from config file:      targeted
```

76.7. logging

Verify that **syslog** is running and activated on boot to enable logging of deny messages in **/var/log/messages**.

```
[root@rhel155 ~]# chkconfig --list syslog
syslog          0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

Verify that **auditd** is running and activated on boot to enable logging of easier to read messages in **/var/log/audit/audit.log**.

```
[root@rhel155 ~]# chkconfig --list auditd
auditd         0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

If not activated, then run **chkconfig --levels 2345 auditd on** and **service auditd start**.

```
[root@rhel155 ~]# service auditd status
auditd (pid 1660) is running...
[root@rhel155 ~]# service syslog status
syslogd (pid 1688) is running...
klogd (pid 1691) is running...
```

The **/var/log/messages** log file will tell you that **selinux** is disabled.

```
root@deb503:~# grep -i selinux /var/log/messages
Jun 25 15:59:34 deb503 kernel: [    0.084083] SELinux:  Disabled at boot.
```

Or that it is enabled.

```
root@deb503:~# grep SELinux /var/log/messages | grep -i Init
```

```
Jun 25 15:09:52 deb503 kernel: [ 0.084094] SELinux: Initializing.
```

76.8. DAC or MAC

Standard Unix permissions use **Discretionary Access Control** to set permissions on files. This means that a user that owns a file, can make it world readable by typing **chmod 777 \$file**.

With **selinux** the kernel will enforce **Mandatory Access Control** which strictly controls what processes or threads can do with files (superseding DAC). Processes are confined by the kernel to the minimum access they require.

76.9. ls -Z

To see the DAC permissions on a file, use **ls -l** to display user and group **owner** and permissions (here **rw-r--r--**).

```
root@deb503:~/selinux# touch test42.txt
root@deb503:~/selinux# ls -l
total 0
-rw-r--r-- 1 root root 0 2010-06-25 15:38 test42.txt
```

For MAC permissions there is new **-Z** option added to **ls**. The output shows an **selinux** user named **unconfined_u**, a role named **object_r**, a type named **unconfined_home_t**, and a level **S0**.

```
root@deb503:~/selinux# ls -Z
unconfined_u:object_r:unconfined_home_t:s0 test42.txt
```

76.10. /selinux

When selinux is active, there is a new virtual file system named **/selinux**. (You can compare it to **/proc** and **/dev**.)

```
[root@RHEL5 ~]# ls /selinux/
access          context  mls
avc             create   null
booleans       disable  policyvers
checkreqprot   enforce  relabel
commit_pending_bools  load    user
compat_net     member
```

Although some files in **/selinux** appear with size 0, they often contain a boolean value. Check **/selinux/enforce** to see if selinux is running in enforced mode.

```
[root@RHEL5 ~]# ls -l /selinux/enforce
-rw-r--r-- 1 root root 0 Apr 29 08:21 /selinux/enforce
[root@RHEL5 ~]# echo `cat /selinux/enforce`
1
```

76.11. /etc/selinux/config

The main configuration file for **selinux** is **/etc/selinux/config**. When in **permissive** mode, the file looks like this.

```
[root@rhel55 ~]# more /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - SELinux is fully disabled.
SELINUX=permissive
# SELINUXTYPE= type of policy in use. Possible values are:
#     targeted - Only targeted network daemons are protected.
#     strict - Full SELinux protection.
SELINUXTYPE=targeted
```

76.12. identity

The **SELinux Identity** of a user is distinct from the user ID. An identity is part of a security context, and (via domains) determines what you can do. The screenshot shows user **root** having identity **user_u**.

```
[root@rhel55 ~]# id -Z
user_u:system_r:unconfined_t
```

76.13. type (or domain)

The **selinux domain** is the security context of a process. An **selinux domain** determines what a process can do. The screenshot shows **init** running in domain **init_t** and the **mingetty**'s running in domain **getty_t**.

```
[root@RHEL5 ~]# ps fax -Z | grep init_t
system_u:system_r:init_t      1 ?        Ss      0:01 init [3]
[root@RHEL5 ~]# ps fax -Z | grep getty_t
system_u:system_r:getty_t    2941 tty1   Ss+    0:00 /sbin/mingetty tty1
system_u:system_r:getty_t    2942 tty2   Ss+    0:00 /sbin/mingetty tty2
```

The **selinux type** is similar to an **selinux domain**, but refers to directories and files instead of processes.

76.14. role

The **selinux role** defines the domains that can be used. A **role** is denied to enter a domain, unless the **role** is explicitly authorized to do so.

76.15. security context

The combination of identity, role and domain or type make up the **selinux security context**. The **id** will show you your security context in the form identity:role:domain.

```
[paul@RHEL5 ~]$ id | cut -d' ' -f4
context=user_u:system_r:unconfined_t
```

The **ls -Z** command shows the security context for a file in the form identity:role:type.

```
[paul@RHEL5 ~]$ ls -Z test
-rw-rw-r-- paul paul user_u:object_r:user_home_t test
```

The security context for processes visible in `/proc` defines both the type (of the file in `/proc`) and the domain (of the running process). Let's take a look at the `init` process and `/proc/1/`.

The `init` process runs in domain **init_t**.

```
[root@RHEL5 ~]# ps -ZC init
LABEL                                PID TTY          TIME CMD
system_u:system_r:init_t             1 ?              00:00:01 init
```

The `/proc/1/` directory, which identifies the **init** process, has type **init_t**.

```
[root@RHEL5 ~]# ls -Zd /proc/1/
dr-xr-xr-x root root system_u:system_r:init_t /proc/1/
```

It is not a coincidence that the domain of the **init** process and the type of `/proc/1/` are both **init_t**.

Don't try to use **chcon** on `/proc`! It will not work.

76.16. transition

An **selinux transition** (aka an selinux labelling) determines the security context that will be assigned. A transition of process domains is used when you execute a process. A transition of file type happens when you create a file.

An example of file type transition.

```
[paul@RHEL5 ~]$ touch test
[paul@RHEL5 ~]$ touch /tmp/test
[paul@RHEL5 ~]$ ls -Z test
-rw-rw-r-- paul paul user_u:object_r:user_home_t test
[paul@RHEL5 ~]$ ls -Z /tmp/test
-rw-rw-r-- paul paul user_u:object_r:tmp_t /tmp/test
[paul@RHEL5 ~]$
```

76.17. policy

Everything comes together in an **selinux policy**. Policies define user access to roles, role access to domains and domain access to types.

76.18. extended attributes

Extended attributes are use by **selinux** to store security contexts. These attributes can be viewed with **ls** when **selinux** is running.

```
[root@RHEL5 home]# ls --context
drwx----- paul paul system_u:object_r:user_home_dir_t paul
drwxr-xr-x root root user_u:object_r:user_home_dir_t project42
drwxr-xr-x root root user_u:object_r:user_home_dir_t project55
[root@RHEL5 home]# ls -Z
drwx----- paul paul system_u:object_r:user_home_dir_t paul
drwxr-xr-x root root user_u:object_r:user_home_dir_t project42
drwxr-xr-x root root user_u:object_r:user_home_dir_t project55
[root@RHEL5 home]#
```

When selinux is not running, then **getfattr** is the tool to use.

```
[root@RHEL5 etc]# getfattr -m . -d hosts
# file: hosts
security.selinux="system_u:object_r:etc_t:s0\000"
```

76.19. process security context

A new option is added to **ps** to see the selinux security context of processes.

```
[root@RHEL5 etc]# ps -ZC mingetty
LABEL                                PID TTY          TIME CMD
system_u:system_r:getty_t            2941 tty1          00:00:00 mingetty
system_u:system_r:getty_t            2942 tty2          00:00:00 mingetty
```

76.20. chcon

Use **chcon** to change the selinux security context.

This example shows how to use **chcon** to change the **type** of a file.

```
[root@rhel55 ~]# ls -Z /var/www/html/test42.txt
-rw-r--r-- root root user_u:object_r:httpd_sys_content_t /var/www/html/test42.txt
[root@rhel55 ~]# chcon -t samba_share_t /var/www/html/test42.txt
[root@rhel55 ~]# ls -Z /var/www/html/test42.txt
-rw-r--r-- root root user_u:object_r:samba_share_t /var/www/html/test42.txt
```

76.21. a practical example

The **apache webserver** is by default targeted with **selinux**. The next screenshot shows that any file created in **/var/www/html** will by default get the **http_sys_content_t** type.

```
[root@rhel55 ~]# touch /var/www/html/test42.txt
[root@rhel55 ~]# ls -Z /var/www/html/test42.txt
-rw-r--r-- root root user_u:object_r:httpd_sys_content_t /var/www/html/test42.txt
```

Files created elsewhere do not get this type.

```
[root@rhel55 ~]# touch /root/test42.txt
[root@rhel55 ~]# ls -Z /root/test42.txt
-rw-r--r-- root root user_u:object_r:user_home_t /root/test42.txt
```

Make sure **apache** runs.

```
[root@rhel55 ~]# service httpd start
[ OK ]
```

Will this work ? Yes it does.

```
[root@rhel55 ~]# wget http://localhost/test42.txt
--2010-06-26 15:40:28-- http://localhost/test42.txt
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
....
```

Why does this work ? Because apache runs in the **httpd_t** domain.

```
[root@rhel55 ~]# ps -ZC httpd
LABEL                                PID TTY          TIME CMD
user_u:system_r:httpd_t              2979 ?            00:00:07 httpd
user_u:system_r:httpd_t              2981 ?            00:00:00 httpd
user_u:system_r:httpd_t              2982 ?            00:00:00 httpd
user_u:system_r:httpd_t              2983 ?            00:00:00 httpd
user_u:system_r:httpd_t              2984 ?            00:00:00 httpd
user_u:system_r:httpd_t              2985 ?            00:00:00 httpd
user_u:system_r:httpd_t              2986 ?            00:00:00 httpd
```

```
user_u:system_r:httpd_t          2987 ?          00:00:00 httpd
user_u:system_r:httpd_t          2988 ?          00:00:00 httpd
```

So let's try to change the **selinux type** of this file.

```
[root@rhel155 ~]# chcon -t samba_share_t /var/www/html/test42.txt
[root@rhel155 ~]# ls -Z /var/www/html/test42.txt
-rw-r--r-- root root user_u:object_r:samba_share_t /var/www/html/test42.txt
```

There are two possibilities now: either it works, or it fails. It works when **selinux** is in **permissive mode**, it fails when in **enforcing mode**.

```
[root@rhel155 ~]# wget http://localhost/test42.txt
--2010-06-26 15:41:33-- http://localhost/test42.txt
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 200 OK
...
```

The log file clearly shows that it would have failed in **enforcing mode**.

```
[root@rhel155 ~]# grep test42 /var/log/audit/audit.log
type=AVC msg=audit(1277559693.656:105): avc: denied { getattr } for \
pid=2982 comm="httpd" path="/var/www/html/test42.txt" dev=dm-0 ino=1974\
99 scontext=user_u:system_r:httpd_t:s0 tcontext=user_u:object_r:samba_s\
hare_t:s0 tclass=file
type=AVC msg=audit(1277559693.658:106): avc: denied { read } for pid\
=2982 comm="httpd" name="test42.txt" dev=dm-0 ino=197499 scontext=user_\
u:system_r:httpd_t:s0 tcontext=user_u:object_r:samba_share_t:s0 tclass=\
file
```

Part XXVI. 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 (CMDDBA). The **CMDEV** is focused towards database application developers, and the **CMDDBA** towards database administrators. Both tracks require two exams each. The MySQL cluster DBA certification requires CMDDBA certification and passing the CMCDDBA 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 **Plug and Play** buses.

Older **x86** computers often had **isa** buses, which can be configured using **jumper**s 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/  
input pci usb
```

Can you guess which of these two screenshots was taken on a laptop ?

/usr/sbin/lusb

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 **lusb**, which is executed here on a SPARC system with Ubuntu.

```
root@shaka:~# lusb  
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.

```
paul@barry:~$ zmore /var/lib/usbutils/usb.ids | head
-----> /var/lib/usbutils/usb.ids <-----
#
# List of USB ID's
#
# Maintained by Vojtech Pavlik <vojtech@suse.cz>
# If you have any new entries, send them to the maintainer.
# The latest version can be obtained from
# http://www.linux-usb.org/usb.ids
#
# $Id: usb.ids,v 1.225 2006/07/13 04:18:02 dbrownell Exp $
```

/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
```

	CPU0	CPU1		
0:	1320048	555	IO-APIC-edge	timer
1:	10224	7	IO-APIC-edge	i8042
7:	0	0	IO-APIC-edge	parport0
8:	2	1	IO-APIC-edge	rtc
10:	3062	21	IO-APIC-fasteoi	acpi
12:	131	2	IO-APIC-edge	i8042
15:	47073	0	IO-APIC-edge	ide1
18:	0	1	IO-APIC-fasteoi	yenta
19:	31056	1	IO-APIC-fasteoi	libata, ohci1394
20:	19042	1	IO-APIC-fasteoi	eth0
21:	44052	1	IO-APIC-fasteoi	uhci_hcd:usb1, uhci_hcd:usb2,...
22:	188352	1	IO-APIC-fasteoi	ra0
23:	632444	1	IO-APIC-fasteoi	nvidia
24:	1585	1	IO-APIC-fasteoi	VIA82XX-MODEM, VIA8237

dmesg

You can also use **dmesg** to find irq's allocated at boot time.

```
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**.

```
[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
00a0-00a1 : 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'
[ 20.576000] parport: PnPBIOS parport detected.
[ 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
```

Appendix D. installing linux

D.1. about

The past couple of years the installation of linux has become a lot easier then before, at least for end users installing a distro like Ubuntu, Fedora, Debian or Mandrake on their home computer. Servers usually come pre-installed, and if not pre-installed, then setup of a linux server today is very easy.

Linux can be installed in many different ways. End users most commonly use cdrom's or dvd's for installation, most of the time with a working internet connection te receive updates. Administrators might prefer network installations using protocols like **tftp**, **bootp**, **rarp** and/or **nfs** or response file solutions like **Red Hat Kickstart** or **Solaris Jumpstart**.

D.2. installation by cdrom

Installation of linux from cdrom is easy! Most distributions ask very few questions during install (keyboard type, language, username) and detect all the hardware themselves. There is usually no need to retrieve third-party drivers from the internet. The GUI installation gives options like Desktop (for end users), Workstation (for developers), Server or minimal (usually without graphical interface).

D.3. installation with rarp and tftp

Installing over the network involves powering on the machine, have it find a rarpd server to get an ip-address, then let it find an tftps server to get an installation image copied to the machine. This image can then boot. The procedure below demonstrates how to setup three Sun SPARC servers with Ubuntu Linux, using a Debian Linux machine to host the tftp, bootp and nfs daemons.

First we need to configure the mac to ip resolution in the **/etc/ethers** configuration file. Each server will receive a unique ip-address during installation.

```
root@laika:~# cat /etc/ethers
00:03:ba:02:c3:82      192.168.1.71
00:03:ba:09:7c:f9      192.168.1.72
00:03:ba:09:7f:d2      192.168.1.73
```

We need to install the rarpd and tftpd daemons on the (Debian) machine that will be hosting the install image.

```
root@laika:~# aptitude install rarpd
root@laika:~# aptitude install tftpd
```

The tftp services must be activated in inetd or xinetd.

```
root@laika:~# cat /etc/inetd.conf | tail -1
tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /srv/tftp
```

And finally the linux install image must be present in the tftp served directory. The filename of the image must be the hex ip-address, this is accomplished with symbolic links.

```
root@laika:~# ll /srv/tftp/
total 7.5M
lrwxrwxrwx 1 root root 13 2007-03-02 21:49 C0A80147 -> ubuntu610.img
lrwxrwxrwx 1 root root 13 2007-03-03 14:13 C0A80148 -> ubuntu610.img
lrwxrwxrwx 1 root root 13 2007-03-02 21:49 C0A80149 -> ubuntu610.img
-rw-r--r-- 1 paul paul 7.5M 2007-03-02 21:42 ubuntu610.img
```

Time to enter **boot net** now in the openboot prompt. Twenty minutes later the three servers where humming with linux.

D.4. about Red Hat kickstart

Automating Linux installations with response files can be done with **Red Hat kickstart**. One way to set it up is by using the graphical tool `/usr/sbin/system-config-kickstart`. If you prefer to set it up manually, read on.

You can modify the sample kickstart file **RH-DOCS/sample.ks** (can be found on the documentation dvd). Put this file so **anaconda** can read it.

*Anaconda is the Red Hat installer written in **python**. The name is chose because anacondas are lizard-eating pythons. Lizard is the name of the Caldera Linux installation program.*

Another option is to start with the `/root/anaconda-ks.cfg` file. This is a sample kickstart file that contains all the settings from your current installation.

Do not change the order of the sections inside your kickstart file! The Red Hat System Administration Guide contains about 25 pages describing all the options, most of them are easy ti understand if you already performed a couple of installations.

D.5. using kickstart

To use kickstart, name your kickstart file **ks.cfg** and put it in the root directory of your installation cdrom (or on a usb stick or a floppy). For network based installations, name the file **\$ip-address-kickstart** and place the following in **dhcpd.conf**.

```
filename "/export/kickstart"
next-server remote.installation.server
```

Leaving out the **next-server** line will result in the client looking for the file on the dhcp server itself.

Booting from cdrom with kickstart requires the following command at the **boot:** prompt.

```
linux ks=cdrom:/ks.cfg
```

When the kickstart file is on the network, use nfs or http like in these examples.

```
linux ks=nfs:servername:/path/to/ks.cfg
```

```
linux ks=http://servername/path/to/ks.cfg
```

Appendix E. disk quotas

E.1. About Disk Quotas

To limit the disk space used by user, you can set up **disk quotas**. This requires adding **usrquota** and/or **grpquota** to one or more of the file systems in **/etc/fstab**.

```
root@RHELv4u4:~# cat /etc/fstab | grep usrquota
/dev/VolGroup00/LogVol102    /home    ext3    usrquota,grpquota    0 0
```

Next you need to remount the file system.

```
root@RHELv4u4:~# mount -o remount /home
```

The next step is to build the **quota.user** and/or **quota.group** files. These files (called the **quota files**) contain the table of the disk usage on that file system. Use the **quotacheck** command to accomplish this.

```
root@RHELv4u4:~# quotacheck -cug /home
root@RHELv4u4:~# quotacheck -avug
```

The **-c** is for create, **u** for user quota, **g** for group, **a** for checking all quota enabled file systems in **/etc/fstab** and **v** for verbose information. The next step is to edit individual user quotas with **edquota** or set a general quota on the file system with **edquota -t**. The tool will enable you to put **hard** (this is the real limit) and **soft** (allows a grace period) limits on **blocks** and **inodes**. The **quota** command will verify that quota for a user is set. You can have a nice overview with **repquota**.

The final step (before your users start complaining about lack of disk space) is to enable quotas with **quotaon(1)**.

```
root@RHELv4u4:~# quotaon -vaug
```

Issue the **quotaoff** command to stop all complaints.

```
root@RHELv4u4:~# quotaoff -vaug
```

E.2. Practice Disk quotas

1. Implement disk quotas on one of your new partitions. Limit one of your users to 10 megabyte.
2. Test that they work by copying many files to the quota'd partition.

Appendix F. introduction to vnc

F.1. About VNC

VNC can be configured in gnome or KDE using the **Remote Desktop Preferences**. VNC can be used to run your desktop on another computer, and you can also use it to see and take over the Desktop of another user. The last part can be useful for help desks to show users how to do things. VNC has the added advantage of being operating system independent, a lot of products (realvnc, tightvnc, xvnc, ...) use the same protocol on Solaris, Linux, BSD and more.

F.2. VNC Server

Starting the vnc server for the first time.

```
[root@RHELv4u3 conf]# rpm -qa | grep -i vnc
vnc-server-4.0-8.1
vnc-4.0-8.1
[root@RHELv4u3 conf]# vncserver :2
```

You will require a password to access your desktops.

```
Password:
Verify:
xauth: creating new authority file /root/.Xauthority
```

```
New 'RHELv4u3.localdomain:2 (root)' desktop is RHELv4u3.localdomain:2
```

```
Creating default startup script /root/.vnc/xstartup
Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/RHELv4u3.localdomain:2.log
```

```
[root@RHELv4u3 conf]#
```

F.3. VNC Client

You can now use the **vncviewer** from another machine to connect to your vnc server. It will default to a very simple graphical interface...

```
paul@laika:~$ vncviewer 192.168.1.49:2
VNC viewer version 3.3.7 - built Nov 20 2006 13:05:04
Copyright (C) 2002-2003 RealVNC Ltd.
Copyright (C) 1994-2000 AT&T Laboratories Cambridge.
See http://www.realvnc.com for information on VNC.
VNC server supports protocol version 3.8 (viewer 3.3)
Password:
VNC authentication succeeded
Desktop name "RHELv4u3.localdomain:2 (root)"
Connected to VNC server, using protocol version 3.3
...
```

If you don't like the simple twm window manager, you can comment out the last two lines of `~/vnc/xstartup` and add a **gnome-session &** line to have vnc default to gnome instead.

```
[root@RHELv4u3 ~]# cat .vnc/xstartup
#!/bin/sh

# Uncomment the following two lines for normal desktop:
# unset SESSION_MANAGER
# exec /etc/X11/xinit/xinitrc

[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey
vncconfig -iconic &
# xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
# twm &
gnome-session &
[root@RHELv4u3 ~]#
```

Don't forget to restart your vnc server after changing this file.

```
[root@RHELv4u3 ~]# vncserver -kill :2
Killing Xvnc process ID 5785
[root@RHELv4u3 ~]# vncserver :2

New 'RHELv4u3.localdomain:2 (root)' desktop is RHELv4u3.localdomain:2

Starting applications specified in /root/.vnc/xstartup
Log file is /root/.vnc/RHELv4u3.localdomain:2.log
```

F.4. Practice VNC

1. Use VNC to connect from one machine to another.

Appendix G. cloning

G.1. About cloning

You can have distinct goals for cloning a server. For instance a clone can be a cold iron backup system used for manual disaster recovery of a service. Or a clone can be created to serve in a test environment. Or you might want to make an almost identical server. Let's take a look at some offline and online ways to create a clone of a Linux server.

G.2. About offline cloning

The term offline cloning is used when you power off the running Linux server to create the clone. This method is easy since we don't have to consider open files and we don't have to skip virtual file systems like `/dev` or `/sys`. The offline cloning method can be broken down into these steps:

1. Boot source and target server with a bootable CD
2. Partition, format and mount volumes on the target server
3. Copy files/partitions from source to target over the network

The first step is trivial. The second step is explained in the Disk Management chapter. For the third step, you can use a combination of `ssh` or `netcat` with `cp`, `dd`, `dump` and `restore`, `tar`, `cpio`, `rsync` or even `cat`.

G.3. Offline cloning example

We have a working Red Hat Enterprise Linux 5 server, and we want a perfect copy of it on newer hardware. First thing to do is discover the disk layout.

```
[root@RHEL5 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       15G   4.5G  9.3G  33% /
/dev/sda1       99M   31M   64M  33% /boot
```

The `/boot` partition is small but big enough. If we create an identical partition, then `dd` should be a good cloning option. Suppose the `/` partition needs to be enlarged on the target system. The best option then is to use a combination of `dump` and `restore`. Remember that `dd` copies blocks, whereas `dump/restore` copies files.

The first step to do is to boot the target server with a live CD and partition the target disk. To do this we use the Red Hat Enterprise Linux 5 install CD. At the CD boot prompt we type "linux rescue". The cd boots into a root console where we can use `fdisk` to discover and prepare the attached disks.

When the partitions are created and have their filesystem, then we can use `dd` to copy the `/boot` partition.

```
ssh root@192.168.1.40 "dd if=/dev/sda1 | dd of=/dev/sda1"
```

Then we use a `dump` and `restore` combo to copy the `/` partition.

```
mkdir /mnt/x  
mount /dev/sda2 /mnt/x  
cd /mnt/x  
ssh root@192.168.1.40 "dump -0 -f - /" | restore -r -f -"
```

Appendix H. License

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles

are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either

commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

License

* B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

* C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

* D. Preserve all the copyright notices of the Document.

* E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

* F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

* G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

* H. Include an unaltered copy of this License.

* I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

* J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

* L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

* M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

* N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

* O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of,

License

you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies

that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

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/dmesg, 274
/bin/ksh, 73, 204
/bin/login, 370
/bin/rm, 77
/bin/sh, 73
/boot, 55
/boot/grub, 55
/boot/grub/, 357
/boot/grub/grub.cfg, 55
/boot/grub/grub.conf, 55, 358
/boot/grub/menu.lst, 357
/dev, 36, 59, 284
/dev/hdX, 272
/dev/ht, 518
/dev/nst, 517
/dev/null, 59, 117
/dev/pts/1, 59
/dev/random, 70
/dev/sdb, 309
/dev/sdX, 272
/dev/st, 517
/dev/tty1, 59
/dev/urandom, 69, 71
/dev/zero, 70
/etc, 55
/etc/apache, 659
/etc/apache2/apache2.conf, 660
/etc/apt/sources.list, 416
/etc/at.allow, 385
/etc/at.deny, 385
/etc/bashrc, 205
/etc/cron.allow, 386
/etc/cron.d, 387
/etc/cron.deny, 386
/etc/crontab, 387
/etc/default/useradd, 190
/etc/ethers, 715
/etc/exports, 483, 493
/etc/filesystems, 292, 299
/etc/fstab, 232, 295, 301, 308, 404, 484, 493, 718
/etc/group, 208, 215, 684
/etc/gshadow, 210
/etc/hostname, 447
/etc/hosts, 71
/etc/httpd, 659
/etc/httpd/conf/httpd.conf, 660
/etc/inetd.conf, 490, 542
/etc/init.d/, 55, 372, 373
/etc/init.d/rc, 370
/etc/init.d/rcS, 369
/etc/init.d/samba, 531
/etc/init.d/smb, 531
/etc/init.d/winbind, 532
/etc/inittab, 367, 369, 370
/etc/inputrc, 204
/etc/login.defs, 194
/etc/lvm/.cache, 333
/etc/modprobe.conf, 507
/etc/modprobe.d/, 507
/etc/mtab, 300, 368
/etc/named.conf, 615
/etc/network/interfaces, 441, 462, 465
/etc/nsswitch.conf, 582, 584
/etc/passwd, 132, 189, 195, 195, 197, 215, 370, 591, 684
/etc/profile, 204
/etc/protocols, 438
/etc/raidtab, 316
/etc/rc.d/rc, 370
/etc/rc.d/rc.sysinit, 368
/etc/rcS.d/, 369
/etc/rcX.d/, 369
/etc/resolv.conf, 71, 603
/etc/samba/passdb.tdb, 591
/etc/samba/smb.conf, 536, 537, 538, 555, 580
/etc/samba/smbpasswd, 561, 588
/etc/selinux/config, 701
/etc/services, 438, 490
/etc/shadow, 191, 193, 227
/etc/shells, 158, 197

/etc/shutdown.allow, 378
/etc/skel, 56, 196
/etc/squid/squid.conf, 665
/etc/ssh, 470
/etc/ssh/ssh_config, 470
/etc/ssh/sshd_config, 470
/etc/sudoers, 199, 199
/etc/sysconfig, 56
/etc/sysconfig/firstboot, 56
/etc/sysconfig/harddisks, 56
/etc/sysconfig/hwconf, 56
/etc/sysconfig/iptables, 486
/etc/sysconfig/keyboard, 56
/etc/sysconfig/network, 443
/etc/sysconfig/network-scripts/, 443
/etc/sysconfig/network-scripts/ifcfg-bond0, 463
/etc/sysctl.conf, 645
/etc/syslog.conf, 394, 395
/etc/X11/xorg.conf, 55
/etc/xinetd.conf, 489
/etc/xinetd.d, 489
/etc/xinetd.d/swat, 542
/etc/yum.conf, 423
/etc/yum.repos.d/, 423
/export, 57
/home, 57
/lib, 54, 513
/lib/kbd/keymaps/, 56
/lib/modules, 54, 504, 509
/lib/modules/<kernel-version>/modules.dep, 506
/lib32, 54
/lib64, 54
/media, 57
/opt, 54
/proc, 36, 60
/proc/bus, 711
/proc/bus/pci, 712
/proc/bus/usb/devices, 711
/proc/cmdline, 497
/proc/cpuinfo, 61
/proc/devices, 284, 284
/proc/dma, 714
/proc/filesystems, 292, 299
/proc/interrupts, 62, 712
/proc/ioports, 713
/proc/kallsyms, 503
/proc/kcore, 62
/proc/mdstat, 316
/proc/meminfo, 401, 402
/proc/modules, 504
/proc/mounts, 300
/proc/net/bonding, 463, 465
/proc/partitions, 284
/proc/scsi/scsi, 276
/proc/swaps, 403
/proc/sys, 61
/proc/sys/net/ipv4/ip_forward, 645
/root, 57
/root/anaconda-ks.cfg, 716
/run, 67
/sbin, 53, 76, 445
/sbin/init, 367
/sbin/mingetty, 370
/sbin/telinit, 376
/selinux, 700
/selinux/enforce, 700
/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, 513
/usr/local, 64
/usr/sbin/system-config-kickstart, 716
/usr/share, 65
/usr/share/doc, 443
/usr/share/games, 65
/usr/share/man, 65
/usr/src, 65, 499
/var, 66
/var/cache, 66
/var/lib, 67
/var/lib/nfs/etab, 483, 493
/var/lib/rpm, 67, 418
/var/lib/usbutils/usb.ids, 711
/var/lock, 67
/var/log, 66
/var/log/audit/audit.log, 699
/var/log/auth.log, 393
/var/log/btmp, 391, 392

-
- /var/log/lastlog, 391
 - /var/log/messages, 66, 497, 512
 - /var/log/secure, 393
 - /var/log/squid, 666
 - /var/log/syslog, 66
 - /var/log/wtmp, 377, 391
 - /var/run, 67
 - /var/run/utmp, 391
 - /var/spool, 67
 - /var/tmp, 67
 - ., 27
 - ./configure, 426
 - .., 27
 - .. (directory), 237
 - . (directory), 237
 - . (shell), 159
 - .bash_history, 102
 - .bash_login, 204
 - .bash_logout, 206
 - .bash_profile, 204
 - .bashrc, 204, 205
 - .deb, 408
 - .exrc, 153
 - .htaccess, 662
 - .htpasswd, 661
 - .my.cnf, 685
 - .rpm, 408
 - .ssh, 474
 - .vimrc, 153
 - `(backtick), 96
 - ~, 27
 - ~/.ssh/authorized_keys, 475
 - '(single quote), 96
 - " (double quotes), 75
 - (((shell), 179
 - (shell), 160
 - [(file globbing), 107
 - [(shell), 164
 - \$? (shell variables), 84
 - \$() embedded shell, 96
 - \$ (shell variables), 90
 - \$\$, 245
 - \$HISTFILE, 102
 - \$HISTFILESIZE, 102
 - \$HISTSIZE, 102
 - \$LANG, 108
 - \$PATH, 76, 91
 - \$PPID, 245
 - \$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**
- A (DNS record), 608
 - AAAA (DNS record), 608
 - access control list, 232
 - access time, 270
 - acl, 234
 - acls, 232
 - active partition, 361
 - agp, 711
 - AIX, 3
 - alias(bash), 77
 - alias(shell), 77
 - Alica and Bob, 471
 - allow hosts (Samba), 573
 - anycast, 435
 - apache2, 659
 - apropos, 23
 - apt-get(8), 409, 412
 - aptitude, 514, 528, 529
 - aptitude(1), 500
 - aptitude(8), 409, 415, 683
 - arguments(shell), 74
 - arp(1), 448
 - arp(protocol), 456
 - arp table, 448
 - at(1), 383, 384
 - ata, 270
 - atapi, 271
 - atm, 436
 - atq(1), 384
-

atrm(1), 384
auditd, 699
authoritative (dns), 611
authoritative zone, 607
axfr, 613

B

backticks, 96
badblocks(8), 277
base64, 118
bash, 171
bash history, 101
bash -x, 160
bg(1), 264
Bill Callkins, 356
binaries, 53
bind(DNS), 626
binding, 461
binding(ip), 460
BIOS, 355
block device, 270
bonding(ip), 460
boot(grub), 359
bootloader, 357
bootp, 443, 456, 715
Bourne again shell, 73
broadcast, 435
browsable (Samba), 573
browseable (Samba), 573
browser master, 588
BSD, 3, 355
btrfs, 292
bum(8), 375
bunzip2, 141
bus, 711
bzcat, 141
bzImage, 359
bzip2, 140, 141, 141
bzip2(1), 359, 518
bzip2, 141

C

cable select, 271
caching only name server, 609
cal, 139
Canonical, 367
case, 181
case sensitive, 36

cat, 124
cat(1), 46
cd(bash builtin), 27
cd -(bash builtin), 28
CentOS, 5
chage(1), 194
chain(iptables), 649
chainloader, 361
chainloading, 361
char(mysql), 688
character device, 270
chattr(1), 521
chcon(1), 702, 703
chgrp(1), 215
chkconfig, 56, 372, 699
chkconfig(8), 372
chmod, 196, 220, 700
chmod(1), 150, 218
chmod +x, 158, 221
chown, 196
chown(1), 215
CHS, 270
chsh(1), 197
CIFS, 533
Cisco, 437
CMDBA, 708
CMDEV, 708
CNAME (DNS record), 608
comm(1), 129
command line scan, 74
command mode(vi), 147
copyleft, 8
copyright, 7, 7
cp(1), 38, 38
cpio(1), 418, 522
cpu, 711
create(mysql), 686, 688, 695
create mask (Samba), 574
cron(8), 383
crontab(1), 386
crontab(5), 386
crypt, 192
csh, 158
Ctrl-Alt-Delete, 377, 378
Ctrl d, 46
ctrl-r, 102
Ctrl-Z, 263
current directory, 27

cut, 132
cut(1), 126
cylinder, 270

D

daemon, 23, 244, 371
date, 138
dd(1), 287, 356, 403, 522
deb(5), 409
Debian, 5
debsums, 514
default(grub), 358, 360
default gateway, 449
delete(mysql), 693
Dennis Ritchie, 3
deny hosts (Samba), 573
depmod(1), 506
describe(mysql), 689
devfs, 63
device driver, 284
devices.txt, 284
df(1), 301, 301
df -i, 236
dhclient(1), 446
dhcp, 443, 456
dhcp client, 441, 446
dhcpd.conf, 716
dhcp server, 603
directory, 237, 291
directory mask (Samba), 574
directory security mask(samba), 574
disk platters, 270
distribution, 4
distributions, 52
dma, 714
dmesg(1), 274, 713, 714
dmesg(8), 498
DNAT, 644
dns, 456, 602, 602
dns namespace, 604
dns server, 603
domain (dns), 605
domain(selinux), 701
domainname, 607
domain name system, 602, 602
DOS, 361
dpkg, 528
dpkg(1), 683

dpkg(8), 409, 411
dpkg -S, 514
drop(mysql), 687, 689, 695
dsa, 471
du(1), 301
dump(1), 521
dumpkeys(1), 56

E

e2fsck(1), 295
echo, 74
echo(1), 74, 75, 245
echo \$-, 95
echo *, 109
edquota(1), 718
Edubuntu, 5
egrep, 368
eiciel, 234
ELF, 54
elif, 165
elilo, 357
el torito, 292
embedding(shell), 96
env(1), 93, 93
environment variable, 90
EOF, 118
Eric Allman, 394
escaping (shell), 109
eth0, 441
ethtool(1), 450
eval, 179
Evi Nemeth, 371
exec, 246
executables, 53
exit (bash), 102
export, 93
exportfs(1), 483, 493
ext2, 291, 294
ext3, 291
extended partition, 283

F

fallback(grub), 358
fat16, 292
fat32, 292
fd (partition type), 315
fddi, 436
fdisk, 348

fdisk(1), 284, 285, 286, 315
fdisk(8), 273
Fedora, 5
fg(1), 264
FHS, 52
file(1), 36, 54
file globbing, 106
file ownership, 215
file system, 290
Filesystem Hierarchy Standard, 52
filters, 123
find(1), 137, 226, 227, 238
firewall, 643
FireWire, 63
fixed ip, 443
fixed ip address, 441
for (bash), 165
force create mode(samba), 574
force directory mode(samba), 574
force directory security mode(samba), 574
force group(samba), 561
force security mode(samba), 574
force user(samba), 561
fork, 246
forwarder (dns), 609
forward lookup query, 603
FOSS, 7
four freedoms, 8
FQDN, 447
fqdn, 607
frame relay, 436
free(1), 401, 402
Free Software, 7
free software, 8
freeware, 7
fsck(1), 295
ftp, 489
ftp://ftp.kernel.org, 498
fully qualified domain name, 607
function (shell), 182

G

gateway, 449
gcc(1), 193
getenforce, 698
getent(1), 584
getfacl, 232
getfattr(1), 703

getopts, 174
GID, 208
glob(7), 107
glue record (dns), 608
gnome-session, 720
GNU, 3
gpasswd, 210
GPL, 8
GPLv3, 8
grant(mysql), 687
grep, 368, 504
grep(1), 124
grep -i, 124
grep -v, 125
groupadd(1), 208
group by(mysql), 693
groupdel(1), 209
groupmod(1), 209
groups, 208
groups(1), 209
grpquota, 718
grub, 357, 357, 361
grub-install, 362
guest ok (Samba), 548
gunzip(1), 140
gzip, 140
gzip(1), 140, 359, 518

H

halt(8), 377
hard link, 238
hdparm(8), 278
head(1), 45
head (hard disk device), 270
here directive, 47
here document, 118
here string, 118
hidden files, 29
hiddenmenu(grub), 358
hide unreadable (Samba), 573
host (DNS record), 608
hostname, 447, 533, 607
hostname(1), 447
hosts.txt, 602
hosts allow (Samba), 573
hosts deny (Samba), 573
HP, 3
HP-UX, 3

htpasswd(1), 661
http://www.kernel.org, 498
http://www.pathname.com/fhs/, 52
httpd, 659

I

IBM, 3, 533
icmp, 438
id_dsa, 475
id_dsa.pub, 475
id_rsa, 475
id_rsa.pub, 475
id(1), 188, 702
ide, 284
identity(selinux), 701
idmap gid(samba), 580
idmap uid(samba), 580
IEEE 1394, 63
ifcfg(1), 461
ifcfg-eth0, 444
ifconfig(1), 445, 446, 461, 462, 463, 465
ifdown(1), 442, 444, 446, 461
ifenslave, 465
if then else (bash), 165
ifup(1), 442, 444, 446, 461, 462, 463
igmp, 438
inetd, 489
inetd(8), 542
init, 244, 367, 377
init=/bin/bash, 497
initiator(iSCSI), 345
initng, 367
initrd, 502
initrd(grub), 360
inode, 235, 238
inode table, 236
insert(mysql), 690
insert mode(vi), 147
insmod(1), 505, 506
integer(mysql), 688
Intel, 355
interrupt, 712
invalid users (Samba), 572
IO Ports, 713
iptables, 486, 649
iptables save, 651
IRQ, 712
isa, 711

iSCSI, 345
iscsiadm, 348
iso9660, 292, 523
iterative query, 610
ixfr, 613

J

jbod, 313
jobs, 263
joliet, 292
journaling, 291
Jumpstart, 715

K

Ken Thompson, 3
Kerberos, 482, 493
kernel, 54
kernel(grub), 359
keymaps(5), 56
kickstart, 715, 716
kill(1), 244, 249, 249, 371, 371
killall(1), 250
kmyfirewall, 486
Korn shell, 103
Korn Shell, 197
ks.cfg, 716
ksh, 103, 158
kudzu, 56

L

LAMP, 682
LAN, 436
last(1), 377, 391
lastb(1), 392
lastlog(1), 391
LBA, 270
ldap, 483
ldd, 513
less(1), 48
let, 180
libraries, 513
lilo, 357, 357, 362
lilo.conf, 362
Linus Torvalds, 3
Linux Mint, 5
ln, 239
ln(1), 238
loadkeys(1), 56
locate(1), 138

logger(1), 396
 logical AND, 85
 logical drive, 283
 logical drives, 287
 logical OR, 85
 Logiciel Libre, 8
 login, 391
 logrotate(1), 397
 LPIC 1 Certification, 707
 LPIC 2 Certification, 707
 ls, 217, 236, 700
 ls(1), 29, 29, 236, 237, 703
 ls -l, 216
 lsmod, 504
 lsmod(1), 504
 lspci, 712
 lsscsi(1), 275
 lsusb, 711
 ltrace, 514
 lvcreate(1), 323, 325, 340
 lvdisplay(1), 326, 335
 lvextend(1), 326, 341
 LVM, 321
 lvmdiskscan(1), 331
 lvol0, 340
 lvremove(1), 340
 lvrename(1), 341
 lvs(1), 335
 lvscan(1), 335

M

mac address, 446
 magic(5), 36
 major number, 284
 make, 511
 make(1), 426
 make bzImage, 508
 make clean, 508
 make menuconfig, 508
 make modules, 509
 make mrproper, 508
 make xconfig, 508
 MAN, 437
 man(1), 23, 24, 24
 mandb(1), 25
 man hier, 52
 man -k, 23
 master (hard disk device), 271

master boot record, 287, 356
 master server (DNS), 612
 mbr, 287, 287, 356
 MBR, 523
 md5, 193
 mdadm(1), 316
 mingetty, 370
 minor number, 284
 mirror, 313
 mkdir, 196, 299
 mkdir(1), 31, 221
 mkdir -p, 31
 mke2fs(1), 291, 294, 325
 mkfifo, 256
 mkfile(1), 403
 mkfs, 236
 mkfs(1), 291, 294
 mkinitrd(1), 291, 510
 mknod(1), 517
 mkswap(1), 403
 modinfo, 512
 modinfo(1), 505
 modprobe(1), 506, 507
 more(1), 48
 mount, 299
 mount(1), 298, 300, 484, 493
 mounting, 298
 mount point, 299
 mt(1), 518
 multicast, 434
 mv(1), 39
 MX (DNS record), 608
 mysql, 682, 684, 685, 686
 mysql(group), 684
 mysql(user), 684
 mysql-client, 683
 mysqld, 684
 mysql-server, 683

N

NAPT, 644
 NAT, 643
 NetBIOS names, 533
 netcat, 551
 net groupmap, 593
 net rpc join(samba), 581
 netstat(1), 449
 net use(microsoft), 550, 555, 566

net view(microsoft), 536, 542
network file system, 481
nfs, 481, 482, 715
NFS, 492
nice, 258
nice(1), 256
nmbd(8), 532
no_subtree_check(nfs), 483
noacl(mount), 303
noclobber, 115
nodev, 292, 299
noexec(mount), 302
nosuid(mount), 303
nounset(shell), 94
Novell Certified Linux Professional, 708
NS (DNS record), 608
nslookup, 603
NT_STATUS_BAD_NETWORK_NAME,
567
NT_STATUS_LOGON_FAILURE, 567

O

octal permissions, 220
od(1), 130, 357
OEL, 5
OpenBoot(Sun), 356
OpenBSD, 470
open source, 8
open source definition, 8
open source software, 7
openssh, 470
openssh-server, 477
openssl(1), 192
Oracle Enterprise Linux, 5
order by(mysql), 692
OS/2, 361
owner, 217

P

package management, 408
packet filtering, 643
packet forwarding, 643
paging, 401
PAN, 437
Parallel ATA, 271
parent directory, 27
parity(raid), 313
parted(1), 285

partition, 283
partition table, 287, 287
partprobe(1), 287
passdb backend (Samba), 561
passwd, 194
passwd(1), 24, 191, 191, 192, 227
passwd(5), 24
password(grub), 359
PAT, 644
path, 28, 29
Paul Mockapetris, 602
pc-card, 711
pci, 711
pci-express, 711
pcmcia, 711
pgrep(1), 247
php, 682
PID, 244
pidof(1), 245
ping, 438, 449
pipe, 120
pipes, 256
pkill(1), 250
policy(SELinux), 703
popd, 34
port forwarding, 644
portmap, 482, 492
POST, 355
poweroff(8), 377
Power On Self Test, 355
PPID, 244
primary dns server, 611
primary group, 190
primary partition, 283, 356, 361
primary server (DNS), 612
private key, 471
process, 244
process id, 244
proprietary, 7
proxy server, 664
ps, 246
ps(1), 703
ps -ef, 247
ps fax, 247
PTR (DNS record), 608
public domain, 7
public key, 471
pushd, 34

pvchange(1), 337
pvcreate(1), 323, 325, 336
pvdisplay(1), 325, 332
pvmove(1), 337
pvremove(1), 336
pvresize(1), 336
pvs(1), 331
pvscan(1), 331
pwd(1), 27, 28

Q

query (dns), 603
quota.group, 718
quota.user, 718
quota's, 718
quota(1), 718
quotacheck(1), 718
quotaoff(1), 718
quotaon(1), 718

R

RAID, 312
raid 1, 313
random number generator, 70
rarp, 715
read, 172
read list (Samba), 572
read only (Samba), 555
reboot, 102
reboot(8), 377
recursive query, 610
Red Hat, 5
regular expressions, 103
reiserfs, 292
Remote Desktop, 719
rename(1), 40
renice, 257
renice(1), 256
repository, 4, 408, 409
repquota(1), 718
resize2fs(1), 326
respawn(init), 370, 370
restore(1), 521
reverse lookup query, 603
rfc 3010, 482
rfc 3530, 482
RHCE, 707
Richard Stallman, 3

rlogin, 470
rm(1), 37, 239
rmdir(1), 31
rmdir -p, 31
rmmod(1), 506
rm -rf, 38
roaming profiles(samba), 592
rock ridge, 292
role(selinux), 702
root, 53, 189, 198, 199, 199
root(DNS), 604
root(grub), 360
root(mysql), 683
root directory, 52
root hints, 605
root server (dns), 610
root servers(DNS), 435
root servers (dns), 604
rootsquash, 483, 493
rotational latency, 270
route(1), 449, 449
router, 437
rpc, 482
RPC, 492
rpcinfo(1), 482, 492
rpm, 67, 417, 528
rpm(1), 683
rpm(8), 409, 529
rpm2cpio(8), 418
rpm -qf, 514
rpm -V, 515
rsa, 471
rsh, 470
runlevel, 367
runlevel(1), 376

S

salt (encryption), 193
samba, 528
sample.ks, 716
sata, 271
savedefault(grub), 360
Scientific, 5
scp(1), 475
scsi, 270
scsi_info(1), 276
scsi id, 271
secondary dns server, 611

secondary server (DNS), 612
sector, 270
security(Samba), 548
security mask(samba), 574
security mode(samba), 565
sed, 131
seek time, 270
select(mysql), 690, 691, 691
SELinux, 697
selinux, 699
selinux-activate, 698
service(1), 372, 486
service(8), 531
sestatus, 699
set, 95
set(shell), 92
set +x, 78
setenforce, 698
setfacl, 232
setgid, 226, 226
setuid, 160, 199, 227, 227, 303
set -x, 78
sfdisk(1), 287
she-bang (shell), 158
shell, 204
shell comment, 86
shell escaping, 86
shell expansion, 74, 74
shell functions, 182
shift, 172
shopt, 175
show(mysql), 686, 688
shutdown(8), 376
SIGHUP, 249
SIGKILL, 376
SIGTERM, 250, 376
silo, 357
single user mode, 497
skeleton, 56
slave (hard disk device), 271
slave server (DNS), 612
sleep, 139
SMB, 533
smbclient, 540, 549
smbclient(1), 539, 566
smbd(8), 532, 536, 560
smbpasswd(1), 593
smbpasswd(8), 560, 565
smbtree, 541
smbtree(1), 540
SMF, 367
smtp, 608
SNAT, 644
soa (dns record), 611
soft link, 239
Solaris, 3, 355
sort, 132
sort(1), 128
source, 159, 173
SPARC, 356
split(1), 524
SQL, 682, 690
squid, 664, 665
ssh, 470
ssh_host_dsa_key, 477
ssh_host_dsa_key.pub, 477
ssh_host_rsa_key, 477
ssh_host_rsa_key.pub, 477
sshd, 477
ssh-keygen, 475
ssh-keygen(1), 474
ssh -X, 476
stanza(grub), 359
stateful firewall, 643
stderr, 115
stdin, 115, 120, 124
stdout, 115, 120, 124
sticky bit, 226
strace, 515
strings(1), 48
striped disk, 313
su, 195, 210, 513
su -, 91
su(1), 198, 198
subtree_check(nfs), 483
sudo, 195, 199
sudo(1), 199
sudo su -, 200
Sun, 3, 355, 367
SunOS, 3
superuser, 189
swapoff(1), 403
swapon(1), 403
swap partition, 292
swap partition(s), 405
swapping, 401

swap space, 403
swat, 489
swat(8), 542
symbolic link, 239
sysctl(1), 447
sysfs, 63
syslog, 497
syslogd, 394
System.map, 503
system-config-securitylevel, 486
System V, 54, 367

T

tab key(bash), 29
tac(1), 47
tail(1), 45, 396
tar(1), 426, 519, 520
tcp, 438, 482
tcpdump, 454, 457
tdbsam, 561, 588, 591
tee(1), 124
telinit(8), 376
telnet, 470, 489
test, 164
testparm(1), 537, 537, 538
tftp, 715
time, 139
time(1), 508
timeout(grub), 358
title(grub), 359
tld, 606
TLD (dns), 606
top, 250
top(1), 248, 401, 402
top level domain, 606
touch(1), 37
tr, 127
tr(1), 126
track, 270
transition(selinux), 702
trigger(mysql), 695
triggers(mysql), 683
tune2fs(1), 291, 294, 308
type(selinux), 701
type(shell), 76

U

Ubuntu, 5

udf, 292
udp, 438, 482
umask(1), 221
unalias(bash), 78
uname(1), 496
uniq, 132
uniq(1), 129
universally unique identifier, 308
Unix, 3
unset, 95
unset(shell), 92
until (bash), 166
update(mysql), 691
updatedb(1), 138
update-rc.d, 372
update-rc.d(8), 374
upstart, 367
usb, 63, 711
use(mysql), 687
useradd, 190, 196
useradd(1), 192, 196
useradd -D, 190
userdel(1), 190
usermod, 209
usermod(1), 190, 194, 195
usrquota, 718
uuid, 308

V

valid users (Samba), 572
vanilla, 409
varchar(mysql), 688
vfat, 292
vgchange(1), 339
vgcreate(1), 323, 325, 338
vgdisplay(1), 333
vgextend(1), 338
vgmerge(1), 339
vgreduce(1), 338
vgremove(1), 338
vgs(1), 333
vgscan(1), 333
vi, 210, 515
vi(1), 146
vigr(1), 210
vim(1), 146
vimtutor(1), 146
vipw(1), 195

virtual memory, 401
visudo(1), 199
vmlinuz, 502
vmstat, 405
vnc, 719
vncviewer(1), 719
vol_id(1), 308
vrije software, 8

W

w(1), 188
WAN, 436
watch(1), 396
wbinfo(1), 583, 584
wc(1), 127
webalizer, 662
whatis(1), 23
whereis(1), 24
which(1), 76
while (bash), 166
white space(shell), 74
who, 132
who(1), 188, 376, 391
who am i, 188
whoami(1), 188
wild cards, 108
winbind(8), 582
winbind(samba), 580
winbindd(8), 532, 532, 582
wireshark, 454, 470
workgroup, 548
WPAN, 437
writable (Samba), 555
write list (Samba), 572

X

X, 55
X.25, 436
x86, 355
xinetd, 489, 489
xinetd(8), 542
xstartup(vnc), 720
X Window System, 55

Y

yaboot, 357
yum, 515, 529
yum(8), 420

Z

z/IPL, 357
zcat, 140
zfs, 292
zImage, 359
zmore, 140
zombie, 244
zone (dns), 607, 611
zone transfer (dns), 611