Linux Introduction

Paul Cobbaut Bert Van Vreckem

September 18, 2024

Contents

Feel free to contact the author(s):

- Paul Cobbaut (Netsec BVBA): paul.cobbaut@gmail.com, https://cobbaut.be/
- Bert Van Vreckem (HOGENT): http://github.com/bertvv

This copy was generated on September 18, 2024.

This book covers the topics introduced in the course "Computer Systems" for the Bachelor of Applied Computer Science at the HOGENT, Belgium. The course consists partly of a basic introduction to the Linux operating system. This book is not the official handbook of that course, but is targeted at students who need a refresher on the topics covered.

The content is based on the Linux Training book series by Paul Cobbaut, with updates and additions written by the HOGENT Linux team.

More information and free .pdf available at https://hogenttin.github.io/linux-training-hogent/.

0.1. Conventions used

The contributors to this work have taken great care that the examples with command line interactions are correct and work as expected. However, sometimes the output of a command may differ slightly from the examples in this book. This can be due to differences in the version of the software, the operating system, the environment in which the command is executed, or the specific state of the system at the time of execution.

Some Linux distributions may have commands that behave differently than their counterparts in other distributions. This is especially true for the package management commands.

Command line examples are shown in a monospaced font with a prompt that indicates the user and hostname in the form `user@hostname:current_directory$`. For example:

```
student@debian:~$ ls
root@linux:~# ls
```

We follow the following conventions:

- Regular user vs root:
  - A regular user prompt is shown as `$`, the user name is generally `student`.
  - A root prompt (with elevated privileges) is shown as `#`.
- The linux distribution is indicated by the host name:
  - If the command should work on any Linux distribution, the hostname is `linux`.
  - If the command is specific to a certain distribution, the hostname is the name of that distribution, e.g. `debian`, `ubuntu`, `rhel`, etc.
  - If you see `el` as the host name (short for Enterprise Linux), you can assume that it was tested on an Alma Linux machine and that it should work on any Red Hat-based distribution.
- Commands that are run on a prompt that is not necessarily a Linux system (e.g. you're running Linux in a VM on a Windows host) are shown with a generic `>` prompt, e.g. `> winget install Git.Git`.

0.2. Reporting errors

Did you find an error in this book, or is the output of a command considerably different from what is shown? Please report it by creating an issue on the linux-training-hogent GitHub repository.

Part I.

introduction to Linux

1. Linux history

(Written by Paul Cobbaut, https://github.com/paulcobbaut/)

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 8. Working with Directories` page 73.

1.1. 1969

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`: the official AT&T Unix, and the free `BSD` Unix.

Development of BSD descendants like FreeBSD, OpenBSD, NetBSD, DragonFly BSD and PC-BSD is still active today.

```
https://en.wikipedia.org/wiki/Dennis_Ritchie
https://en.wikipedia.org/wiki/Ken_Thompson
https://en.wikipedia.org/wiki/BSD
https://en.wikipedia.org/wiki/Comparison_of_BSD_operating_systems
```

1.2. 1980s

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` are GNU tools.

```
https://en.wikipedia.org/wiki/Richard_Stallman
https://en.wikipedia.org/wiki/IBM_AIX
https://en.wikipedia.org/wiki/HP-UX
```

1.3. 1990s

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.

```
http://en.wikipedia.org/wiki/Linus_Torvalds
https://en.wikipedia.org/wiki/History_of_Linux
https://en.wikipedia.org/wiki/Linux
https://lwn.net
http://www.levenez.com/unix/   (a huge Unix history poster)
```

1.4. 2015

Today more than 97 percent of the world's supercomputers (including the complete top 10), more than 80 percent 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, self-driving cars, space station laptops...) run `Linux`. Linux is by far the most commonly used operating system in the world.

Linux kernel version 4.0 was released in April 2015. Its source code grew by several hundred thousand lines (compared to version 3.19 from February 2015) thanks to contributions of thousands of developers paid by hundreds of commercial companies including Red Hat, Intel, Samsung, Broadcom, Texas Instruments, IBM, Novell, Qualcomm, Nokia, Oracle, Google, AMD and even Microsoft (and many more).

```
http://kernelnewbies.org/DevelopmentStatistics
http://kernel.org
http://www.top500.org
```

2. distributions

(Written by Paul Cobbaut, https://github.com/paulcobbaut/ with contributions by Bert Van Vreckem https://github.com/bertvv/)

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. Linux and GNU

The Linux Kernel project was started by Linus Torvalds in 1991 while he was a computer science student. He wanted to run a UNIX-like operating system on his own PC. Now, a kernel in itself is not a complete operating system. The kernel does not provide a terminal, tools to manage files, etc. However, the GNU project (which stands for *GNU's Not UNIX*), started by Richard Stallman, had been working on a complete operating system since 1983. The GNU project had a lot of the necessary tools and libraries to make a complete POSIX-compliant operating system, a.o. the GNU Compiler Collection (GCC), the GNU C Library (glibc), the GNU Core Utilities (coreutils), the GNU Bash shell, etc. They were also working on a kernel, called GNU Hurd, but development was prohibitively slow. Indeed, it was not until 2015 that the Hurd kernel was ready to be actually used.

Long story short, the Linux kernel in combination with the GNU tools and libraries made a complete operating system. This is why the operating system is often referred to as *GNU/Linux*. Both Linux as the GNU projects are open source and released under the GNU General Public License. This made it easy for third parties to redistribute GNU+Linux and add other compatible (open source) software packages to form a complete operating system with everything an end user needs to be productive on the computer. This is what we call a *Linux distribution*. The oldest still active distribution is Slackware, which was started in 1993 by Patrick Volkerding. Since then, many distributions have been created, each with their own goals and target audience. Some distributions (or distro's in short) are built from the ground up, but others are based on existing distributions, leading to large "families" of like-minded distro's.

Writing a comprehensive overview of all Linux distributions is way beyond the scope of this course, but it is useful to know about some of the main ones. If you want to know more about a specific distribution, you can check out the DistroWatch website, which is a great resource for information about Linux distributions.

2.2. Package management

One of the central and identifying components of a Linux Distribution is the default selection of software and the package management system to install, update and remove software. For most applications, there is choice in the open source world, so different distributions will

make different decisions on what to include and what to avoid. Sometimes this is regrettably the cause of dispute and drama in the Linux community, but on the other hand, it is also the driver of a lot of innovation and diversity and it empowers the user with a lot of freedom of choice and control.

The package manager was actually one of the most important innovations that Linux pioneered in. It is a system that keeps track of all the software installed on a computer and allows the user to select and install new applications from online package repositories. Hotfixes or new releases of the software included in a distribution are made available in these repositories and can be downloaded and installed with a single command. This makes it very easy to keep a Linux system up to date and secure. When Apple introduced the App Store in 2008, it was actually a latecomer to the concept of a central secure software repository.

The concept of an open source package repository also enables reuse of software and libraries. Applications don't have to write their own code to do things like read and write files, manage memory, etc. They can use libraries that are already available on the system and that are used by other applications. The package manager also takes care of dependencies, which are other software packages that are required for the software to work. This makes it very easy to install complex software with a single command.

2.3. The Red Hat family of distributions

Red Hat is one of the first commercial companies that successfully leveraged open source software as a business strategy. They started in 1993 and grew in the next decades to become a billion dollar company. In 2019, Red Hat was acquired by IBM for 34 billion dollars and it still operates as an independent subsidiary.

The flagship product of Red Hat is Red Hat Enterprise Linux, or RHEL in short. RHEL is a commercial Linux distribution, but on release, the source code is made available. The business model of Red Hat is based on selling support contracts.

RHEL is a stable and secure operating system, with long support cycles, which is why it is widely used in enterprise environments where the stability of IT infrastructure is of paramount importance. Enterprise software vendors that target Linux as a platform, usually certify their software to run on RHEL. This is why RHEL is often used in data centers, cloud environments and other mission-critical systems.

In order to innovate on the RHEL platform, Red Hat is also involved in the development of the Fedora distribution. Fedora is a community-driven project that aims to be a cutting-edge, free and open source operating system that showcases the latest in free and open source software. It is used as a testbed for new technologies that will eventually make their way into RHEL. Fedora has a release cycle of 6 months. Where RHEL is particularly suited as a server operating system, Fedora is an excellent choice as a desktop operating system for power users and IT professionals.

Since RHEL is open source, it is in principle possible to create a compatible clone of RHEL, albeit without the support and without Red Hat branding. This is exactly what the CentOS project did for years. CentOS used to be a community driven project that aimed to be 100% (*bug-for-bug*) compatible with RHEL and based on the released source code of all software included in RHEL. However, in 2014, Red Hat acquired the CentOS project, and later, they announced that CentOS Linux was going to be replaced by CentOS Stream, which is a rolling release distribution "upstream" of RHEL. This means that CentOS Stream now takes the place between Fedora and RHEL, and it is no longer a 100% compatible clone of RHEL anymore.

This incensed many users and organizations that relied on CentOS as a free and compatible alternative to RHEL. The CentOS project was forked, and the Rocky Linux project was started by Gregory Kurtzer, who was also one of the original founders of CentOS. The goal of Rocky Linux is to be a 100% compatible replacement for CentOS Linux. Likewise, AlmaLinux was started by CloudLinux, another company that was involved in the CentOS project. These RHEL-like distributions are sometimes referred to as "Enterprise Linux" or EL.

Distinctive features of the Red Hat family of distributions are:

- The use of the `RPM` package format (Red Hat Package Management) and the `dnf` package manager
- The `systemd` init system
- The `firewalld` firewall management tool
- The *SELinux* security framework
- The *Anaconda* installer
- The *Cockpit* web-based management interface
- Their own container runtimes, *runc* and *crun* and management tools *podman* and *buildah* (instead of Docker)

Oracle Enterprise Linux is Oracle's commercial Linux distribution, put in the market as a direct competitor to RHEL. Scientific Linux was a community driven project that was used by scientific institutions like CERN and Fermilab, but it was discontinued in 2021. The final maintenance window for Scientific Linux 7 is June 30, 2024. After that, users are advised to migrate to AlmaLinux. The Amazon Linux distribution is a RHEL-like distribution that is used as the default operating system for Amazon Web Services (AWS) EC2 instances.

2.4. The Debian family of distributions

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 the well-known Ubuntu (see below). Debian comes in three versions: stable, testing and unstable. Every Debian release is named after a character in the movie Toy Story.

Canonical, a company founded by South African entrepreneur Mark Shuttleworth, 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 commercial support for Ubuntu. Ubuntu is known for their *Long Term Support* (LTS) releases, which are supported for 5 years (or 10 years for a fee). Intermediate releases come out every 6 months (in April and October) and are supported for 9 months. Releases are named after the year and month of the release, e.g. 19.10 for October 2019. LTS releases come out every even year in April, e.g. 22.04 and 24.04. Canonical also has the reputation of going their own way and doing things differently from the rest of the Linux community. For example, they developed their own init system, Upstart (which was later abandoned and replaced by systemd), and their own display server, Mir (which was later replaced by Wayland), a desktop environment (Unity, later replaced with Gnome), etc. Some of these decisions were controversial and have led to a lot of criticism, but the strength of the open source community lies precisely in the freedom to make different choices, which is a driver for innovation.

Distinctive features of the Debian family of distributions are:

- The use of the `deb` package format and the `apt` package manager (Advanced Package Tool)
- The `systemd` init system
- The `ufw` firewall management tool
- The *AppArmor* security framework
- The *Debian-installer* installer
- The Docker container runtime and management tools

Linux Mint, Edubuntu and many other distributions with a name ending on -buntu are based on Ubuntu and thus share a lot with Debian. Kali Linux is another Debian-based distribution that is specifically designed for digital forensics and penetration testing. It comes with a lot of pre-installed tools for hacking and security testing. Kali is not suitable for daily use as a

desktop operating system, but it is very popular among security professionals and hobbyists. The popular mini-computer Raspberry Pi has its own Debian-based distribution called Raspberry Pi OS.

2.5. Notable "independent" distributions

Apart from the two big families of distributions, i.e. Red Hat and Debian families, there are many other distributions that are not based on either of these. Some of the most notable ones are:

- Alpine Linux: an independent non-commercial, general purpose distribution with a focus on security and simplicity. Alpine Linux is very small and lightweight, and it is often used in containers.
- Arch Linux: another independent general purpose distribution. Arch Linux is a rolling release distribution, which means that you install it once and then continuously update individual packages when new versions become available. The distribution itself does not have an overarching (see what I did there?) release cycle. Arch has its own package manager, Pacman. One of the most notable features of Arch Linux is its outstanding documentation, which is very extensive and well written and even quite useful for users of other distributions. Installing Arch Linux is not as straightforward as installing other distributions: you start with a minimal system with the kernel and a shell, and then you build up the system to your own liking. This is not for novice users, but it is a great way to learn about the inner workings of a Linux system.
- openSUSE: a general purpose community driven distribution that is sponsored by SUSE, a German company that also offers commercial support for derivative distro's SUSE Linux Enterprise Server (SLES) and Desktop (SLED). openSUSE is known for its YaST (Yet another Setup Tool) configuration tool, which is a central place to configure many aspects of the system. openSUSE comes in two flavours: Leap and Tumbleweed. Leap is a regular release distribution with a fixed release cycle, while Tumbleweed is a rolling release distribution.

2.6. Which to choose?

If you ask 10 people what the best Linux distribution is, chances are that you will get 20 different answers. Posting it as a question on a forum may lead to a discussion that goes on for weeks or months, if not years. You will get a lot of passionate and sometimes even insightful opinions, but in the end you won't be none the wiser. So giving good advice that is universally applicable is very hard, indeed.

Below are some very personal opinions (albeit informed by experience) on some of the most popular Linux distributions. Keep in mind that any of the below Linux distributions can be a stable server and a nice graphical desktop client.

Distribution name	Reason(s) for using
AlmaLinux	You want a stable Red Hat-like server OS without commercial support contract.
Arch	You want to know how Linux *really* works and want to take your time to learn.
Debian	An excellent choice for servers, laptops, and any other device.
Fedora	You want a Red Hat-like OS on your laptop/desktop.
Kali	You want a pointy-clicky hacking interface.

Distribution name	Reason(s) for using
Linux Mint	You want a personal graphical desktop to play movies, music and games.
RHEL	You are a manager and need good commercial support.
RockyLinux	You want a stable Red Hat-like server OS without commercial support contract.
Ubuntu Desktop	Very popular, suited for beginners and based on Debian.
Ubuntu Server	(LTS particulary) You want a Debian-like OS with commercial support.

When you are new to Linux, and are looking for a distribution with a graphical desktop and all the tools that you need as a daily driver, check out the latest Linux Mint (suitable for computer novices and experienced computer users alike) or Fedora (recommended for power users and IT professionals).

If you only want to practice the Linux command line, or are interested in the use of Linux as a server, then install a VM with the latest release of either Debian stable and/or AlmaLinux (without graphical interface)[1].

As you gain experience, you can try out other distributions and see what you like best. Good luck on your journey and enjoy the ride!

[1] Remark that this advice was originally written in 2015 and basically still holds in 2024. The only amendment is that AlmaLinux has taken the place of CentOS as a recommendation for a server OS.

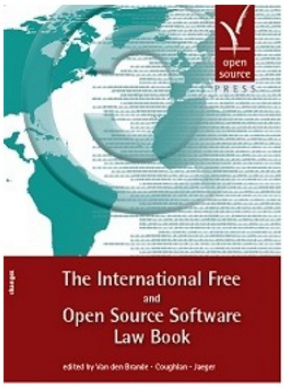
3. licensing

(Written by Ywein Van den Brande, with contributions by: Paul Cobbaut, https://github.com/paulcobbaut/)

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

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

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



`http://ifosslawbook.org`
`http://www.crealaw.eu`

3.1. about software licenses

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

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

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

3.2. public domain software and freeware

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

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

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

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

3.3. Free Software or Open Source Software

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

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

Examples of free software are gcc, MySQL and gimp.

Detailed information about the four freedoms can be found here:

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

The open source definition can be found at:

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

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

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

3.4. GNU General Public License

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

3.5. using GPLv3 software

You can use `GPLv3 software` almost without any conditions. If you solely run the software you even don't have to accept the terms of the GPLv3. However, any other use - such as modifying or distributing the software - implies acceptance.

In case you use the software internally (including over a network), you may modify the software without being obliged to distribute your modification. You may hire third parties to work on the software exclusively for you and under your direction and control. But if you modify the software and use it otherwise than merely internally, this will be considered as distribution. You must distribute your modifications under GPLv3 (the copyleft principle). Several more obligations apply if you distribute GPLv3 software. Check the GPLv3 license carefully.

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

3.6. BSD license

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

This is a permissive free software license. The license places minimal restrictions on how the software can be redistributed. This is in contrast to copyleft licenses such as the 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.

Part II.

command structure

4. commands and arguments

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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

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

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

```
student@linux:~$ echo Burtonville
Burtonville
student@linux:~$ echo Smurfs are blue
Smurfs are blue
```

4.1. 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.

4.2. 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`.

```
[student@linux ~]$ echo Hello World
Hello World
[student@linux ~]$ echo Hello   World
Hello World
[student@linux ~]$ echo   Hello   World
Hello World
[student@linux ~]$    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.

4.3. 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`.

```
[student@linux ~]$ echo 'A line with      single    quotes'
A line with      single    quotes
[student@linux ~]$
```

4.4. double quotes

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

```
[student@linux ~]$ echo "A line with      double    quotes"
A line with      double    quotes
[student@linux ~]$
```

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

4.5. 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).

```
[student@linux ~]$ echo -e "A line with \na newline"
A line with
a newline
[student@linux ~]$ echo -e 'A line with \na newline'
A line with
a newline
[student@linux ~]$ echo -e "A line with \ta tab"
A line with     a tab
[student@linux ~]$ echo -e 'A line with \ta tab'
A line with     a tab
[student@linux ~]$
```

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

4.6. commands

4.6.1. 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.

4.6.2. 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.

```
student@linux:~$ type cd
cd is a shell builtin
student@linux:~$ 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.

```
student@linux:~$ type ls
ls is aliased to `ls --color=auto'
```

4.6.3. 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.

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

4.6.4. 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@linux ~]# 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
```

4.7. aliases

4.7.1. 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.

```
[student@linux ~]$ cat count.txt
one
two
three
[student@linux ~]$ alias dog=tac
[student@linux ~]$ dog count.txt
three
two
one
```

4.7.2. abbreviate commands

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

```
student@linux:~$ alias ll='ls -lh --color=auto'
student@linux:~$ alias c='clear'
student@linux:~$
```

4.7.3. 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`.

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

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

4.7.4. 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.

```
student@linux:~$ alias c ll
alias c='clear'
alias ll='ls -lh --color=auto'
```

4.7.5. unalias

You can undo an alias with the `unalias` command.

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

4.8. 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.

```
[student@linux ~]$ set -x
++ echo -ne '\033]0;student@linux:~\007'
[student@linux ~]$ echo $USER
+ echo paul
paul
++ echo -ne '\033]0;student@linux:~\007'
[student@linux ~]$ echo \$USER
+ echo '$USER'
$USER
++ echo -ne '\033]0;student@linux:~\007'
[student@linux ~]$ set +x
+ set +x
[student@linux ~]$ echo $USER
paul
```

4.9. 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
```

17. 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).

18. Use one `echo` command to display three words on three lines.

4.10. 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:
student@linux:~$ set -x
student@linux:~$ 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"
```

17. 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).

18. Use one `echo` command to display three words on three lines.

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

5. shell history

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

The shell makes it easy for us to repeat commands, this chapter explains how.

5.1. repeating the last command

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

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

5.2. 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.

```
student@linux:~/test42$ touch file42
student@linux:~/test42$ cat file42
student@linux:~/test42$ !to
touch file42
student@linux:~/test42$
```

5.3. history

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

```
student@linux:~/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
```

5.4. !n

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

```
student@linux:~/test$ !43
ls
file1  file2  summer.txt  winter.txt
```

5.5. 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.

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

5.6. $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.

```
student@linux:~$ echo $HISTSIZE
500
```

You can change it to any value you like.

```
student@linux:~$ HISTSIZE=15000
student@linux:~$ echo $HISTSIZE
15000
```

5.7. $HISTFILE

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

```
student@linux:~$ 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.*

5.8. $HISTFILESIZE

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

```
student@linux:~$ echo $HISTFILESIZE
15000
```

5.9. prevent recording a command

You can prevent a command from being recorded in `history` using a space prefix.

```
student@linux:~/github$ echo abc
abc
student@linux:~/github$  echo def
def
student@linux:~/github$ echo ghi
ghi
student@linux:~/github$ history 3
 9501  echo abc
 9502  echo ghi
 9503  history 3
```

5.10. (optional)regular expressions

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

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

5.11. (optional) Korn shell history

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
$
```

5.12. 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 (by press Ctrl-shift-t in gnome-terminal, or by opening an extra putty.exe in MS Windows) with the same user account. When is command history written to the history file ?

5.13. 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.

```
student@linux:~$ 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.

```
student@linux:~$ !55
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 (by press Ctrl-shift-t in gnome-terminal, or by opening an extra putty.exe in MS Windows) with the same user account. When is command history written to the history file ?

```
when you type exit
```

Part III.

variables

6. shell variables

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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

6.1. $ 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.

```
[student@linux ~]$ echo This is the $SHELL shell
This is the /bin/bash shell
[student@linux ~]$ echo This is $SHELL on computer $HOSTNAME
This is /bin/bash on computer RHELv8u3.localdomain
[student@linux ~]$ echo The userid of $USER is $UID
The userid of paul is 500
[student@linux ~]$ echo My homedir is $HOME
My homedir is /home/paul
```

6.2. case sensitive

This example shows that shell variables are case sensitive!

```
[student@linux ~]$ echo Hello $USER
Hello paul
[student@linux ~]$ echo Hello $user
Hello
```

6.3. creating variables

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

```
[student@linux gen]$ MyVar=555
[student@linux gen]$ echo $MyVar
555
[student@linux gen]$
```

6.4. quotes

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

```
[student@linux ~]$ MyVar=555
[student@linux ~]$ echo $MyVar
555
[student@linux ~]$ echo "$MyVar"
555
[student@linux ~]$ echo '$MyVar'
$MyVar
```

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

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

6.5. 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.

6.6. unset

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

```
[student@linux ~]$ MyVar=8472
[student@linux ~]$ echo $MyVar
8472
[student@linux ~]$ unset MyVar
[student@linux ~]$ echo $MyVar

[student@linux ~]$
```

6.7. $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.

```
student@linux:~$ PS1=prompt
prompt
promptPS1='prompt '
prompt
prompt PS1='> '
>
> PS1='\u@\h$ '
student@linux$
student@linux$ PS1='\u@\h:\W$'
student@linux:~$
```

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\$ "
```

6.8. $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.

```
[[student@linux ~]$ 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.

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

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.

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

6.9. 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.

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

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).

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

6.10. export

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

```
[student@linux ~]$ var3=three
[student@linux ~]$ var4=four
[student@linux ~]$ export var4
[student@linux ~]$ echo $var3 $var4
three four
[student@linux ~]$ bash
[student@linux ~]$ echo $var3 $var4
four
```

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

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

6.11. 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.

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

6.12. 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).

```
[student@linux gen]$ echo $MyVar

[student@linux gen]$
```

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

```
student@linux:~$ set -u
student@linux:~$ echo $Myvar
bash: Myvar: unbound variable
student@linux:~$ set +u
student@linux:~$ echo $Myvar

student@linux:~$
```

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

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

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

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

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

6.14. 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
export
declare -x
```

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

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

6. Destroy your `answer` variable.

```
unset answer
```

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

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

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

```
bash
echo $var2
```

9. 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
```

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

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

Part IV.

the semicolon

7. control operators

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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(&).

7.1. ; 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.

```
[student@linux ~]$ echo Hello
Hello
[student@linux ~]$ echo World
World
[student@linux ~]$ echo Hello ; echo World
Hello
World
[student@linux ~]$
```

7.2. & 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.

```
[student@linux ~]$ sleep 20 &
[1] 7925
[student@linux ~]$
... wait 20 seconds ...
[student@linux ~]$
[1]+  Done                    sleep 20
```

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

7.3. $? 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 `$?`.

```
student@linux:~/test$ touch file1
student@linux:~/test$ echo $?
0
student@linux:~/test$ rm file1
student@linux:~/test$ echo $?
0
student@linux:~/test$ rm file1
rm: cannot remove `file1': No such file or directory
student@linux:~/test$ echo $?
1
student@linux:~/test$
```

7.4. && 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).

```
student@linux:~$ echo first && echo second
first
second
student@linux:~$ 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`.

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

7.5. || 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).

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

Another example of the same `logical OR` principle.

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

7.6. 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.

```
student@linux:~/test$ rm file1 && echo It worked! || echo It failed!
It worked!
student@linux:~/test$ rm file1 && echo It worked! || echo It failed!
rm: cannot remove `file1': No such file or directory
It failed!
student@linux:~/test$
```

7.7. # 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.

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

7.8. \ escaping special characters

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

```
[student@linux ~]$ echo hello \; world
hello ; world
[student@linux ~]$ echo hello\ \ \ world
hello   world
[student@linux ~]$ echo escaping \\\ \#\ \&\ \"\ \'
escaping \ # & " '
[student@linux ~]$ echo escaping \\\?\*\"\'
escaping \?*"'
```

7.8.1. 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.

```
[student@linux ~]$ echo This command line \
> is split in three \
> parts
This command line is split in three parts
[student@linux ~]$
```

7.9. 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)

7.10. 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/` .

```
student@linux:~$ cd ; touch test42 && echo it worked || echo it failed
it worked
student@linux:~$ 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 \ * [ } ~ \\ . "\"
```

Part V.

getting help

8. man pages

(Written by Paul Cobbaut, https://github.com/paulcobbaut/)

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.

8.1. 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).

```
student@linux:~$ man whois
Reformatting whois(1), please wait ...
```

8.2. man $configfile

Most `configuration files` have their own manual.

```
student@linux:~$ man syslog.conf
Reformatting syslog.conf(5), please wait ...
```

8.3. man $daemon

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

```
student@linux:~$ man syslogd
Reformatting syslogd(8), please wait ...
```

8.4. man -k (apropos)

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

```
student@linux:~$ 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
```

8.5. whatis

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

```
student@linux:~$ whatis route
route (8)            - show / manipulate the IP routing table
```

8.6. whereis

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

```
student@linux:~$ whereis -m whois
whois: /usr/share/man/man1/whois.1.gz
```

This file is directly readable by `man`.

```
student@linux:~$ man /usr/share/man/man1/whois.1.gz
```

8.7. 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]
```

8.8. 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.

```
[student@linux ~]$ man passwd      # opens the first manual found
[student@linux ~]$ man 5 passwd    # opens a page from section 5
```

8.9. 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...

```
student@linux:~$ man woman
No manual entry for woman
```

8.10. mandb

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

```
root@linux:~# mandb
0 man subdirectories contained newer manual pages.
0 manual pages were added.
0 stray cats were added.
0 old database entries were purged.
```

Or run `makewhatis` on CentOS/Redhat.

```
[root@linux ~]# apropos scsi
scsi: nothing appropriate
[root@linux ~]# makewhatis
[root@linux ~]# apropos scsi
hpsa                 (4)  - HP Smart Array SCSI driver
lsscsi               (8)  - list SCSI devices (or hosts) and their attributes
sd                   (4)  - Driver for SCSI Disk Drives
st                   (4)  - SCSI tape device
```

Part VI.

the file system

9. the Linux file tree

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/, Serge Van Ginderachter, https://github.com/srgvg/)

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.

```
[student@linux ~]$ 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`.

9.4.1. /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.

```
student@linux:~$ 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_mountoptions  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
...
```

9.4.2. 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.

9.4.3. /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.

```
student@linux:~$ 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
```

9.4.4. /lib

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

```
student@linux:~$ 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
```

9.4.4.1. /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.

9.4.4.2. /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.

```
student@linux:~$ 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
student@linux:~$ 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`.

9.4.5. /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

9.5.1. /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.

9.5.2. /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.

```
student@linux:~$ 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
student@linux:~$
```

There is much more to be found in /etc.

9.5.2.1. /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.

9.5.2.2. /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.

9.5.2.3. /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.

9.5.2.4. /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 RHELv8u4 with everything installed.

```
student@linux:~$ 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         i18n           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
student@linux:~$
```

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.

```
student@linux:~$ 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@linux:/etc/sysconfig# cat keyboard
KEYBOARDTYPE="pc"
KEYTABLE="us"
```

We will discuss networking files in this directory in the networking chapter.

9.6. data directories

9.6.1. /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:

```
student@linux:~$ 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.

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

9.6.2. /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.

9.6.3. /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.

9.6.4. /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`.

```
student@linux:~$ ls /media/
cdrom  cdrom0  usbdisk
```

9.6.5. /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.

9.6.6. /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

9.7.1. /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.

9.7.1.1. 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
#
student@linux:~$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3  /dev/sdb  /dev/sdb1  /dev/sdb2

#
# IDE or ATAPI
#
student@linux:~$ 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.

9.7.1.2. /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).

9.7.1.3. /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 ;-).*

9.7.2. /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.

```
student@linux:~$ 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@linux:~$ 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).

```
student@linux:~$ date
Mon Jan 29 18:06:32 EST 2007
student@linux:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 Jan 29 18:06 /proc/cpuinfo
student@linux:~$
student@linux:~$   ... time passes ...
student@linux:~$
student@linux:~$ date
Mon Jan 29 18:10:00 EST 2007
student@linux:~$ 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.

```
student@linux:~$ file /proc/cpuinfo
/proc/cpuinfo: empty
student@linux:~$ 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...

```
student@linux:~$ 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.

9.7.2.1. /proc/interrupts

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

```
student@linux:~$ 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.

```
student@linux:~$ 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
```

9.7.2.2. /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.

```
student@linux:~$ ls -lh /proc/kcore
-r-------- 1 root root 2.0G 2007-01-30 08:57 /proc/kcore
student@linux:~$
```

9.7.3. /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 (NFS is discussed later).

9.8.1. /usr/bin

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

```
student@linux:~$ ls /usr/bin | wc -l
1395
```

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

9.8.2. /usr/include

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

```
student@linux:~$ 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
 ...
```

9.8.3. /usr/lib

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

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

9.8.4. /usr/local

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

```
student@linux:~$ ls /usr/local/
bin  etc  games  include  lib  man  sbin  share  src
student@linux:~$ du -sh /usr/local/
128K    /usr/local/
```

9.8.5. /usr/share

The /usr/share directory contains architecture independent data. As you can see, this is a fairly large directory.

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

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

```
student@linux:~$ 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).

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

9.8.6. /usr/src

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

```
student@linux:~$ 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.

9.9.1. /var/log

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

```
[student@linux ~]$ 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
```

9.9.2. /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@linux ~]# 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
```

9.9.3. /var/cache

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

```
student@linux:~$ ls /var/cache/
apt      dictionaries-common    gdm       man        software-center
binfmts  flashplugin-installer  hald      pm-utils
cups     fontconfig             jockey    pppconfig
debconf  fonts                  ldconfig  samba
```

9.9.4. /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).

9.9.5. /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/`.

9.9.6. /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=zeroes.txt count=1 bs=100

od zeroes.txt
```

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

4. Now issue the following command:

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

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

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

```
ls -l /dev/sd* /dev/hd*

ls -l /dev/tty* /dev/input/mou*
```

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

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

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

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

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

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

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

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

13. Open two command prompts (Ctrl-Shift-T in gnome-terminal) or terminals (Ctrl-Alt-F1, Ctrl-Alt-F2, ...) and issue the `who am i` in both. Then try to echo a word from one terminal to the other.

14. Read the man page of `random` and explain the difference between `/dev/random` and `/dev/urandom`.

9.11. solution: file system tree

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

```
ls /bin/cat ; file /bin/cat

ls /bin/dd ; file /bin/dd

ls /bin/echo ; file /bin/echo
```

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

```
ls -lh /boot/vm*
```

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

```
cd ~/test

dd if=/dev/zero of=zeroes.txt count=1 bs=100

od zeroes.txt
```

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

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

4. Now issue the following command:

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

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

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

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

```
ls -l /dev/sd* /dev/hd*

ls -l /dev/tty* /dev/input/mou*
```

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

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

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

```
/etc/hosts/etc/hosts contains hostnames with their ip address

/etc/resolv.conf/etc/resolv.conf should contain the ip address of a DNS name server.
```

9. the Linux file tree

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". The /root directory is not accessible for normal users on most modern Linux sy
```

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 VII.

directory contents

10. working with directories

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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

This module also discusses `absolute` and `relative paths` and `path completion` in the `bash` shell.

10.1. 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 (also called a terminal, console or xterm) and type `pwd`. The tool displays your `current directory`.

```
student@linux:~$ pwd
/home/paul
```

10.2. cd

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

```
student@linux$ cd /etc
student@linux$ pwd
/etc
student@linux$ cd /bin
student@linux$ pwd
/bin
student@linux$ cd /home/paul/
student@linux$ pwd
/home/paul
```

10.2.1. cd ~

The `cd` is also a shortcut to get back into your home directory. Just typing `cd` without a target directory, will put you in your home directory. Typing `cd ~` has the same effect.

```
student@linux$ cd /etc
student@linux$ pwd
/etc
student@linux$ cd
student@linux$ pwd
/home/paul
student@linux$ cd ~
student@linux$ pwd
/home/paul
```

10.2.2. cd ..

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

```
student@linux$ pwd
/usr/share/games
student@linux$ cd ..
student@linux$ pwd
/usr/share
```

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

10.2.3. cd -

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

```
student@linux$ pwd
/home/paul
student@linux$ cd /etc
student@linux$ pwd
/etc
student@linux$ cd -
/home/paul
student@linux$ cd -
/etc
```

10.3. 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.

```
student@linux$ pwd
/home/paul
student@linux$ cd home
bash: cd: home: No such file or directory
student@linux$ cd /home
student@linux$ 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`.

```
student@linux$ pwd
/home
student@linux$ cd /paul
bash: cd: /paul: No such file or directory
student@linux$ cd paul
student@linux$ 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.

```
student@linux$ pwd
/
student@linux$ cd home
student@linux$ pwd
/home
student@linux$ cd /
student@linux$ cd /home
student@linux$ 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.

10.4. 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!

10.5. ls

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

```
student@linux:~$ ls
allfiles.txt  dmesg.txt  services   stuff  summer.txt
student@linux:~$
```

10.5.1. 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 Linux file system starts with a dot, it is considered a `hidden file` and it doesn't show up in regular file listings.

```
student@linux:~$ ls
allfiles.txt  dmesg.txt  services  stuff  summer.txt
student@linux:~$ ls -a
.   allfiles.txt   .bash_profile  dmesg.txt   .lesshst  stuff
..  .bash_history  .bashrc        services    .ssh      summer.txt
student@linux:~$
```

10.5.2. 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.

```
student@linux:~$ ls -l
total 17296
-rw-r--r-- 1 paul paul 17584442 Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul    96650 Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul    19558 Sep 17 00:04 services
drwxr-xr-x 2 paul paul     4096 Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul        0 Sep 17 00:04 summer.txt
```

10.5.3. 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.

Note that we use the letter L as an option in this screenshot, not the number 1.

```
student@linux:~$ ls -l -h
total 17M
-rw-r--r-- 1 paul paul  17M Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K Sep 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
student@linux:~$ ls -lh
total 17M
-rw-r--r-- 1 paul paul  17M Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K Sep 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
student@linux:~$ ls -hl
total 17M
-rw-r--r-- 1 paul paul  17M Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K Sep 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
student@linux:~$ ls -h -l
total 17M
-rw-r--r-- 1 paul paul  17M Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K Sep 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
student@linux:~$
```

10.6. 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 / .

```
student@linux:~$ mkdir mydir
student@linux:~$ cd mydir
student@linux:~/mydir$ ls -al
total 8
drwxr-xr-x  2 paul paul 4096 Sep 17 00:07 .
drwxr-xr-x 48 paul paul 4096 Sep 17 00:07 ..
student@linux:~/mydir$ mkdir stuff
student@linux:~/mydir$ mkdir otherstuff
student@linux:~/mydir$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff
student@linux:~/mydir$
```

10.6.1. mkdir -p

The following command will fail, because the `parent directory` of `threedirsdeep` does not exist.

```
student@linux:~$ mkdir mydir2/mysubdir2/threedirsdeep
mkdir: cannot create directory ‘mydir2/mysubdir2/threedirsdeep’: No such fi\
le or directory
```

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

```
student@linux:~$ mkdir -p mydir2/mysubdir2/threedirsdeep
student@linux:~$ cd mydir2
student@linux:~/mydir2$ ls -l
total 4
drwxr-xr-x 3 paul paul 4096 Sep 17 00:11 mysubdir2
student@linux:~/mydir2$ cd mysubdir2
student@linux:~/mydir2/mysubdir2$ ls -l
total 4
drwxr-xr-x 2 paul paul 4096 Sep 17 00:11 threedirsdeep
student@linux:~/mydir2/mysubdir2$ cd threedirsdeep/
student@linux:~/mydir2/mysubdir2/threedirsdeep$ pwd
/home/paul/mydir2/mysubdir2/threedirsdeep
```

10.7. rmdir

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

```
student@linux:~/mydir$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff
student@linux:~/mydir$ rmdir otherstuff
```

```
student@linux:~/mydir$ cd ..
student@linux:~$ rmdir mydir
rmdir: failed to remove ‘mydir’: Directory not empty
student@linux:~$ rmdir mydir/stuff
student@linux:~$ rmdir mydir
student@linux:~$
```

10.7.1. rmdir -p

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

```
student@linux:~$ mkdir -p test42/subdir
student@linux:~$ rmdir -p test42/subdir
student@linux:~$
```

10.8. 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.

10.9. 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.

10. working with directories

```
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            # opens the manual
/pushd              # searches for pushd
n                   # next (do this two/three times)
```

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.

```
student@linux:/etc$ cd /bin
student@linux:/bin$ pushd /lib
/lib /bin
student@linux:/lib$ pushd /proc
/proc /lib /bin
student@linux:/proc$ popd
/lib /bin
student@linux:/lib$ popd
/bin
```

Part VIII.

globbing

11. file globbing

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

Typing `man 7 glob` (on Debian) will tell you that long ago there was a program called `/etc/glob` that would expand wildcard patterns.

Today the shell is responsible for `file globbing` (or dynamic filename generation). This chapter will explain `file globbing`.

11.1. * 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.)

```
[student@linux gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab  FileAB  fileabc
[student@linux gen]$ ls File*
File4  File55  FileA  Fileab  FileAB
[student@linux gen]$ ls file*
file1  file2  file3  fileab  fileabc
[student@linux gen]$ ls *ile55
File55
[student@linux gen]$ ls F*ile55
File55
[student@linux gen]$ ls F*55
File55
[student@linux gen]$
```

11.2. ? 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.

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

11.3. [] 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.

```
[student@linux gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab  FileAB  fileabc
[student@linux gen]$ ls File[5A]
FileA
[student@linux gen]$ ls File[A5]
FileA
[student@linux gen]$ ls File[A5][5b]
File55
[student@linux gen]$ ls File[a5][5b]
File55  Fileab
[student@linux gen]$ ls File[a5][5b][abcdefghijklm]
ls: File[a5][5b][abcdefghijklm]: No such file or directory
[student@linux gen]$ ls file[a5][5b][abcdefghijklm]
fileabc
[student@linux 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`.

```
[student@linux gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab  FileAB  fileabc
[student@linux gen]$ ls file[a5][!Z]
fileab
[student@linux gen]$ ls file[!5]*
file1  file2  file3  fileab  fileabc
[student@linux gen]$ ls file[!5]?
fileab
[student@linux gen]$
```

11.4. a-z and 0-9 ranges

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

```
[student@linux gen]$ ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[student@linux gen]$ ls file[a-z]*
fileab  fileab2  fileabc
[student@linux gen]$ ls file[0-9]
file1  file2  file3
[student@linux gen]$ ls file[a-z][a-z][0-9]*
fileab2
[student@linux gen]$
```

11.5. $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).

```
student@linux:~/test$ ls [A-Z]ile?
file1  file2  file3  File4
student@linux:~/test$ ls [a-z]ile?
file1  file2  file3  File4
student@linux:~/test$ echo $LANG
en_US.UTF-8
student@linux:~/test$ LANG=C
student@linux:~/test$ echo $LANG
C
student@linux:~/test$ ls [a-z]ile?
file1  file2  file3
student@linux:~/test$ ls [A-Z]ile?
File4
student@linux:~/test$
```

If `$LC_ALL` is set, then this will also need to be reset to prevent file globbing.

11.6. 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.

```
student@linux:~$ mkdir test42
student@linux:~$ cd test42
student@linux:~/test42$ echo *
*
student@linux:~/test42$ touch file42 file33
student@linux:~/test42$ echo *
file33 file42
```

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

```
student@linux:~/test42$ echo *
file33 file42
student@linux:~/test42$ echo \*
*
student@linux:~/test42$ echo '*'
*
student@linux:~/test42$ echo "*"
*
```

11.7. practice: shell globbing

1. Create a test directory and enter it.

2. Create the following 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 ?

11.8. solution: shell globbing

1. Create a test directory and enter it.

```
mkdir testdir; cd testdir
```

2. Create the following 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. file globbing

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 IX.

file and directory management

12. working with files

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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

12.1. all files are case sensitive

Files on Linux (or any Unix) are `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`.

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

12.2. everything is a file

A `directory` is a special kind of `file`, but it is still a (case sensitive!) `file`. Each terminal window (for example `/dev/pts/4`), any hard disk or partition (for example `/dev/sdb1`) and any process are all represented somewhere in the `file system` as a `file`. It will become clear throughout this course that everything on Linux is a `file`.

12.3. file

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

```
student@linux:~$ file pic33.png
pic33.png: PNG image data, 3840 x 1200, 8-bit/color RGBA, non-interlaced
student@linux:~$ file /etc/passwd
/etc/passwd: ASCII text
student@linux:~$ 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@linux~# file /dev/sda
/dev/sda: block special
root@linux~# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0×83, active, starthead ...
root@linux~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@linux~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

12.4. touch

12.4.1. create an empty file

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

This screenshot starts with an empty directory, creates two files with `touch` and the lists those files.

```
student@linux:~$ ls -l
total 0
student@linux:~$ touch file42
student@linux:~$ touch file33
student@linux:~$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33
-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42
student@linux:~$
```

12.4.2. touch -t

The `touch` command can set some properties while creating empty files. Can you determine what is set by looking at the next screenshot? If not, check the manual for `touch`.

```
student@linux:~$ touch -t 200505050000 SinkoDeMayo
student@linux:~$ touch -t 130207111630 BigBattle.txt
student@linux:~$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 Jul 11  1302 BigBattle.txt
-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33
-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42
-rw-r--r-- 1 paul paul 0 May  5  2005 SinkoDeMayo
student@linux:~$
```

12.5. rm

12.5.1. remove forever

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!

```
student@linux:~$ ls
BigBattle.txt  file33  file42  SinkoDeMayo
student@linux:~$ rm BigBattle.txt
student@linux:~$ ls
file33  file42  SinkoDeMayo
student@linux:~$
```

12.5.2. rm -i

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

```
student@linux:~$ ls
file33  file42  SinkoDeMayo
student@linux:~$ rm -i file33
rm: remove regular empty file `file33'? yes
student@linux:~$ rm -i SinkoDeMayo
rm: remove regular empty file `SinkoDeMayo'? n
student@linux:~$ ls
file42  SinkoDeMayo
student@linux:~$
```

12.5.3. 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.

```
student@linux:~$ mkdir test
student@linux:~$ rm test
rm: cannot remove `test': Is a directory
student@linux:~$ rm -rf test
student@linux:~$ ls test
ls: cannot access test: No such file or directory
student@linux:~$
```

12.6. cp

12.6.1. copy one file

To copy a file, use `cp` with a source and a target argument.

```
student@linux:~$ ls
file42  SinkoDeMayo
student@linux:~$ cp file42 file42.copy
student@linux:~$ ls
file42  file42.copy  SinkoDeMayo
```

12.6.2. copy to another directory

If the target is a directory, then the source files are copied to that target directory.

```
student@linux:~$ mkdir dir42
student@linux:~$ cp SinkoDeMayo dir42
student@linux:~$ ls dir42/
SinkoDeMayo
```

12.6.3. cp -r

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

```
student@linux:~$ ls
dir42  file42  file42.copy  SinkoDeMayo
student@linux:~$ cp -r dir42/ dir33
student@linux:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
student@linux:~$ ls dir33/
SinkoDeMayo
```

12.6.4. copy 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.

```
student@linux:~$ cp file42 file42.copy SinkoDeMayo dir42/
student@linux:~$ ls dir42/
file42  file42.copy  SinkoDeMayo
```

12.6.5. cp -i

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

```
student@linux:~$ cp SinkoDeMayo file42
student@linux:~$ cp SinkoDeMayo file42
student@linux:~$ cp -i SinkoDeMayo file42
cp: overwrite `file42'? n
student@linux:~$
```

12.7. mv

12.7.1. rename files with mv

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

```
student@linux:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
student@linux:~$ mv file42 file33
student@linux:~$ ls
dir33  dir42  file33  file42.copy  SinkoDeMayo
student@linux:~$
```

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

12.7.2. rename directories with mv

The same `mv` command can be used to rename directories.

```
student@linux:~$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir33
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 Oct 15 09:38 file33
-rw-r--r-- 1 paul paul    0 Oct 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 May  5  2005 SinkoDeMayo
student@linux:~$ mv dir33 backup
student@linux:~$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 backup
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 Oct 15 09:38 file33
-rw-r--r-- 1 paul paul    0 Oct 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 May  5  2005 SinkoDeMayo
student@linux:~$
```

12.7.3. mv -i

The `mv` also has a `-i` switch similar to `cp` and `rm`.

this screenshot shows that `mv -i` will ask permission to overwrite an existing file.

```
student@linux:~$ mv -i file33 SinkoDeMayo
mv: overwrite `SinkoDeMayo'? no
student@linux:~$
```

12.8. rename

12.8.1. about rename

The `rename` command is one of the rare occasions where the Linux Fundamentals book has to make a distinction between Linux distributions. Almost every command in the `Fundamentals` part of this book works on almost every Linux computer. But `rename` is different.

Try to use `mv` whenever you need to rename only a couple of files.

12.8.2. rename on Debian/Ubuntu

The `rename` command on Debian uses regular expressions (regular expression or shor regex are explained in a later chapter) to rename many files at once.

Below a `rename` example that switches all occurrences of txt to png for all file names ending in .txt.

```
student@linux:~/test42$ ls
abc.txt  file33.txt  file42.txt
student@linux:~/test42$ rename 's/\.txt/\.png/' *.txt
student@linux:~/test42$ ls
abc.png  file33.png  file42.png
```

This second example switches all (first) occurrences of `file` into `document` for all file names ending in .png.

```
student@linux:~/test42$ ls
abc.png  file33.png  file42.png
student@linux:~/test42$ rename 's/file/document/' *.png
student@linux:~/test42$ ls
abc.png  document33.png  document42.png
student@linux:~/test42$
```

12.8.3. rename on CentOS/RHEL/Fedora

On Red Hat Enterprise Linux, the syntax of `rename` is a bit different. The first example below renames all *.conf files replacing any occurrence of .conf with .backup.

```
[student@linux ~]$ touch one.conf two.conf three.conf
[student@linux ~]$ rename .conf .backup *.conf
[student@linux ~]$ ls
one.backup  three.backup  two.backup
[student@linux ~]$
```

The second example renames all (*) files replacing one with ONE.

```
[student@linux ~]$ ls
one.backup  three.backup  two.backup
[student@linux ~]$ rename one ONE *
[student@linux ~]$ ls
ONE.backup  three.backup  two.backup
[student@linux ~]$
```

12.9. 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)

```
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

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!)

12.10. 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
```

13. basic Unix tools

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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`.

13.1. 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 {} \;
```

13.2. 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.

```
[student@linux ~]$ 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.
[student@linux ~]$ updatedb
fatal error: updatedb: You are not authorized to create a default sloc ...
[student@linux ~]$ su -
Password:
[root@linux ~]# updatedb
[root@linux ~]#
```

Most Linux distributions will schedule the `updatedb` to run once every day.

13.3. date

The `date` command can display the date, time, time zone and more.

```
student@linux ~$ 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.

```
student@linux ~$ 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.

```
student@linux ~$ date +%s
1271501080
```

When will this seconds counter reach two thousand million ?

```
student@linux ~$ date -d '1970-01-01 + 2000000000 seconds'
Wed May 18 04:33:20 CEST 2033
```

13.4. cal

The `cal` command displays the current month, with the current day highlighted.

```
student@linux ~$ 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.

```
student@linux ~$ 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
```

13.5. sleep

The `sleep` command is sometimes used in scripts to wait a number of seconds. This example shows a five second `sleep`.

```
student@linux ~$ sleep 5
student@linux ~$
```

13.6. time

The `time` command can display how long it takes to execute a command. The `date` command takes only a little time.

```
student@linux ~$ 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`.

```
student@linux ~$ 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`.

```
student@linux ~$ time bzip2 text.txt

real    0m2.368s
user    0m0.847s
sys     0m0.539s
```

13.7. gzip - gunzip

Users never have enough disk space, so compression comes in handy. The `gzip` command can make files take up less space.

```
student@linux ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
student@linux ~$ gzip text.txt
student@linux ~$ 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`.

```
student@linux ~$ gunzip text.txt.gz
student@linux ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
```

13.8. zcat - zmore

Text files that are compressed with `gzip` can be viewed with `zcat` and `zmore`.

```
student@linux ~$ head -4 text.txt
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
student@linux ~$ gzip text.txt
student@linux ~$ zcat text.txt.gz | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

13.9. bzip2 - bunzip2

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

```
student@linux ~$ bzip2 text.txt
student@linux ~$ 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`.

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

13.10. bzcat - bzmore

And in the same way `bzcat` and `bzmore` can display files compressed with `bzip2`.

```
student@linux ~$ bzip2 text.txt
student@linux ~$ bzcat text.txt.bz2 | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

13.11. 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. Here are 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 ?

13.12. 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. Here are 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.

```
student@linux ~$ find . -name '*.htm'
./one.htm
./two.htm
student@linux ~$ find . -name '*.htm' -exec mv {} {}l \;
student@linux ~$ 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

Part X.

links

14. file links

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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.

14.1. inodes

14.1.1. inode contents

An `inode` is a data structure that contains metadata about a file. When the file system stores a new file on the hard disk, it stores not only the contents (data) of the file, but also extra properties like the name of the file, the creation date, its permissions, the owner of the file, and more. All this information (except the name of the file and the contents of the file) is stored in the `inode` of the file.

The `ls -l` command will display some of the inode contents, as seen in this screenshot.

```
root@linux ~# ls -ld /home/project42/
drwxr-xr-x 4 root pro42 4.0K Mar 27 14:29 /home/project42/
```

14.1.2. 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@linux ~# df -i
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/mapper/VolGroup00-LogVol00
                     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.

14.1.3. inode number

Each `inode` has a unique number (the inode number). You can see the `inode` numbers with the `ls -li` command.

```
student@linux:~/test$ touch file1
student@linux:~/test$ touch file2
student@linux:~/test$ touch file3
student@linux:~/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
student@linux:~/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).

14.1.4. inode and file contents

Let's put some data in one of the files.

```
student@linux:~/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
student@linux:~/test$ cat file2
It is winter now and it is very cold.
We do not like the cold, we prefer hot summer nights.
student@linux:~/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.

14.2. about directories

14.2.1. 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.

```
student@linux:~/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
student@linux:~/test$
```

14.2.2. . 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.

14.3. hard links

14.3.1. 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.

```
student@linux:~/test$ ln file2 hardlink_to_file2
student@linux:~/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
student@linux:~/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.

14.3.2. 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.

```
student@linux:~/test$ find / -inum 817270 2> /dev/null
/home/paul/test/file2
/home/paul/test/hardlink_to_file2
```

14.4. 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.

```
student@linux:~/test$ ln -s file2 symlink_to_file2
student@linux:~/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
student@linux:~/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).

14.5. removing links

Links can be removed with `rm`.

```
student@linux:~$ touch data.txt
student@linux:~$ ln -s data.txt sl_data.txt
student@linux:~$ ln data.txt hl_data.txt
student@linux:~$ rm sl_data.txt
student@linux:~$ rm hl_data.txt
```

14.6. 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/rc2.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 have more than one hard link (hint: this is identical to all regular files that do not have exactly one hard link).

14.7. 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 #replace xyz with the inode number
```

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/rc2.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 have more than one hard link (hint: this is identical to all regular files that do not have exactly one hard link).

```
find ~ ! -links 1 -type f
```

Part XI.

working with text

15. working with file contents

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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.

15.1. head

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

```
student@linux~$ 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
root@linux~#
```

The `head` command can also display the first `n` lines of a file.

```
student@linux~$ 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
student@linux~$
```

And `head` can also display the first `n bytes`.

```
student@linux~$ head -c14 /etc/passwd
root:x:0:0:roostudent@linux~$
```

15.2. tail

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

```
student@linux~$ 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
student@linux~$
```

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

```
student@linux~$ tail -3 /etc/services
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
student@linux~$
```

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

15.3. cat

The `cat` command is one of the most universal tools, yet 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.

```
student@linux:~$ cat /etc/resolv.conf
domain linux-training.be
search linux-training.be
nameserver 192.168.1.42
```

15.3.1. concatenate

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

```
student@linux:~$ echo one >part1
student@linux:~$ echo two >part2
student@linux:~$ echo three >part3
student@linux:~$ cat part1
one
student@linux:~$ cat part2
two
student@linux:~$ cat part3
```

```
three
student@linux:~$ cat part1 part2 part3
one
two
three
student@linux:~$ cat part1 part2 part3 >all
student@linux:~$ cat all
one
two
three
student@linux:~$
```

15.3.2. 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.

```
student@linux:~$ cat > winter.txt
It is very cold today!
student@linux:~$ cat winter.txt
It is very cold today!
student@linux:~$
```

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

15.3.3. 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.

```
student@linux:~$ cat > hot.txt <<stop
> It is hot today!
> Yes it is summer.
> stop
student@linux:~$ cat hot.txt
It is hot today!
Yes it is summer.
student@linux:~$
```

15.3.4. 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.

```
student@linux:~$ cat winter.txt
It is very cold today!
student@linux:~$ cat winter.txt > cold.txt
student@linux:~$ cat cold.txt
It is very cold today!
student@linux:~$
```

15.4. tac

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

```
student@linux:~$ cat count
one
two
three
four
student@linux:~$ tac count
four
three
two
one
```

15.5. 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`.

15.6. 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).

```
student@linux:~$ which ls
/bin/ls
student@linux:~$ strings /bin/ls
/lib/ld-linux.so.2
librt.so.1
__gmon_start__
_Jv_RegisterClasses
clock_gettime
libacl.so.1
...
```

15.7. 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 `/etc/services`.

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`.

15.8. 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 `/etc/services`.

15. working with file contents

```
more /etc/services
```

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
```

16. I/O redirection

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

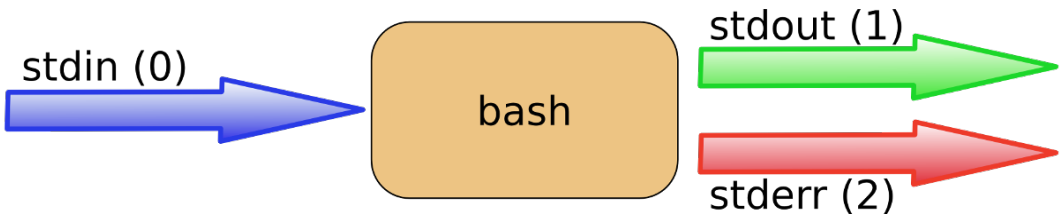
One of the powers of the Unix command line is the use of `input/output redirection` and `pipes`.

This chapter explains `redirection` of input, output and error streams.

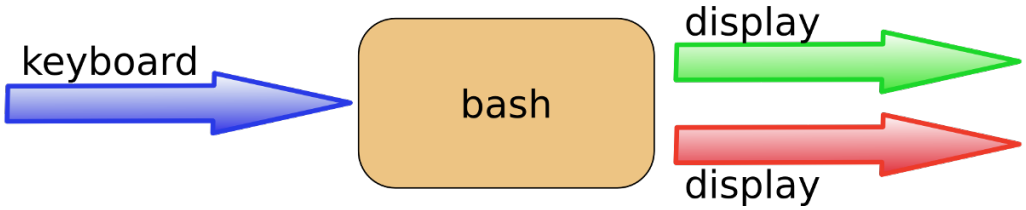
16.1. stdin, stdout, and stderr

The bash shell has three basic streams; it takes input from `stdin` (stream `0`), it sends output to `stdout` (stream 1) and it sends error messages to `stderr` (stream 2) .

The drawing below has a graphical interpretation of these three streams.



The keyboard often serves as `stdin`, whereas `stdout` and `stderr` both go to the display. This can be confusing to new Linux users because there is no obvious way to recognize `stdout` from `stderr`. Experienced users know that separating output from errors can be very useful.

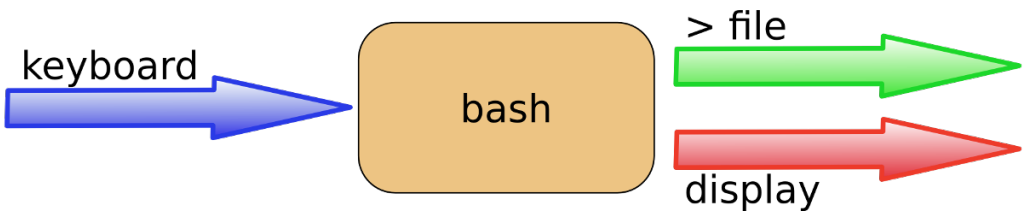


The next sections will explain how to redirect these streams.

16.2. output redirection

16.2.1. > 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.



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

```
[student@linux ~]$ echo It is cold today!
It is cold today!
[student@linux ~]$ echo It is cold today! > winter.txt
[student@linux ~]$ cat winter.txt
It is cold today!
[student@linux ~]$
```

Note that the bash shell effectively `removes` the redirection from the command line before argument 0 is executed. This means that in the case of this command:

```
echo hello > greetings.txt
```

the shell only counts two arguments (echo = argument 0, hello = argument 1). The redirection is removed before the argument counting takes place.

16.2.2. output file is erased

While scanning the line, the shell will see the > sign and `will clear the file`! Since this happens before resolving `argument 0`, this means that even when the command fails, the file will have been cleared!

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

16.2.3. noclobber

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

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

16.2.4. overruling noclobber

The `noclobber` can be overruled with `>|`.

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

16.2.5. » append

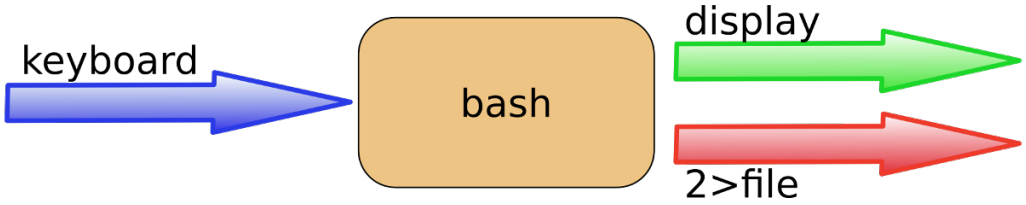
Use `>>` to `append` output to a file.

```
[student@linux ~]$ echo It is cold today! > winter.txt
[student@linux ~]$ cat winter.txt
It is cold today!
[student@linux ~]$ echo Where is the summer ? >> winter.txt
[student@linux ~]$ cat winter.txt
It is cold today!
Where is the summer ?
[student@linux ~]$
```

16.3. error redirection

16.3.1. 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 `>`.

```
[student@linux ~]$ find / > allfiles.txt 2> /dev/null
[student@linux ~]$
```

16.3.2. 2>&1

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

```
[student@linux ~]$ find / > allfiles_and_errors.txt 2>&1
[student@linux ~]$
```

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.

16.4. output redirection and pipes

By default you cannot grep inside `stderr` when using pipes on the command line, because only `stdout` is passed.

```
student@linux:~$ rm file42 file33 file1201 | grep file42
rm: cannot remove ‘file42’: No such file or directory
rm: cannot remove ‘file33’: No such file or directory
rm: cannot remove ‘file1201’: No such file or directory
```

With `2>&1` you can force `stderr` to go to `stdout`. This enables the next command in the pipe to act on both streams.

```
student@linux:~$ rm file42 file33 file1201 2>&1 | grep file42
rm: cannot remove ‘file42’: No such file or directory
```

You cannot use both `1>&2` and `2>&1` to switch `stdout` and `stderr`.

```
student@linux:~$ rm file42 file33 file1201 2>&1 1>&2 | grep file42
rm: cannot remove ‘file42’: No such file or directory
student@linux:~$ echo file42 2>&1 1>&2 | sed 's/file42/FILE42/'
FILE42
```

You need a third stream to switch stdout and stderr after a pipe symbol.

```
student@linux:~$ echo file42 3>&1 1>&2 2>&3 | sed 's/file42/FILE42/'
file42
student@linux:~$ rm file42 3>&1 1>&2 2>&3 | sed 's/file42/FILE42/'
rm: cannot remove ‘FILE42’: No such file or directory
```

16.5. joining stdout and stderr

The `&>` construction will put both `stdout` and `stderr` in one stream (to a file).

```
student@linux:~$ rm file42 &> out_and_err
student@linux:~$ cat out_and_err
rm: cannot remove ‘file42’: No such file or directory
student@linux:~$ echo file42 &> out_and_err
student@linux:~$ cat out_and_err
file42
student@linux:~$
```

16.6. input redirection

16.6.1. < stdin

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

```
[student@linux ~]$ cat < text.txt
one
two
[student@linux ~]$ tr 'onetw' 'ONEZZ' < text.txt
ONE
ZZO
[student@linux ~]$
```

16.6.2. « 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.

```
[student@linux ~]$ cat <<EOF > text.txt
> one
> two
> EOF
[student@linux ~]$ cat text.txt
one
two
[student@linux ~]$ cat <<brol > text.txt
> brel
> brol
[student@linux ~]$ cat text.txt
brel
[student@linux ~]$
```

16.6.3. «< 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).

```
student@linux~$ base64 <<< linux-training.be
bGludXgtdHJhaW5pbmcuYmUK
student@linux~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK
linux-training.be
```

See rfc 3548 for more information about `base64`.

16.7. 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
```

16.8. 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
```

16.9. practice: input/output redirection

1. Activate the `noclobber` shell option.

2. Verify that `noclobber` is active by repeating an `ls` on `/etc/` with redirected output to a file.

3. When listing all shell options, which character represents the `noclobber` option ?

4. Deactivate the `noclobber` option.

5. 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.

6. 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.

16.10. solution: input/output redirection

1. Activate the `noclobber` shell option.

```
set -o noclobber
set -C
```

2. Verify that `noclobber` is active by repeating an `ls` on `/etc/` with redirected output to a file.

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

3. When listing all shell options, which character represents the `noclobber` option ?

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

4. Deactivate the `noclobber` option.

```
set +o noclobber
```

5. 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.

```
student@linux:~$ > tailing.txt
student@linux:~$ tail -f tailing.txt
hello
world

in the other shell:
student@linux:~$ echo hello >> tailing.txt
student@linux:~$ echo world >> tailing.txt
```

6. 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.

```
student@linux:~$ cat > tennis.txt << ace
> Justine Henin
> Venus Williams
> Serena Williams
> Martina Hingis
> Kim Clijsters
> ace
student@linux:~$ cat tennis.txt
Justine Henin
Venus Williams
Serena Williams
Martina Hingis
Kim Clijsters
student@linux:~$
```

17. regular expressions

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

`Regular expressions` are a very powerful tool in Linux. They can be used with a variety of programs like bash, vi, rename, grep, sed, and more.

This chapter introduces you to the basics of `regular expressions`.

17.1. regex versions

There are three different versions of regular expression syntax:

```
BRE: Basic Regular Expressions
ERE: Extended Regular Expressions
PRCE: Perl Regular Expressions
```

Depending on the tool being used, one or more of these syntaxes can be used.

For example the `grep` tool has the `-E` option to force a string to be read as ERE while `-G` forces BRE and `-P` forces PRCE.

Note that `grep` also has `-F` to force the string to be read literally.

The `sed` tool also has options to choose a regex syntax.

`Read the manual of the tools you use!`

17.2. grep

17.2.1. print lines matching a pattern

`grep` is a popular Linux tool to search for lines that match a certain pattern. Below are some examples of the simplest `regular expressions`.

This is the contents of the test file. This file contains three lines (or three `newline` characters).

```
student@linux:~$ cat names
Tania
Laura
Valentina
```

When `grepping` for a single character, only the lines containing that character are returned.

```
student@linux:~$ grep u names
Laura
student@linux:~$ grep e names
Valentina
student@linux:~$ grep i names
Tania
Valentina
```

The pattern matching in this example should be very straightforward; if the given character occurs on a line, then `grep` will return that line.

17.2.2. concatenating characters

Two concatenated characters will have to be concatenated in the same way to have a match.

This example demonstrates that `ia` will match Tan`ia` but not Valent`ina` and `in` will match Valent`in`a but not Tan`ia`.

```
student@linux:~$ grep a names
Tania
Laura
Valentina
student@linux:~$ grep ia names
Tania
student@linux:~$ grep in names
Valentina
student@linux:~$
```

17.2.3. one or the other

PRCE and ERE both use the pipe symbol to signify OR. In this example we `grep` for lines containing the letter i or the letter a.

```
student@linux:~$ cat list
Tania
Laura
student@linux:~$ grep -E 'i|a' list
Tania
Laura
```

Note that we use the `-E` switch of grep to force interpretion of our string as an ERE.

We need to `escape` the pipe symbol in a BRE to get the same logical OR.

```
student@linux:~$ grep -G 'i|a' list
student@linux:~$ grep -G 'i\|a' list
Tania
Laura
```

17.2.4. one or more

The * signifies zero, one or more occurences of the previous and the + signifies one or more of the previous.

```
student@linux:~$ cat list2
ll
lol
lool
loool
student@linux:~$ grep -E 'o*' list2
ll
lol
lool
loool
student@linux:~$ grep -E 'o+' list2
lol
lool
loool
student@linux:~$
```

17.2.5. match the end of a string

For the following examples, we will use this file.

```
student@linux:~$ cat names
Tania
Laura
Valentina
Fleur
Floor
```

The two examples below show how to use the `dollar character` to match the end of a string.

```
student@linux:~$ grep a$ names
Tania
Laura
Valentina
student@linux:~$ grep r$ names
Fleur
Floor
```

17.2.6. match the start of a string

The `caret character (^)` will match a string at the start (or the beginning) of a line.

Given the same file as above, here are two examples.

```
student@linux:~$ grep ^Val names
Valentina
student@linux:~$ grep ^F names
Fleur
Floor
```

Both the dollar sign and the little hat are called `anchors` in a regex.

17.2.7. separating words

Regular expressions use a `\b` sequence to reference a word separator. Take for example this file:

```
student@linux:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
```

Simply grepping for `over` will give too many results.

```
student@linux:~$ grep over text
The governer is governing.
The winter is over.
Can you get over there?
```

Surrounding the searched word with spaces is not a good solution (because other characters can be word separators). This screenshot below show how to use `\b` to find only the searched word:

```
student@linux:~$ grep '\bover\b' text
The winter is over.
Can you get over there?
student@linux:~$
```

Note that `grep` also has a `-w` option to grep for words.

```
student@linux:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
student@linux:~$ grep -w over text
The winter is over.
Can you get over there?
student@linux:~$
```

17.2.8. grep features

Sometimes it is easier to combine a simple regex with `grep` options, than it is to write a more complex regex. These options where discussed before:

```
grep -i
grep -v
grep -w
grep -A5
grep -B5
grep -C5
```

17.2.9. preventing shell expansion of a regex

The dollar sign is a special character, both for the regex and also for the shell (remember variables and embedded shells). Therefore it is advised to always quote the regex, this prevents shell expansion.

```
student@linux:~$ grep 'r$' names
Fleur
Floor
```

17.3. rename

17.3.1. the rename command

On Debian Linux the `/usr/bin/rename` command is a link to `/usr/bin/prename` installed by the `perl` package.

```
student@linux ~ $ dpkg -S $(readlink -f $(which rename))
perl: /usr/bin/prename
```

Red Hat derived systems do not install the same `rename` command, so this section does not describe `rename` on Red Hat (unless you copy the perl script manually).

`There is often confusion on the internet about the rename command because solutions that work fine in Debian (and Ubuntu, xubuntu, Mint, ...) cannot be used in Red Hat (and CentOS, Fedora, ...).`

17.3.2. perl

The `rename` command is actually a perl script that uses `perl regular expressions`. The complete manual for these can be found by typing `perldoc perlrequick` (after installing `perldoc`).

```
root@linux:~# aptitude install perl-doc
The following NEW packages will be installed:
  perl-doc
0 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,170 kB of archives. After unpacking 13.2 MB will be used.
Get: 1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main perl-do ...
Fetched 8,170 kB in 19s (412 kB/s)
Selecting previously unselected package perl-doc.
(Reading database ... 67121 files and directories currently installed.)
Unpacking perl-doc (from .../perl-doc_5.14.2-21+rpi2_all.deb) ...
Adding 'diversion of /usr/bin/perldoc to /usr/bin/perldoc.stub by perl-doc'
Processing triggers for man-db ...
Setting up perl-doc (5.14.2-21+rpi2) ...

root@linux:~# perldoc perlrequick
```

17.3.3. well known syntax

The most common use of the `rename` is to search for filenames matching a certain `string` and replacing this string with an `other string`.

This is often presented as `s/string/other string/` as seen in this example:

```
student@linux ~ $ ls
abc       allfiles.TXT  bllfiles.TXT  Scratch   tennis2.TXT
abc.conf  backup        cllfiles.TXT  temp.TXT  tennis.TXT
student@linux ~ $ rename 's/TXT/text/' *
student@linux ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch    tennis2.text
abc.conf  backup         cllfiles.text  temp.text  tennis.text
```

And here is another example that uses `rename` with the well know syntax to change the extensions of the same files once more:

```
student@linux ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch    tennis2.text
abc.conf  backup         cllfiles.text  temp.text  tennis.text
student@linux ~ $ rename 's/text/txt/' *.text
student@linux ~ $ ls
abc       allfiles.txt  bllfiles.txt  Scratch   tennis2.txt
abc.conf  backup        cllfiles.txt  temp.txt  tennis.txt
student@linux ~ $
```

These two examples appear to work because the strings we used only exist at the end of the filename. Remember that file extensions have no meaning in the bash shell.

The next example shows what can go wrong with this syntax.

```
student@linux ~ $ touch atxt.txt
student@linux ~ $ rename 's/txt/problem/' atxt.txt
student@linux ~ $ ls
abc       allfiles.txt  backup        cllfiles.txt  temp.txt     tennis.txt
abc.conf  aproblem.txt  bllfiles.txt  Scratch       tennis2.txt
student@linux ~ $
```

Only the first occurrence of the searched string is replaced.

17.3.4. a global replace

The syntax used in the previous example can be described as `s/regex/replacement/`. This is simple and straightforward, you enter a `regex` between the first two slashes and a `replacement string` between the last two.

This example expands this syntax only a little, by adding a `modifier`.

```
student@linux ~ $ rename -n 's/TXT/txt/g' aTXT.TXT
aTXT.TXT renamed as atxt.txt
student@linux ~ $
```

The syntax we use now can be described as `s/regex/replacement/g` where `s` signifies `switch` and `g` stands for `global`.

Note that this example used the `-n` switch to show what is being done (instead of actually renaming the file).

17.3.5. case insensitive replace

Another `modifier` that can be useful is `i`. this example shows how to replace a case insensitive string with another string.

```
student@linux:~/files$ ls
file1.text  file2.TEXT  file3.txt
student@linux:~/files$ rename 's/.text/.txt/i' *
student@linux:~/files$ ls
file1.txt  file2.txt  file3.txt
student@linux:~/files$
```

17.3.6. renaming extensions

Command line Linux has no knowledge of MS-DOS like extensions, but many end users and graphical application do use them.

Here is an example on how to use `rename` to only rename the file extension. It uses the dollar sign to mark the ending of the filename.

```
student@linux ~ $ ls *.txt
allfiles.txt  bllfiles.txt  cllfiles.txt  really.txt.txt  temp.txt  tennis.txt
student@linux ~ $ rename 's/.txt$/.TXT/' *.txt
student@linux ~ $ ls *.TXT
allfiles.TXT  bllfiles.TXT    cllfiles.TXT    really.txt.TXT
temp.TXT      tennis.TXT
student@linux ~ $
```

Note that the `dollar sign` in the regex means `at the end`. Without the dollar sign this command would fail on the really.txt.txt file.

17.4. sed

17.4.1. stream editor

The `stream editor` or short `sed` uses `regex` for stream editing.

In this example `sed` is used to replace a string.

```
echo Sunday | sed 's/Sun/Mon/'
Monday
```

The slashes can be replaced by a couple of other characters, which can be handy in some cases to improve readability.

```
echo Sunday | sed 's:Sun:Mon:'
Monday
echo Sunday | sed 's_Sun_Mon_'
Monday
echo Sunday | sed 's|Sun|Mon|'
Monday
```

17.4.2. interactive editor

While `sed` is meant to be used in a stream, it can also be used interactively on a file.

```
student@linux:~/files$ echo Sunday > today
student@linux:~/files$ cat today
Sunday
student@linux:~/files$ sed -i 's/Sun/Mon/' today
student@linux:~/files$ cat today
Monday
```

17.4.3. simple back referencing

The `ampersand` character can be used to reference the searched (and found) string.

In this example the `ampersand` is used to double the occurence of the found string.

```
echo Sunday | sed 's/Sun/&&/'
SunSunday
echo Sunday | sed 's/day/&&/'
Sundayday
```

17.4.4. back referencing

Parentheses (often called round brackets) are used to group sections of the regex so they can leter be referenced.

Consider this simple example:

```
student@linux:~$ echo Sunday | sed 's_\(Sun\)_\1ny_'
Sunnyday
student@linux:~$ echo Sunday | sed 's_\(Sun\)_\1ny \1_'
Sunny Sunday
```

17.4.5. a dot for any character

In a `regex` a simple dot can signify any character.

```
student@linux:~$ echo 2014-04-01 | sed 's/....-..-../YYYY-MM-DD/'
YYYY-MM-DD
student@linux:~$ echo abcd-ef-gh | sed 's/....-..-../YYYY-MM-DD/'
YYYY-MM-DD
```

17.4.6. multiple back referencing

When more than one pair of `parentheses` is used, each of them can be referenced separately by consecutive numbers.

```
student@linux:~$ echo 2014-04-01 | sed 's/\(....\)-\(..\)-\(..\)/\1+\2+\3/'
2014+04+01
student@linux:~$ echo 2014-04-01 | sed 's/\(....\)-\(..\)-\(..\)/\3:\2:\1/'
01:04:2014
```

This feature is called `grouping`.

17.4.7. white space

The \s can refer to white space such as a space or a tab.

This example looks for white spaces (\s) globally and replaces them with 1 space.

```
student@linux:~$ echo -e 'today\tis\twarm'
today   is      warm
student@linux:~$ echo -e 'today\tis\twarm' | sed 's_\s_ _g'
today is warm
```

17.4.8. optional occurrence

A question mark signifies that the previous is `optional`.

The example below searches for three consecutive letter o, but the third o is optional.

```
student@linux:~$ cat list2
ll
lol
lool
loool
student@linux:~$ grep -E 'ooo?' list2
lool
loool
student@linux:~$ cat list2 | sed 's/ooo\?/A/'
ll
lol
lAl
lAl
```

17.4.9. exactly n times

You can demand an exact number of times the oprevious has to occur.

This example wants exactly three o's.

```
student@linux:~$ cat list2
ll
lol
lool
loool
student@linux:~$ grep -E 'o{3}' list2
loool
student@linux:~$ cat list2 | sed 's/o\{3\}/A/'
ll
lol
lool
lAl
student@linux:~$
```

17.4.10. between n and m times

And here we demand exactly from minimum 2 to maximum 3 times.

```
student@linux:~$ cat list2
ll
lol
lool
loool
student@linux:~$ grep -E 'o{2,3}' list2
lool
loool
student@linux:~$ grep 'o\{2,3\}' list2
lool
loool
student@linux:~$ cat list2 | sed 's/o\{2,3\}/A/'
ll
lol
lAl
lAl
student@linux:~$
```

17.5. bash history

The `bash shell` can also interprete some regular expressions.

This example shows how to manipulate the exclamation mask history feature of the bash shell.

```
student@linux:~$ mkdir hist
student@linux:~$ cd hist/
student@linux:~/hist$ touch file1 file2 file3
student@linux:~/hist$ ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
student@linux:~/hist$ !l
ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
student@linux:~/hist$ !l:s/1/3
ls -l file3
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file3
student@linux:~/hist$
```

This also works with the history numbers in bash.

```
student@linux:~/hist$ history 6
 2089  mkdir hist
 2090  cd hist/
 2091  touch file1 file2 file3
 2092  ls -l file1
 2093  ls -l file3
 2094  history 6
student@linux:~/hist$ !2092
ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
student@linux:~/hist$ !2092:s/1/2
ls -l file2
```

```
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file2
student@linux:~/hist$
```

Part XII.

user group management

18. groups

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

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`.

18.1. groupadd

Groups can be created with the `groupadd` command. The example below shows the creation of five (empty) groups.

```
root@linux:~# groupadd tennis
root@linux:~# groupadd football
root@linux:~# groupadd snooker
root@linux:~# groupadd formula1
root@linux:~# groupadd salsa
```

18.2. group file

Users can be a member of several groups. Group membership is defined by the `/etc/group` file.

```
root@linux:~# tail -5 /etc/group
tennis:x:1006:
football:x:1007:
snooker:x:1008:
formula1:x:1009:
salsa:x:1010:
root@linux:~#
```

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.

18.3. groups

A user can type the `groups` command to see a list of groups where the user belongs to.

```
[harry@linux ~]$ groups
harry sports
[harry@linux ~]$
```

18.4. usermod

Group membership can be modified with the useradd or `usermod` command.

```
root@linux:~# usermod -a -G tennis inge
root@linux:~# usermod -a -G tennis katrien
root@linux:~# usermod -a -G salsa katrien
root@linux:~# usermod -a -G snooker sandra
root@linux:~# usermod -a -G formula1 annelies
root@linux:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
snooker:x:1008:sandra
formula1:x:1009:annelies
salsa:x:1010:katrien
root@linux:~#
```

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.

18.5. groupmod

You can change the group name with the `groupmod` command.

```
root@linux:~# groupmod -n darts snooker
root@linux:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
formula1:x:1009:annelies
salsa:x:1010:katrien
darts:x:1008:sandra
```

18.6. groupdel

You can permanently remove a group with the `groupdel` command.

```
root@linux:~# groupdel tennis
root@linux:~#
```

18.7. 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@linux ~]# gpasswd -A serena sports
[root@linux ~]# su - serena
[serena@linux ~]$ id harry
uid=516(harry) gid=520(harry) groups=520(harry)
[serena@linux ~]$ gpasswd -a harry sports
Adding user harry to group sports
[serena@linux ~]$ id harry
uid=516(harry) gid=520(harry) groups=520(harry),522(sports)
[serena@linux ~]$ tail -1 /etc/group
sports:x:522:serena,venus,harry
[serena@linux ~]$
```

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@linux ~]$ gpasswd -d serena sports
Removing user serena from group sports
[serena@linux ~]$ exit
```

Information about group administrators is kept in the `/etc/gshadow` file.

```
[root@linux ~]# tail -1 /etc/gshadow
sports:!:serena:venus,harry
[root@linux ~]#
```

To remove all group administrators from a group, use the `gpasswd` command to set an empty administrators list.

```
[root@linux ~]# gpasswd -A "" sports
```

18.8. newgrp

You can start a `child shell` with a new temporary `primary group` using the `newgrp` command.

```
root@linux:~# mkdir prigroup
root@linux:~# cd prigroup/
root@linux:~/prigroup# touch standard.txt
root@linux:~/prigroup# ls -l
total 0
-rw-r--r--. 1 root root 0 Apr 13 17:49 standard.txt
root@linux:~/prigroup# echo $SHLVL
1
root@linux:~/prigroup# newgrp tennis
root@linux:~/prigroup# echo $SHLVL
2
root@linux:~/prigroup# touch newgrp.txt
root@linux:~/prigroup# ls -l
```

```
total 0
-rw-r--r--. 1 root tennis 0 Apr 13 17:49 newgrp.txt
-rw-r--r--. 1 root root   0 Apr 13 17:49 standard.txt
root@linux:~/prigroup# exit
exit
root@linux:~/prigroup#
```

18.9. 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.

18.10. 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.

18.11. 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 XIII.

user management

19. introduction to users

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

This little chapter will teach you how to identify your user account on a Unix computer using commands like `who am i`, `id`, and more.

In a second part you will learn how to become another user with the `su` command.

And you will learn how to run a program as another user with `sudo`.

19.1. whoami

The `whoami` command tells you your username.

```
[student@linux ~]$ whoami
paul
[student@linux ~]$
```

19.2. who

The `who` command will give you information about who is logged on the system.

```
[student@linux ~]$ who
root     pts/0        2014-10-10 23:07 (10.104.33.101)
paul     pts/1        2014-10-10 23:30 (10.104.33.101)
laura    pts/2        2014-10-10 23:34 (10.104.33.96)
tania    pts/3        2014-10-10 23:39 (10.104.33.91)
[student@linux ~]$
```

19.3. who am i

With `who am i` the `who` command will display only the line pointing to your current session.

```
[student@linux ~]$ who am i
paul     pts/1        2014-10-10 23:30 (10.104.33.101)
[student@linux ~]$
```

19.4. w

The `w` command shows you who is logged on and what they are doing.

```
[student@linux ~]$ w
 23:34:07 up 31 min,  2 users,  load average: 0.00, 0.01, 0.02
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0     23:07   15.00s  0.01s  0.01s top
paul     pts/1     23:30    7.00s  0.00s  0.00s w
[student@linux ~]$
```

19.5. id

The `id` command will give you your user id, primary group id, and a list of the groups that you belong to.

```
student@linux:~$ id
uid=1000(paul) gid=1000(paul) groups=1000(paul)
```

On RHEL/CentOS you will also get `SELinux` context information with this command.

```
[root@linux ~]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r\
:unconfined_t:s0-s0:c0.c1023
```

19.6. su to another user

The `su` command allows a user to run a shell as another user.

```
laura@linux:~$ su tania
Password:
tania@linux:/home/laura$
```

19.7. su to root

Yes you can also `su` to become `root`, when you know the `root password`.

```
laura@linux:~$ su root
Password:
root@linux:/home/laura#
```

19.8. su as root

You need to know the password of the user you want to substitute to, unless your are logged in as `root`. The `root` user can become any existing user without knowing that user's password.

```
root@linux:~# id
uid=0(root) gid=0(root) groups=0(root)
root@linux:~# su - valentina
valentina@linux:~$
```

19.9. 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.

```
root@linux:~# su laura
laura@linux:/root$ exit
exit
root@linux:~# su - laura
laura@linux:~$ pwd
/home/laura
```

19.10. su -

When no username is provided to `su` or `su -`, the command will assume `root` is the target.

```
tania@linux:~$ su -
Password:
root@linux:~#
```

19.11. run a program as another user

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`.

First the command fails for `paul`.

```
student@linux:~$ /usr/sbin/useradd -m valentina
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
```

But with `sudo` it works.

```
student@linux:~$ sudo /usr/sbin/useradd -m valentina
[sudo] password for paul:
student@linux:~$
```

19.12. visudo

Check the man page of `visudo` before playing with the `/etc/sudoers` file. Editing the `sudoers` is out of scope for this fundamentals book.

```
student@linux:~$ apropos visudo
visudo               (8)  - edit the sudoers file
student@linux:~$
```

19.13. sudo su -

On some Linux systems like Ubuntu and Xubuntu, 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@linux:~# 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.

```
student@linux:~$ sudo su -
Password:
root@linux:~#
```

19.14. sudo logging

Using `sudo` without authorization will result in a severe warning:

```
student@linux:~$ sudo su -

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for paul:
paul is not in the sudoers file.  This incident will be reported.
student@linux:~$
```

The root user can see this in the `/var/log/secure` on Red Hat and in `/var/log/auth.log` on Debian).

```
root@linux:~# tail /var/log/secure | grep sudo | tr -s ' '
Apr 13 16:03:42 rhel65 sudo: paul : user NOT in sudoers ; TTY=pts/0 ; PWD=\
/home/paul ; USER=root ; COMMAND=/bin/su -
root@linux:~#
```

19.15. practice: introduction to users

1. Run a command that displays only your currently logged on user name.
2. Display a list of all logged on users.
3. Display a list of all logged on users including the command they are running at this very moment.
4. Display your user name and your unique user identification (userid).

5. Use `su` to switch to another user account (unless you are root, you will need the password of the other account). And get back to the previous account.

6. Now use `su -` to switch to another user and notice the difference.

Note that `su -` gets you into the home directory of `Tania`.

7. Try to create a new user account (when using your normal user account). this should fail. (Details on adding user accounts are explained in the next chapter.)

8. Now try the same, but with `sudo` before your command.

19.16. solution: introduction to users

1. Run a command that displays only your currently logged on user name.

```
laura@linux:~$ whoami
laura
laura@linux:~$ echo $USER
laura
```

2. Display a list of all logged on users.

```
laura@linux:~$ who
laura     pts/0        2014-10-13 07:22 (10.104.33.101)
laura@linux:~$
```

3. Display a list of all logged on users including the command they are running at this very moment.

```
laura@linux:~$ w
 07:47:02 up 16 min,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    10.104.33.101    07:30    6.00s  0.04s  0.00s w
root     pts/1    10.104.33.101    07:46    6.00s  0.01s  0.00s sleep 42
laura@linux:~$
```

4. Display your user name and your unique user identification (userid).

```
laura@linux:~$ id
uid=1005(laura) gid=1007(laura) groups=1007(laura)
laura@linux:~$
```

5. Use `su` to switch to another user account (unless you are root, you will need the password of the other account). And get back to the previous account.

```
laura@linux:~$ su tania
Password:
tania@linux:/home/laura$ id
uid=1006(tania) gid=1008(tania) groups=1008(tania)
tania@linux:/home/laura$ exit
laura@linux:~$
```

6. Now use `su -` to switch to another user and notice the difference.

```
laura@linux:~$ su - tania
Password:
tania@linux:~$ pwd
/home/tania
tania@linux:~$ logout
laura@linux:~$
```

Note that `su -` gets you into the home directory of `Tania`.

7. Try to create a new user account (when using your normal user account). this should fail. (Details on adding user accounts are explained in the next chapter.)

```
laura@linux:~$ useradd valentina
-su: useradd: command not found
laura@linux:~$ /usr/sbin/useradd valentina
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
```

It is possible that `useradd` is located in `/sbin/useradd` on your computer.

8. Now try the same, but with `sudo` before your command.

```
laura@linux:~$ sudo /usr/sbin/useradd valentina
[sudo] password for laura:
laura is not in the sudoers file.  This incident will be reported.
laura@linux:~$
```

Notice that `laura` has no permission to use the `sudo` on this system.

20. user management

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

This chapter will teach you how to use `useradd`, `usermod` and `userdel` to create, modify and remove user accounts.

You will need `root` access on a Linux computer to complete this chapter.

20.1. user management

User management on Linux can be done in three complementary 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!

20.2. /etc/passwd

The local user database on Linux (and on most Unixes) is `/etc/passwd`.

```
[root@linux ~]# 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.

More information can be found by typing `man 5 passwd`.

```
[root@linux ~]# man 5 passwd
```

20.3. 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@linux ~]# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

20.4. 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@linux ~]# useradd -m -d /home/yanina -c "yanina wickmayer" yanina
[root@linux ~]# 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.

20.5. /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
```

20.6. userdel

You can delete the user yanina with `userdel`. The -r option of userdel will also remove the home directory.

```
[root@linux ~]# userdel -r yanina
```

20.7. 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
```

20.8. 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@linux ~]# mkdir /home/laura
[root@linux ~]# chown laura:laura /home/laura
[root@linux ~]# chmod 700 /home/laura
[root@linux ~]# ls -ld /home/laura/
drwx------ 2 laura laura 4096 Jun 24 15:17 /home/laura/
```

20.9. /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@linux ~]# 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
```

20.10. deleting home directories

The -r option of `userdel` will make sure that the home directory is deleted together with the user account.

```
[root@linux ~]# ls -ld /home/wim/
drwx------ 2 wim wim 4096 Jun 24 15:19 /home/wim/
[root@linux ~]# userdel -r wim
[root@linux ~]# ls -ld /home/wim/
ls: /home/wim/: No such file or directory
```

20.11. 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@linux ~]# 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@linux ~]# usermod -s /bin/bash laura
[root@linux ~]# tail -1 /etc/passwd
laura:x:528:534:art dealer:/home/laura:/bin/bash
```

20.12. 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.

```
[laura@linux ~]$ chsh -l
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/ksh
/bin/tcsh
/bin/csh
[laura@linux ~]$
```

Note that the -l option does not exist on Debian and that the above screenshot assumes that ksh and csh shells are installed.

The screenshot below shows how laura can change her default shell (active on next login).

```
[laura@linux ~]$ chsh -s /bin/ksh
Changing shell for laura.
Password:
Shell changed.
```

20.13. practice: user management

1. Create a user account named serena, including a home directory and a description (or comment) that reads Serena Williams. Do all this in one single command.

2. Create a user named venus, including home directory, bash shell, a description that reads Venus Williams all in one single command.

3. Verify that both users have correct entries in /etc/passwd, /etc/shadow and /etc/group.

4. Verify that their home directory was created.

5. Create a user named `einstime` with `/bin/date` as his default logon shell.

6. What happens when you log on with the `einstime` user ? Can you think of a useful real world example for changing a user's login shell to an application ?

7. Create a file named `welcome.txt` and make sure every new user will see this file in their home directory.

8. Verify this setup by creating (and deleting) a test user account.

9. Change the default login shell for the `serena` user to `/bin/bash`. Verify before and after you make this change.

20.14. solution: user management

1. Create a user account named `serena`, including a home directory and a description (or comment) that reads `Serena Williams`. Do all this in one single command.

```
root@linux:~# useradd -m -c 'Serena Williams' serena
```

2. Create a user named `venus`, including home directory, bash shell, a description that reads `Venus Williams` all in one single command.

```
root@linux:~# useradd -m -c "Venus Williams" -s /bin/bash venus
```

3. Verify that both users have correct entries in `/etc/passwd`, `/etc/shadow` and `/etc/group`.

```
root@linux:~# tail -2 /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/sh
venus:x:1009:1011:Venus Williams:/home/venus:/bin/bash
root@linux:~# tail -2 /etc/shadow
serena:!:16358:0:99999:7:::
venus:!:16358:0:99999:7:::
root@linux:~# tail -2 /etc/group
serena:x:1010:
venus:x:1011:
```

4. Verify that their home directory was created.

```
root@linux:~# ls -lrt /home | tail -2
drwxr-xr-x 2 serena    serena    4096 Oct 15 10:50 serena
drwxr-xr-x 2 venus     venus     4096 Oct 15 10:59 venus
root@linux:~#
```

5. Create a user named `einstime` with `/bin/date` as his default logon shell.

```
root@linux:~# useradd -s /bin/date einstime
```

Or even better:

```
root@linux:~# useradd -s $(which date) einstime
```

6. What happens when you log on with the `einstime` user ? Can you think of a useful real world example for changing a user's login shell to an application ?

```
root@linux:~# su - einstime
Wed Oct 15 11:05:56 UTC 2014    # You get the output of the date command
root@linux:~#
```

It can be useful when users need to access only one application on the server. Just logging in opens the application for them, and closing the application automatically logs them out.

7. Create a file named `welcome.txt` and make sure every new user will see this file in their home directory.

```
root@linux:~# echo Hello > /etc/skel/welcome.txt
```

8. Verify this setup by creating (and deleting) a test user account.

```
root@linux:~# useradd -m test
root@linux:~# ls -l /home/test
total 4
-rw-r--r-- 1 test test 6 Oct 15 11:16 welcome.txt
root@linux:~# userdel -r test
root@linux:~#
```

9. Change the default login shell for the `serena` user to `/bin/bash`. Verify before and after you make this change.

```
root@linux:~# grep serena /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/sh
root@linux:~# usermod -s /bin/bash serena
root@linux:~# grep serena /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/bash
root@linux:~#
```

21. user passwords

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/)

This chapter will tell you more about passwords for local users.

Three methods for setting passwords are explained; using the `passwd` command, using `openssel passwd`, and using the `crypt` function in a C program.

The chapter will also discuss password settings and disabling, suspending or locking accounts.

21.1. 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.

```
[tania@linux ~]$ passwd
Changing password for user tania.
Changing password for tania.
(current) UNIX password:
New password:
BAD PASSWORD: The password is shorter than 8 characters
New password:
BAD PASSWORD: The password is a palindrome
New password:
BAD PASSWORD: The password is too similar to the old one
passwd: Have exhausted maximum number of retries for service
```

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.

```
root@linux:~# passwd tania
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

21.2. shadow file

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@linux ~]# tail -4 /etc/shadow
paul:$6$ikp2Xta5BT.Tml.p$2TZjNnOYNNQKpwLJqoGJbVsZG5/Fti8ovBRd.VzRbiDSl7TEq\
IaSMH.TeBKnTS/SjlMruW8qffC0JNORW.BTW1:16338:0:99999:7:::
tania:$6$8Z/zovxj$9qvoqT8i9KIrmN.k4EQwAF5ryz5yzNwEvYjAa9L5XVXQu.z4DlpvMREH\
eQpQzvRnqFdKkVj17H5ST.c79HDZw0:16356:0:99999:7:::
laura:$6$glDuTY5e$/NYYWLxfHgZFWeoujaXSMcR.Mz.lGOxtcxFocFVJNb98nbTPhWFXfKWG\
SyYh1WCv6763Wq54.w24Yr3uAZBOm/:16356:0:99999:7:::
valentina:$6$jrZa6PVI$1uQgqR6En9mZB6mKJ3LXRB4CnFko6LRhbh.v4iqUk9MVreui1lv7\
GxHOUDSKA0N55ZRNhGHa6T2ouFnVno/0o1:16356:0:99999:7:::
[root@linux ~]#
```

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.

All the passwords in the screenshot above are hashes of `hunter2`.

21.3. 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 ~]#
```

21.4. 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.

The `openssl passwd` command will generate several distinct hashes for the same password, for this it uses a `salt`.

```
student@linux:~$ openssl passwd hunter2
86jcUNlnGDFpY
student@linux:~$ openssl passwd hunter2
Yj7mDO9OAnvq6
student@linux:~$ openssl passwd hunter2
YqDcJeGoDbzKA
student@linux:~$
```

This `salt` can be chosen and is visible as the first two characters of the hash.

```
student@linux:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
student@linux:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
student@linux:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
student@linux:~$
```

This example shows how to create a user with password.

```
root@linux:~# useradd -m -p $(openssl passwd hunter2) mohamed
```

Note that this command puts the password in your command history!

21.5. encryption with crypt

A third option is to create your own C program using the crypt function, and compile this into a command.

```
student@linux:~$ 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.

```
student@linux:~$ 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`.

```
student@linux:~$ ./MyCrypt hunter2 42
42ZrbtP1Ze8G.
student@linux:~$ ./MyCrypt hunter2 33
33d6taYSiEUXI
```

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.

```
student@linux:~$ ./MyCrypt hunter2 '$1$42'
$1$42$7l6Y3xT5282XmZrtDOF9f0
student@linux:~$ ./MyCrypt hunter2 '$6$42'
$6$42$OqFFAVnI3gTSYG0yI9TZWX9cpyQzwIop7HwpG1LLEsNBiMr4w6OvLX1KDa./UpwXfrFk1i...
```

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!

```
student@linux:~$ ./MyCrypt hunter2 '$1$hunter2'
$1$hunter2$YVxrxDmidq7Xf8Gdt6qM2.
```

21.6. /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@linux:~# grep ^PASS /etc/login.defs
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7
```

Debian also has this file.

```
root@linux:~# grep PASS /etc/login.defs
#  PASS_MAX_DAYS   Maximum number of days a password may be used.
#  PASS_MIN_DAYS   Minimum number of days allowed between password changes.
#  PASS_WARN_AGE   Number of days warning given before a password expires.
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_WARN_AGE   7
#PASS_CHANGE_TRIES
#PASS_ALWAYS_WARN
#PASS_MIN_LEN
#PASS_MAX_LEN
# NO_PASSWORD_CONSOLE
root@linux:~#
```

21.7. 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@linux:~# chage -l paul
Last password change                                    : Mar 27, 2014
Password expires                                        : never
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
```

```
Maximum number of days between password change          : 99999
Number of days of warning before password expires       : 7
root@linux:~#
```

21.8. 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 command in the next screenshot will show the hashed password of `laura` in `/etc/shadow`. The next command disables the password of `laura`, making it impossible for Laura to authenticate using this password.

```
root@linux:~# grep laura /etc/shadow | cut -c1-70
laura:$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa3CoDUJV
root@linux:~# usermod -L laura
```

As you can see below, the password hash is simply preceded with an exclamation mark.

```
root@linux:~# grep laura /etc/shadow | cut -c1-70
laura:!$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa3CoDUJ
root@linux:~#
```

The root user (and users with `sudo` rights on `su`) still will be able to `su` into the `laura` account (because the password is not needed here). Also note that `laura` will still be able to login if she has set up passwordless ssh!

```
root@linux:~# su - laura
laura@linux:~$
```

You can unlock the account again with `usermod -U`.

```
root@linux:~# usermod -U laura
root@linux:~# grep laura /etc/shadow | cut -c1-70
laura:$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa3CoDUJV
```

Watch out for tiny differences in the command line options of `passwd`, `usermod`, and `useradd` on different Linux distributions. Verify the local files when using features like `"disabling, suspending, or locking"` on user accounts and their passwords.

21.9. 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@linux ~]# vipw /etc/passwd
vipw: the password file is busy (/etc/ptmp present)
```

21.10. practice: user passwords

1. Set the password for `serena` to `hunter2`.

2. Also set a password for `venus` and then lock the `venus` user account with `usermod`. Verify the locking in `/etc/shadow` before and after you lock it.

3. Use `passwd -d` to disable the `serena` password. Verify the `serena` line in `/etc/shadow` before and after disabling.

4. What is the difference between locking a user account and disabling a user account's password like we just did with `usermod -L` and `passwd -d`?

5. Try changing the password of serena to serena as serena.

6. Make sure `serena` has to change her password in 10 days.

7. Make sure every new user needs to change their password every 10 days.

8. Take a backup as root of `/etc/shadow`. Use `vi` to copy an encrypted `hunter2` hash from `venus` to `serena`. Can `serena` now log on with `hunter2` as a password ?

9. Why use `vipw` instead of `vi` ? What could be the problem when using `vi` or `vim` ?

10. Use `chsh` to list all shells (only works on RHEL/CentOS/Fedora), and compare to `cat /etc/shells`.

11. Which `useradd` option allows you to name a home directory ?

12. How can you see whether the password of user `serena` is locked or unlocked ? Give a solution with `grep` and a solution with `passwd`.

21.11. solution: user passwords

1. Set the password for `serena` to `hunter2`.

```
root@linux:~# passwd serena
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

2. Also set a password for `venus` and then lock the `venus` user account with `usermod`. Verify the locking in `/etc/shadow` before and after you lock it.

```
root@linux:~# passwd venus
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@linux:~# grep venus /etc/shadow | cut -c1-70
venus:$6$gswzXICW$uSnKFV1kFKZmTPaMVS4AvNA/KO27OxN0v5LHdV9ed0gTyXrjUeM/
root@linux:~# usermod -L venus
root@linux:~# grep venus /etc/shadow | cut -c1-70
venus:!$6$gswzXICW$uSnKFV1kFKZmTPaMVS4AvNA/KO27OxN0v5LHdV9ed0gTyXrjUeM
```

Note that `usermod -L` precedes the password hash with an exclamation mark (!).

3. Use `passwd -d` to disable the `serena` password. Verify the `serena` line in `/etc/shadow` before and after disabling.

```
root@linux:~# grep serena /etc/shadow | cut -c1-70
serena:$6$Es/omrPE$F2Ypu8kpLrfKdW0v/UIwA5jrYyBD2nwZ/dt.i/IypRgiPZSdB/B
root@linux:~# passwd -d serena
passwd: password expiry information changed.
root@linux:~# grep serena /etc/shadow
serena::16358:0:99999:7:::
root@linux:~#
```

4. What is the difference between locking a user account and disabling a user account's password like we just did with `usermod -L` and `passwd -d`?

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`.

5. Try changing the password of serena to serena as serena.

```
log on as serena, then execute: passwd serena ... it should fail!
```

6. Make sure `serena` has to change her password in 10 days.

```
chage -M 10 serena
```

7. Make sure every new user needs to change their password every 10 days.

```
vi /etc/login.defs (and change PASS_MAX_DAYS to 10)
```

8. Take a backup as root of `/etc/shadow`. Use `vi` to copy an encrypted `hunter2` hash from `venus` to `serena`. Can `serena` now log on with `hunter2` as a password ?

```
Yes.
```

9. 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 (with vipw).
```

10. Use `chsh` to list all shells (only works on RHEL/CentOS/Fedora), and compare to `cat /etc/shells`.

```
chsh -l
cat /etc/shells
```

11. Which `useradd` option allows you to name a home directory ?

```
-d
```

12. How can you see whether the password of user `serena` is locked or unlocked ? Give a solution with `grep` and a solution with `passwd`.

```
grep serena /etc/shadow
```

```
passwd -S serena
```

Part XIV.

file permissions

22. standard file permissions

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Alex M. Schapelle, https://github.com/zero-pytagoras/, Bert Van Vreckem, https://github.com/bertvv/)

This chapter contains details about basic file security through *file ownership* and *file permissions*.

22.1. file ownership

22.1.1. 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 example.

```
student@linux:~/owners$ ls -lh
total 636K
-rw-r--r--. 1 student snooker  1.1K Apr  8 18:47 data.odt
-rw-r--r--. 1 student student  626K Apr  8 18:46 file1
-rw-r--r--. 1 student tennis    185 Apr  8 18:46 file2
-rw-rw-r--. 1 root root           0 Apr  8 18:47 stuff.txt
```

User `student` owns three files: `file1` has `student` as *user owner* and has the group `student` as *group owner*, `data.odt` is *group owned* by the group `snooker`, `file2` by the group `tennis`.

The last file is called `stuff.txt` and is owned by the `root` user and the `root` group.

22.1.2. chgrp

You can change the group owner of a file using the `chgrp` command. You must have root privileges to do this.

```
root@linux:/home/student/owners# ls -l file2
-rw-r--r--. 1 root tennis 185 Apr  8 18:46 file2
root@linux:/home/student/owners# chgrp snooker file2
root@linux:/home/student/owners# ls -l file2
-rw-r--r--. 1 root snooker 185 Apr  8 18:46 file2
root@linux:/home/student/owners#
```

22.1.3. chown

The user owner of a file can be changed with `chown` command. You must have root privileges to do this. In the following example, the user owner of `file2` is changed from `root` to `student`.

```
root@linux:/home/student# ls -l FileForStudent
-rw-r--r-- 1 root student 0 2008-08-06 14:11 FileForStudent
root@linux:/home/student# chown student FileForStudent
root@linux:/home/student# ls -l FileForStudent
-rw-r--r-- 1 student student 0 2008-08-06 14:11 FileForStudent
```

You can also use `chown user:group` to change both the user owner and the group owner.

```
root@linux:/home/student# ls -l FileForStudent
-rw-r--r-- 1 student student 0 2008-08-06 14:11 FileForStudent
root@linux:/home/student# chown root:project42 FileForStudent
root@linux:/home/student# ls -l FileForStudent
-rw-r--r-- 1 root project42 0 2008-08-06 14:11 FileForStudent
```

22.2. 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`.

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

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

```
student@linux:~$ ls -l /dev/console /dev/sda
crw--w---- 1 root tty  5, 1 Mar  8 08:32 /dev/console
brw-rw---- 1 root disk 8, 0 Mar  8 08:32 /dev/sda
```

And here you can see a directory, a regular file and a symbolic link.

```
student@linux:~$ ls -ld /etc /etc/hosts /etc/os-release
drwxr-xr-x 81 root root 4096 Mar  8 08:32 /etc
-rw-r--r--  1 root root  186 Feb 26 14:58 /etc/hosts
lrwxrwxrwx  1 root root   21 Dec  9 21:08 /etc/os-release -> ../usr/lib/os-
release
```

22.3. permissions

22.3.1. rwx

The nine characters following the file type denote the permissions in three triplets. A permission can be `r` for **r**ead access, `w` for **w**rite access, and `x` for e**x**ecute. 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.

permission	on a file	on a directory
read	read file contents (`cat`)	read directory contents (`ls`)
write	change file contents	create/delete files (`touch`,`rm`)
e**x**ecute	execute the file	enter the directory (`cd`)

22.3.2. three sets of rwx

We already know that the output of `ls -l` starts with ten characters for each file. This example shows a regular file (because the first character is a -).

```
student@linux:~/test$ ls -l proc42.sh
-rwxr-xr--  1 student proj  984 Feb  6 12:01 proc42.sh
```

Below is a table describing the function of all ten characters.

position	characters	function
1	-	file type
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.

22.3.3. permission examples

Some example combinations on files and directories are seen in this example. The name of the file explains the permissions.

```
student@linux:~/perms$ ls -lh
total 12K
drwxr-xr-x 2 student student 4.0K 2007-02-07 22:26 AllEnter_UserCreateDelete
-rwxrwxrwx 1 student student    0 2007-02-07 22:21 EveryoneFullControl.txt
-r--r----- 1 student student    0 2007-02-07 22:21 OnlyOwnersRead.txt
-rwxrwx--- 1 student student    0 2007-02-07 22:21 OwnersAll_RestNothing.txt
dr-xr-x--- 2 student student 4.0K 2007-02-07 22:25 UserAndGroupEnter
dr-x------ 2 student student 4.0K 2007-02-07 22:25 OnlyUserEnter
```

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*. The `root` user ignores all restrictions and can do anything with any file.

22.3.4. setting permissions with symbolic notation

Permissions can be changed with `chmod MODE FILE ...`. You need to be the owner of the file to do this. The first example gives (`+`) the *user owner* (`u`) execute (`x`) permissions.

```
student@linux:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 student student 0 2007-02-07 22:34 permissions.txt
student@linux:~/perms$ chmod u+x permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 student student 0 2007-02-07 22:34 permissions.txt
```

This example removes (`-`) the group owner's (`g`) read (`r`) permission.

```
student@linux:~/perms$ chmod g-r permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwx---r-- 1 student student 0 2007-02-07 22:34 permissions.txt
```

This example removes (`-`) the other's (`o`) read (`r`) permission.

```
student@linux:~/perms$ chmod o-r permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwx------ 1 student student 0 2007-02-07 22:34 permissions.txt
```

This example gives (`+`) all (`a`) of them the write (`w`) permission.

```
student@linux:~/perms$ chmod a+w permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 student student 0 2007-02-07 22:34 permissions.txt
```

You don't even have to type the `a`.

```
student@linux:~/perms$ chmod +x permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwx-wx-wx 1 student student 0 2007-02-07 22:34 permissions.txt
```

You can also set explicit permissions with `=`.

```
student@linux:~/perms$ chmod u=rw permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rw--wx-wx 1 student student 0 2007-02-07 22:34 permissions.txt
```

Feel free to make any kind of combination, separating them with a comma. Remark that spaces are **not** allowed!

```
student@linux:~/perms$ chmod u=rw,g=rw,o=r permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 student student 0 2007-02-07 22:34 permissions.txt
```

Even fishy combinations are accepted by `chmod`.

```
student@linux:~/perms$ chmod u=rwx,ug+rw,o=r permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwxrw-r-- 1 student student 0 2007-02-07 22:34 permissions.txt
```

Summarized, in order to change permissions with `chmod` using symbolic notation:

- first specify who the permissions are for: `u` for the user owner, `g` for the group owner, `o` for others, and `a` for all. `a` is the default and can be omitted.
- then specify the operation: `+` to add permissions, `-` to remove permissions, and `=` to set permissions.
- finally specify the permission(s): `r` for read, `w` for write, and `x` for execute.
- multiple operations can be combined with a comma (no spaces!)

22.3.5. setting permissions with octal notation

Most Unix administrators will use the "old school" octal system to talk about and set permissions. Consider the triplet to be a binary number with 0 indicating the permission is not set and 1 indicating the permission is set. You then have $2^3 = 8$ possible combinations, hence the name *octal*. You can then convert the binary number to an octal number, equating `r` to 4, `w` to 2, and `x` to 1.

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

Since we have three triplets, we can use three octal digits to represent the permissions. This makes 777 equal to `rwxrwxrwx` and by the same logic, 654 mean `rw-r-xr--`. The `chmod` command will accept these numbers.

```
student@linux:~/perms$ chmod 777 permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 student student 0 2007-02-07 22:34 permissions.txt
student@linux:~/perms$ chmod 664 permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 student student 0 2007-02-07 22:34 permissions.txt
student@linux:~/perms$ chmod 750 permissions.txt
student@linux:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 student student 0 2007-02-07 22:34 permissions.txt
```

Remark that in practice, some combinations will never occur:

- The permissions of a user will never be smaller than the permissions of the group owner or others. Consequently, the digits will always be in descending order.
- Setting the write or execute permission without read access is useless. Consequently, you will never use 1, 2, or 3 in an octal permission code

- A directory will always have the read and execute permission set or unset together. It is useless to allow a user to read the directory contents, but not let them `cd` into that directory. Allowing `cd` without read access is also useless. The permission code for a directory will therefore always be odd.

Here's a little tip: you can print the permissions of a file in either octal or symbolic notation with the `stat` command (check the man page of `stat` to see how this works).

```
[student@linux ~]$ stat -c '%A %a' /etc/passwd
-rw-r--r-- 644
[student@linux ~]$ stat -c '%A %a' /etc/shadow
---------- 0
[student@linux ~]$ stat -c '%A %a' /bin/ls
-rwxr-xr-x 755
```

22.3.6. umask

When creating a file or directory, a set of default permissions are applied. These default permissions are determined by the `umask` value. The `umask` specifies permissions that you do not want set on by default. You can display the `umask` with the `umask` command.

```
[student@linux ~]$ umask
0002
[student@linux ~]$ touch test
[student@linux ~]$ ls -l test
-rw-rw-r--  1 student student    0 Jul 24 06:03 test
[student@linux ~]$
```

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` value of 0022 has the same effect as 0033.

In practice, you will only use umask values:

- 0: don't take away any permissions
- 2: take away write permissions
- 7: take away all permissions

You can set the `umask` value to a new value with the `umask` command. The `umask` value is a four-digit octal number. The first digit is for special permissions (and is always zero), the second for the user permissions (is in practice always 0, since there is no use in taking away the user's permissions), the third for the group owner (sometimes 0, but usually 2 or 7), and the last for others (usually 2 or 7, 0 is very uncommon and can be considered to be a security risk).

The `umask` value is subtracted from 777 to get the default permissions and in the case of a file, the execute bit is removed.

```
[student@linux ~]$ umask 0002
[student@linux ~]$ touch file0002
[student@linux ~]$ mkdir dir0002
[student@linux ~]$ ls -ld *0002
drwxrwxr-x. 2 student student 6 Mar  8 10:48 dir0002
-rw-rw-r--. 1 student student 0 Mar  8 10:47 file0002
[student@linux ~]$ umask 0027
[student@linux ~]$ touch file0027
[student@linux ~]$ mkdir dir0027
```

```
[student@linux ~]$ ls -ld *0027
drwxr-x---. 2 student student 6 Mar  8 10:48 dir0027
-rw-r-----. 1 student student 0 Mar  8 10:48 file0027
[student@linux ~]$ umask 0077
[student@linux ~]$ touch file0077
[student@linux ~]$ mkdir dir0077
[student@linux ~]$ ls -ld *0077
drwx------. 2 student student 6 Mar  8 10:51 dir0077
-rw-------. 1 student student 0 Mar  8 10:51 file0077
```

22.3.7. mkdir -m

When creating directories with `mkdir` you can use the `-m` option to set the `mode`. This example explains.

```
student@linux~$ mkdir -m 700 MyDir
student@linux~$ mkdir -m 777 Public
student@linux~$ ls -dl MyDir/ Public/
drwx------ 2 student student 4096 2011-10-16 19:16 MyDir/
drwxrwxrwx 2 student student 4096 2011-10-16 19:16 Public/
```

22.3.8. cp -p

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

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

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

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

22.4. 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. Delete the file created by root. Is this possible?

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--`?
11. Display the umask value in octal and in symbolic form.
12. Set the umask to 0077, but use the symbolic format to set it. Verify that this works.
13. Create a file as root, give only read to others. Can a normal user read this file? Test writing to this file with `vi` or `nano`.
14. Create a file as a normal user, take away all permissions for the group owner and others. Can you still read the file? Can root read the file? Can root write to the file?
15. Create a directory that belongs to group `users`, 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.

22.5. solution: standard file permissions

1. As normal user, create a directory `~/permissions`. Create a file owned by yourself in there.

```
[student@linux ~]$ mkdir permissions
[student@linux ~]$ touch permissions/myfile.txt
[student@linux ~]$ ls -l permissions/
total 0
-rw-r--r--. 1 student student 0 Mar  8 10:59 myfile.txt
```

2. Copy a file owned by root from /etc/ to your permissions dir, who owns this file now ?

```
[student@linux ~]$ ls -l /etc/hosts
-rw-r--r--. 1 root root 174 Feb 26 15:05 /etc/hosts
[student@linux ~]$ cp /etc/hosts ~/permissions/
[student@linux ~]$ ls -l permissions/hosts
-rw-r--r--. 1 student student 174 Mar  8 11:00 permissions/hosts
```

   The copy is owned by you.

3. As root, create a file in the users `~/permissions` directory.

```
[student@linux ~]$ sudo touch permissions/rootfile.txt
[sudo] password for student:
```

4. As normal user, look at who owns this file created by root.

```
[student@linux ~]$ ls -l permissions/*.txt
-rw-r--r--. 1 student student 0 Mar  8 10:59 permissions/myfile.txt
-rw-r--r--. 1 root    root    0 Mar  8 11:02 permissions/rootfile.txt
```

   The file created by root is owned by root.

5. Change the ownership of all files in ~/permissions to yourself.

```
[student@linux ~]$ chown student ~/permissions/*
chown: changing ownership of '/home/student/permissions/rootfile.txt': Operation not p
```

   You cannot become owner of the file that belongs to root. Root must change the ownership.

6. Delete the file created by root. Is this possible?

```
[student@linux ~]$ rm ~/permissions/rootfile.txt
rm: remove write-protected regular empty file '/home/student/permissions/rootfile.txt'
[student@linux ~]$ ls -l permissions/*.txt
-rw-r--r--. 1 student student 0 Mar  8 10:59 permissions/myfile.txt
```

You can delete the file since you have write permission on the directory!

7. With chmod, is 770 the same as `rwxrwx---`?

   yes

8. With chmod, is 664 the same as `r-xr-xr--`?

   no, `rw-rw-r--` is 664 and `r-xr-xr--` is 774

9. With chmod, is 400 the same as `r--------`?

   yes

10. With chmod, is 734 the same as `rwxr-xr--`?

    no, `rwxr-xr--` is 754 and `rwx-wxr--` is 734

11. Display the umask in octal and in symbolic form.

    `umask` and `umask -S`

12. Set the umask to 0077, but use the symbolic format to set it. Verify that this works.

    ```
    [student@linux ~]$ umask -S u=rwx,go=
    u=rwx,g=,o=
    [student@linux ~]$ umask
    0077
    ```

13. Create a file as root, give only read to others. Can a normal user read this file? Test writing to this file with `vi` or `nano`.

    ```
    [student@linux ~]$ sudo vi permissions/rootfile.txt
    [student@linux ~]$ sudo chmod 644 permissions/rootfile.txt
    [student@linux ~]$ ls -l permissions/*.txt
    -rw-r--r--. 1 student student 0 Mar  8 10:59 permissions/myfile.txt
    -rw-r--r--. 1 root    root    6 Mar  8 13:53 permissions/rootfile.txt
    [student@linux ~]$ cat permissions/rootfile.txt
    hello
    [student@linux ~]$ echo " world" >> permissions/rootfile.txt
    -bash: permissions/rootfile.txt: Permission denied
    ```

    Yes, a normal user can read the file, but not write to it.

14. Create a file as a normal user, take away all permissions for the group and others. Can you still read the file? Can root read the file? Can root write to the file?

    ```
    [student@linux ~]$ vi permissions/privatefile.txt
    ... (editing the file) ...
    [student@linux ~]$ cat permissions/privatefile.txt
    hello
    [student@linux ~]$ chmod 600 permissions/privatefile.txt
    [student@linux ~]$ ls -l permissions/privatefile.txt
    -rw-------. 1 student student 0 Mar  8 16:06 permissions/privatefile.txt
    [student@linux ~]$ cat permissions/privatefile.txt
    hello
    ```

    Of course, the owner can still read (and write to) the file.

```
[student@linux ~]$ sudo vi permissions/privatefile.txt
[sudo] password for student:
... (editing the file) ...
[student@linux ~]$ cat permissions/privatefile.txt
hello world
```

Root can read and write to the file. In fact, root ignores all file permissions and can do anything with any file.

15. Create a directory `shared/` that belongs to group `users`, where every member of that group can read and write to files, and create files.

```
[student@linux ~]$ mkdir shared
[student@linux ~]$ sudo chgrp users shared
[student@linux ~]$ chmod 775 shared/
[student@linux ~]$ ls -ld shared/
drwxrwxr-x. 2 student users 6 Mar  8 18:26 shared/
```

A. GNU Free Documentation License

Version 1.3, 3 November 2008

A.1. 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. Secondarily, 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.

A.2. 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.

A.3. 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.

A.4. 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.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has 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, 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.

A.6. 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".

A.7. 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.

A.8. 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.

A.9. 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.

A.10. 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.

A.11. 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.

A.12. 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.