Linux Fundamentals

Paul Cobbaut

October 28, 2024

Contents

Feel free to contact the author:

- Paul Cobbaut: paul.cobbaut@gmail.com, https://cobbaut.be/

Contributors to the Linux Training project are:

- Serge van Ginderachter: serge@ginsys.eu, build scripts and infrastructure setup
- Ywein Van den Brande: ywein@crealaw.eu, license and legal sections
- Bert Van Vreckem: https://github.com/bertvv, translation to Markdown, new build scripts, and infrastructure setup

We'd also like to thank our reviewers:

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

Abstract

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

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

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

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.
Linux Mint	You want a personal graphical desktop to play movies, music and games.

Distribution name	Reason(s) for using
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).

open source PRESS
The International Free and Open Source Software Law Book
edited by Van den Brande · Coughlan · Jaeger

Ywein Van den Brande
Het Praktijkboek Informaticarecht:
Recht rendeert voor uw onderneming
Wat u moet weten over informaticarecht gebundeld in een praktische en deskundige gids.
Crealaw
Het Praktijkboek Informaticarecht
Ywein Van den Brande

`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.

Installing Linux

4. installing Debian 8

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

This module is a step by step demonstration of an actual installation of `Debian 8` (also known as `Jessie`).

We start by downloading an image from the internet and install `Debian 8` as a virtual machine in `Virtualbox`. We will also do some basic configuration of this new machine like setting an `ip address` and fixing a `hostname`.

This procedure should be very similar for other versions of `Debian`, and also for distributions like `Linux Mint`, `xubuntu/ubuntu/kubuntu` or `Mepis`. This procedure can also be helpful if you are using another virtualization solution.

Go to the next chapter if you want to install `CentOS, Fedora, Red Hat Enterprise Linux, ...`.

4.1. Debian

Debian is one of the oldest Linux distributions. I use Debian myself on almost every computer that I own (including `raspbian` on the `Raspberry Pi`).

Debian comes in `releases` named after characters in the movie `Toy Story`. The `Jessie` release contains about 36000 packages.

Table 4.1.: Debian releases

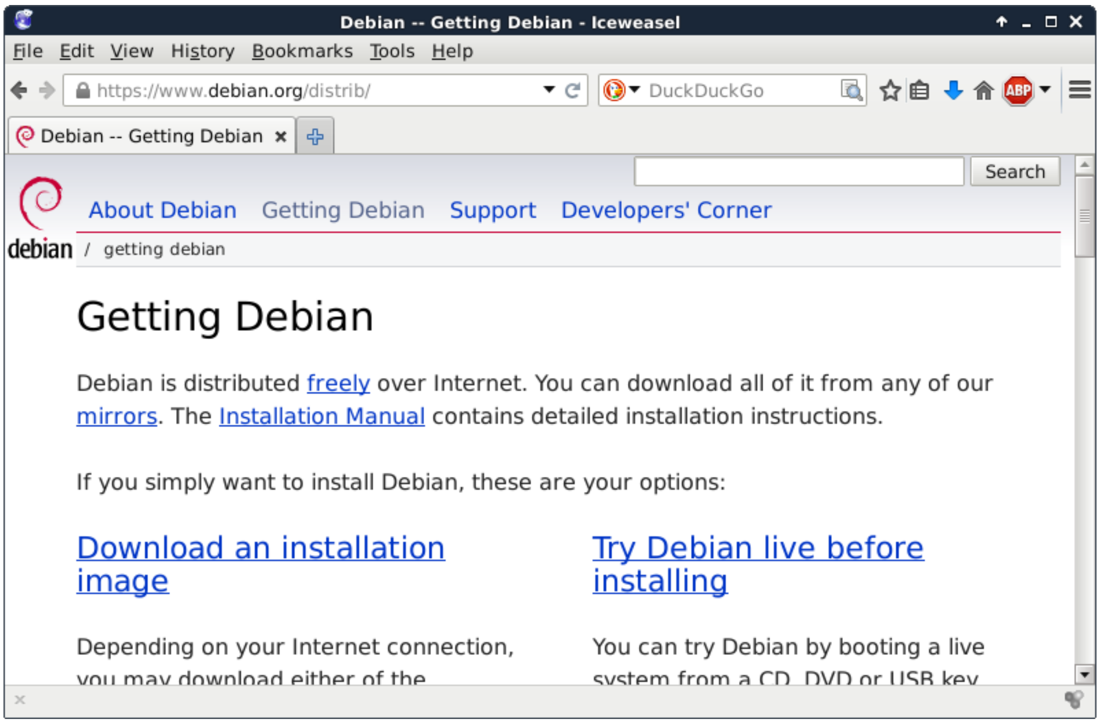
name	number	year
Woody	3.0	2002
Sarge	3.1	2005
Etch	4.0	2007
Lenny	5.0	2009
Squeeze	6.0	2011
Wheezy	7	2013
Jessie	8	2015

There is never a fixed date for the next `Debian` release. The next version is released when it is ready.

4.2. Downloading

All these screenshots were made in November 2014, which means `Debian 8` was still in 'testing' (but in 'freeze', so there will be no major changes when it is released).

Download Debian here:



Debian -- Getting Debian - Iceweasel

https://www.debian.org/distrib/

About Debian Getting Debian Support Developers' Corner

debian / getting debian

Getting Debian

Debian is distributed freely over Internet. You can download all of it from any of our mirrors. The Installation Manual contains detailed installation instructions.

If you simply want to install Debian, these are your options:

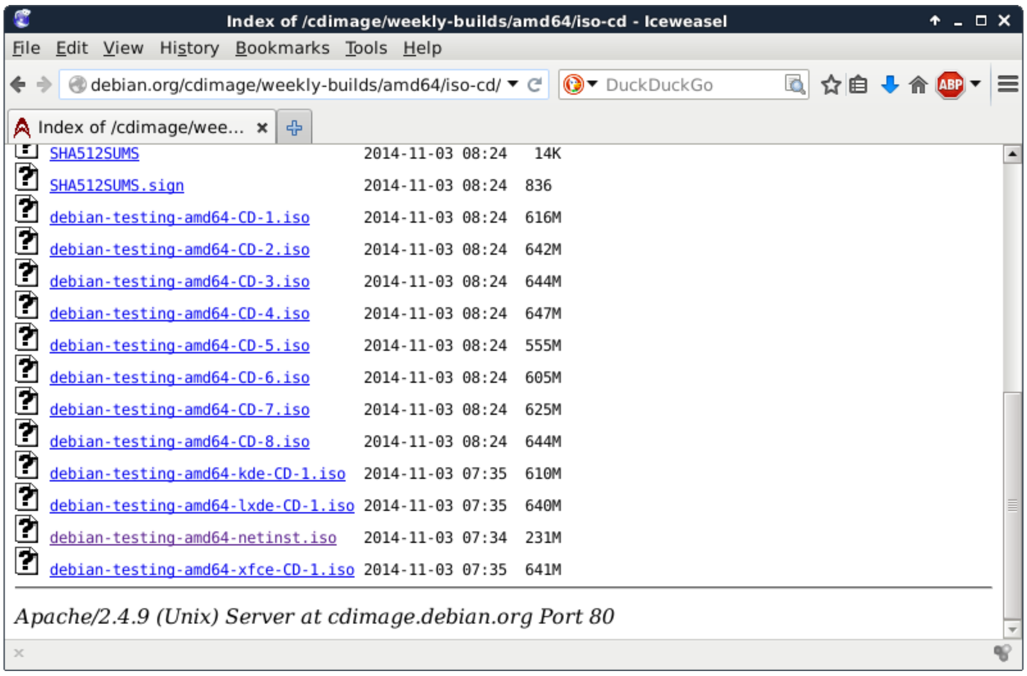
Download an installation image

Try Debian live before installing

Depending on your Internet connection, you may download either of the

You can try Debian by booting a live system from a CD, DVD or USB key

After a couple of clicks on that website, I ended up downloading `Debian 8` (testing) here. It should be only one click once `Debian 8` is released (somewhere in 2015).



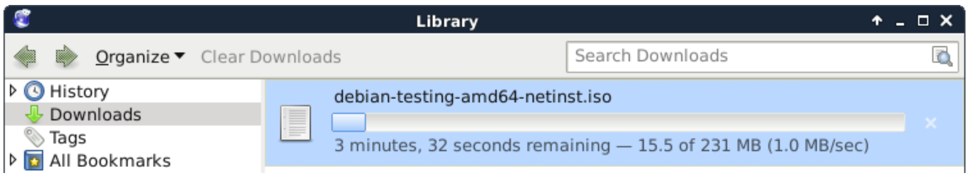
Index of /cdimage/weekly-builds/amd64/iso-cd - Iceweasel

debian.org/cdimage/weekly-builds/amd64/iso-cd/

```
SHA512SUMS                        2014-11-03 08:24   14K
SHA512SUMS.sign                   2014-11-03 08:24  836
debian-testing-amd64-CD-1.iso     2014-11-03 08:24  616M
debian-testing-amd64-CD-2.iso     2014-11-03 08:24  642M
debian-testing-amd64-CD-3.iso     2014-11-03 08:24  644M
debian-testing-amd64-CD-4.iso     2014-11-03 08:24  647M
debian-testing-amd64-CD-5.iso     2014-11-03 08:24  555M
debian-testing-amd64-CD-6.iso     2014-11-03 08:24  605M
debian-testing-amd64-CD-7.iso     2014-11-03 08:24  625M
debian-testing-amd64-CD-8.iso     2014-11-03 08:24  644M
debian-testing-amd64-kde-CD-1.iso 2014-11-03 07:35  610M
debian-testing-amd64-lxde-CD-1.iso 2014-11-03 07:35  640M
debian-testing-amd64-netinst.iso  2014-11-03 07:34  231M
debian-testing-amd64-xfce-CD-1.iso 2014-11-03 07:35  641M

Apache/2.4.9 (Unix) Server at cdimage.debian.org Port 80
```

You have many other options to download and install `Debian`. We will discuss them much later.

This small screenshot shows the downloading of a `netinst` .iso file. Most of the software will be downloaded during the installation. This also means that you will have the most recent version of all packages when the install is finished.



Library

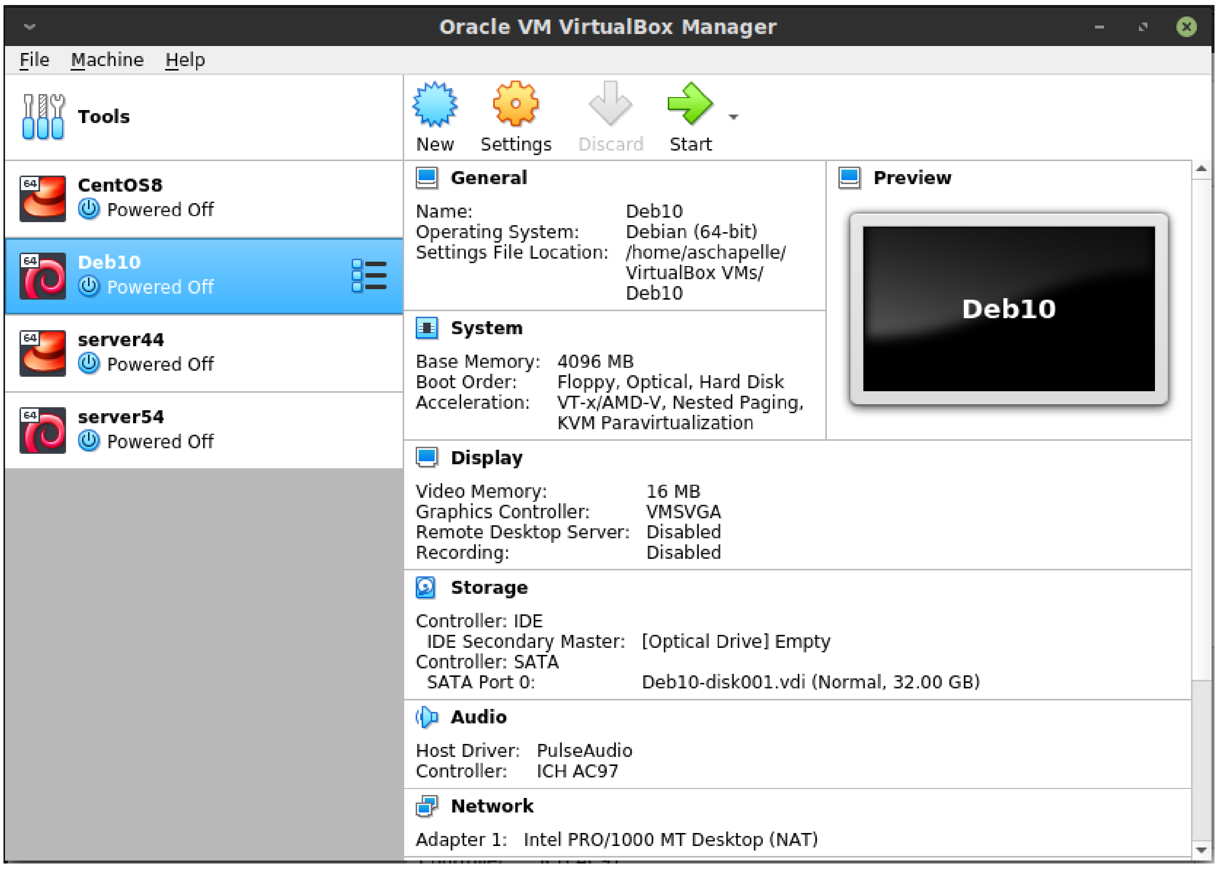
debian-testing-amd64-netinst.iso

3 minutes, 32 seconds remaining — 15.5 of 231 MB (1.0 MB/sec)

I already have Debian 8 installed on my laptop (hence the `student@linux` prompt). Anyway, this is the downloaded file just before starting the installation.

```
student@linux:~$ ls -hl debian-testing-amd64-netinst.iso
-rw-r--r-- 1 paul paul 231M Nov 10 17:59 debian-testing-amd64-netinst.iso
```

Create a new virtualbox machine (I already have five, you might have zero for now). Click the `New` button to start a wizard that will help you create a virtual machine.



General

```
Name:                   Deb10
Operating System:       Debian (64-bit)
Settings File Location: /home/aschapelle/
                        VirtualBox VMs/
                        Deb10
```

System

```
Base Memory:   4096 MB
Boot Order:    Floppy, Optical, Hard Disk
Acceleration:  VT-x/AMD-V, Nested Paging,
               KVM Paravirtualization
```

Display

```
Video Memory:           16 MB
Graphics Controller:    VMSVGA
Remote Desktop Server:  Disabled
Recording:              Disabled
```

Storage

```
Controller: IDE
  IDE Secondary Master:  [Optical Drive] Empty
Controller: SATA
  SATA Port 0:           Deb10-disk001.vdi (Normal, 32.00 GB)
```

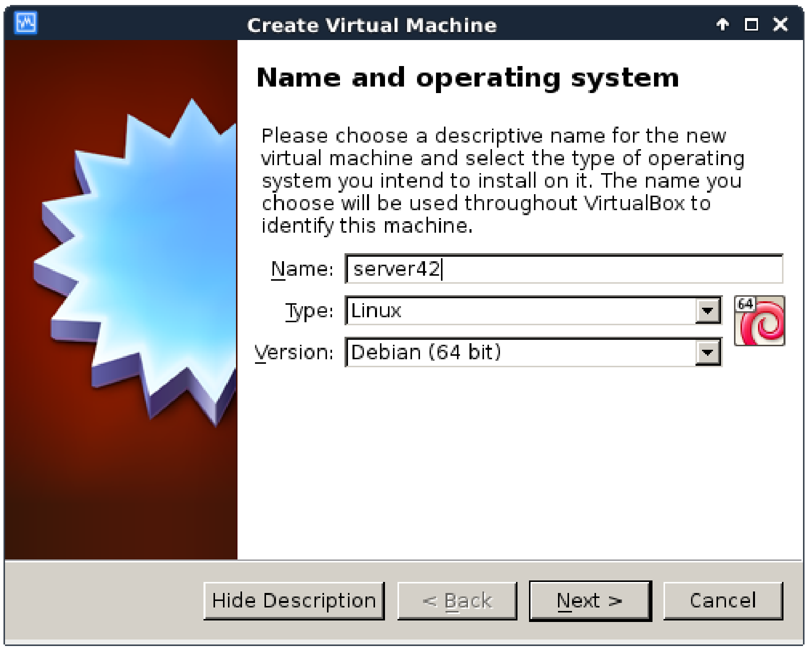
Audio

```
Host Driver:  PulseAudio
Controller:   ICH AC97
```

Network

```
Adapter 1:  Intel PRO/1000 MT Desktop (NAT)
```

The machine needs a name, this screenshot shows that I named it `server42`.



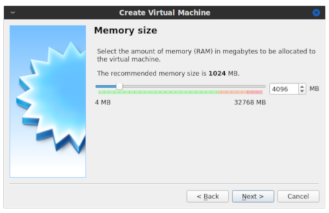
Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: server42
Type: Linux
Version: Debian (64 bit)

Most of the defaults in Virtualbox are ok.

512MB of RAM is enough to practice all the topics in this book.



Create Virtual Machine

Memory size

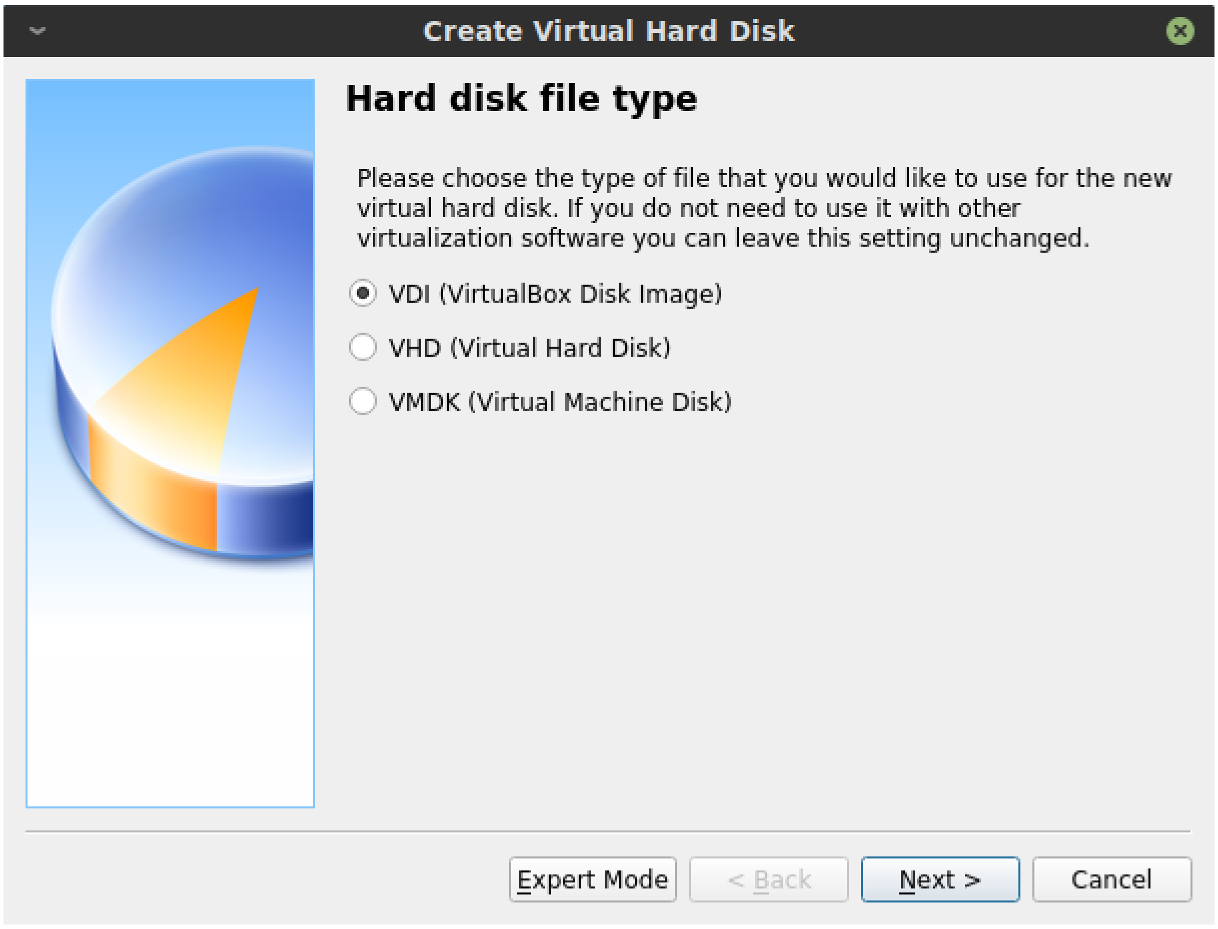
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

4096 MB

4 MB 32768 MB

We do not care about the virtual disk format.



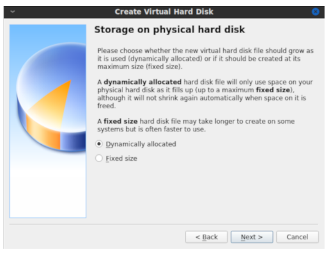
Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- [✓ VDI (VirtualBox Disk Image)]
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)

Choosing `dynamically allocated` will save you some disk space (for a small performance hit).



Create Virtual Hard Disk

Storage on physical hard disk

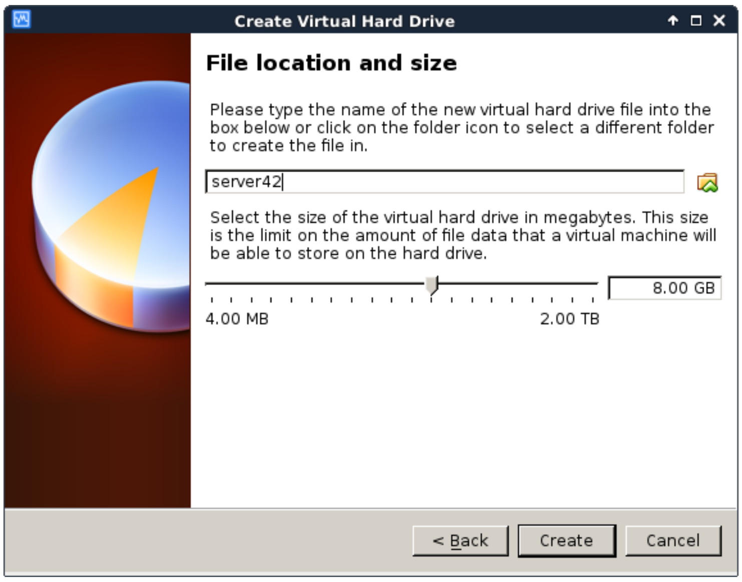
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

- [✓ Dynamically allocated]
- Fixed size

8GB should be plenty for learning about Linux servers.



Please type the name of the new virtual hard drive file into the box below or click on the folder icon to select a different folder to create the file in.

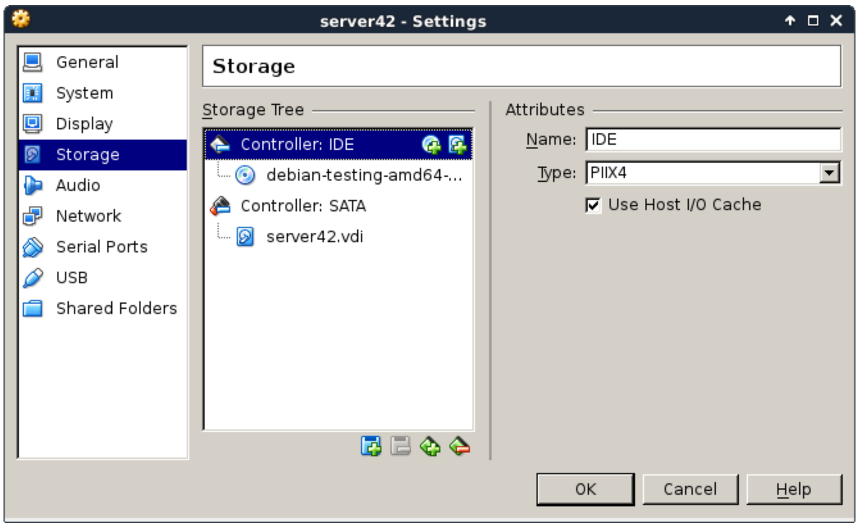
server42

Select the size of the virtual hard drive in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard drive.

8.00 GB

This finishes the wizard. You virtual machine is almost ready to begin the installation.

First, make sure that you attach the downloaded .iso image to the virtual CD drive. (by opening `Settings`, `Storage` followed by a mouse click on the round CD icon)

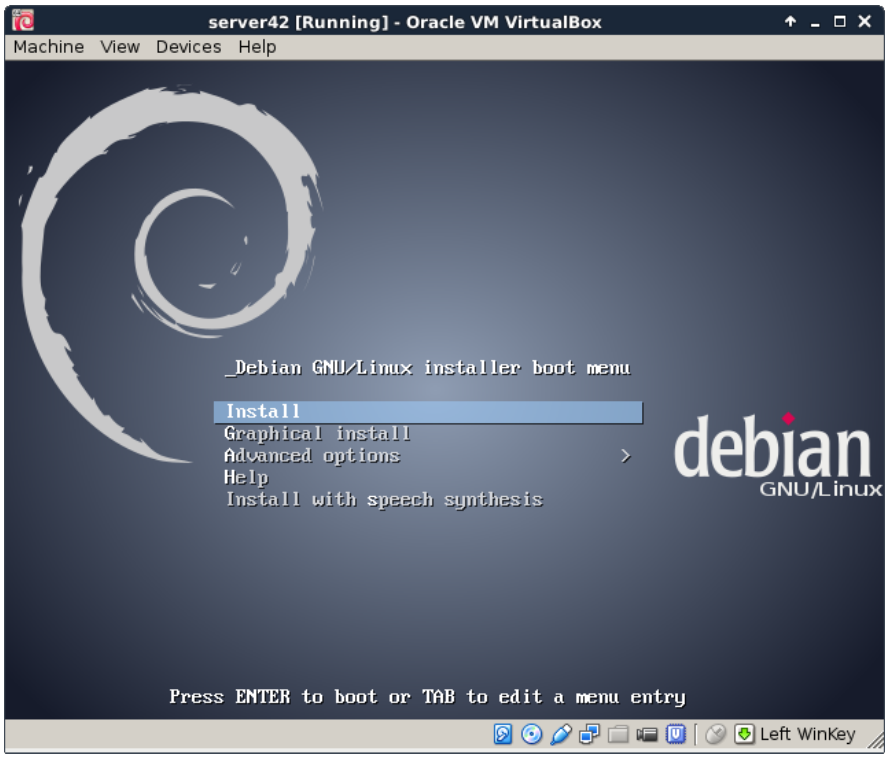


Controller: IDE
debian-testing-amd64-...
Controller: SATA
server42.vdi

Name: IDE
Type: PIIX4
[✓] Use Host I/O Cache

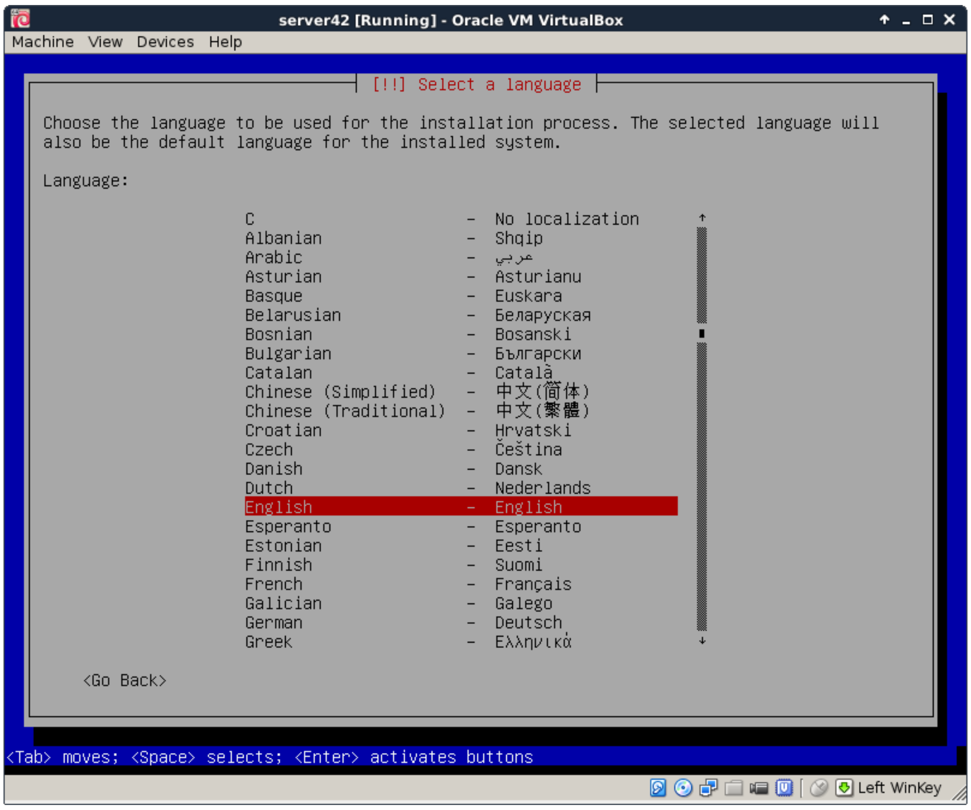
Personally I also disable sound and usb, because I never use these features. I also remove the floppy disk and use a PS/2 mouse pointer. This is probably not very important, but I like the idea that it saves some resources.

Now boot the virtual machine and begin the actual installation. After a couple of seconds you should see a screen similar to this. Choose `Install` to begin the installation of Debian.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
debian GNU/Linux
Left WinKey

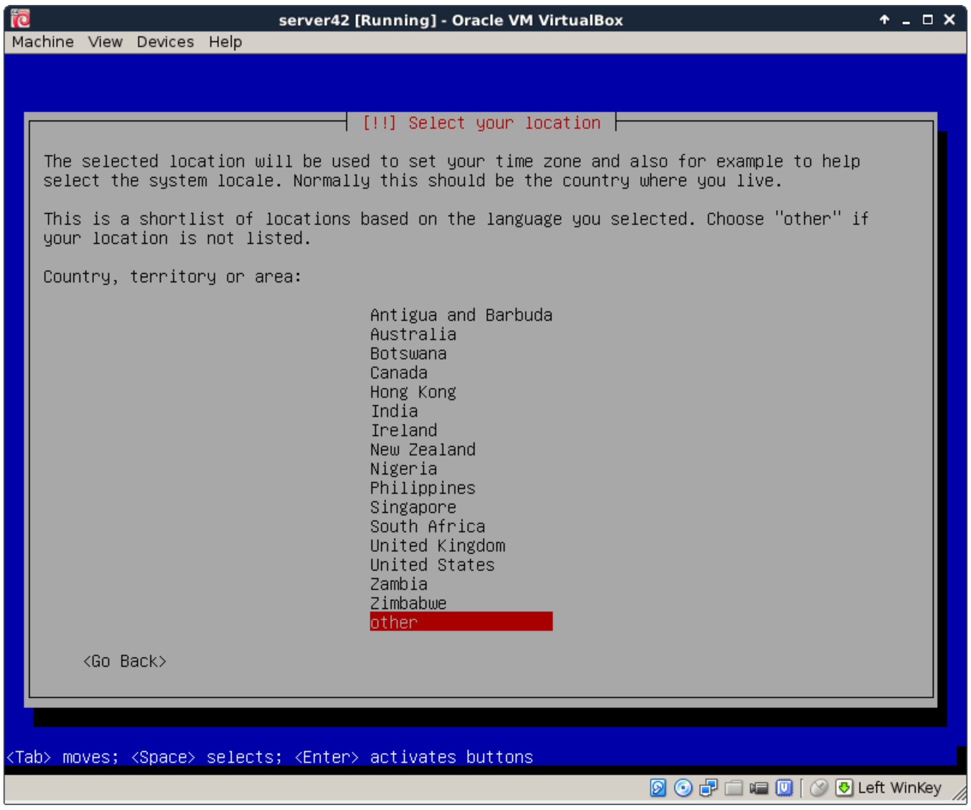
```
Debian GNU/Linux installer boot menu

Install
Graphical install
Advanced options                              >
Help
Install with speech synthesis

Press ENTER to boot or TAB to edit a menu entry
```

First select the language you want to use.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

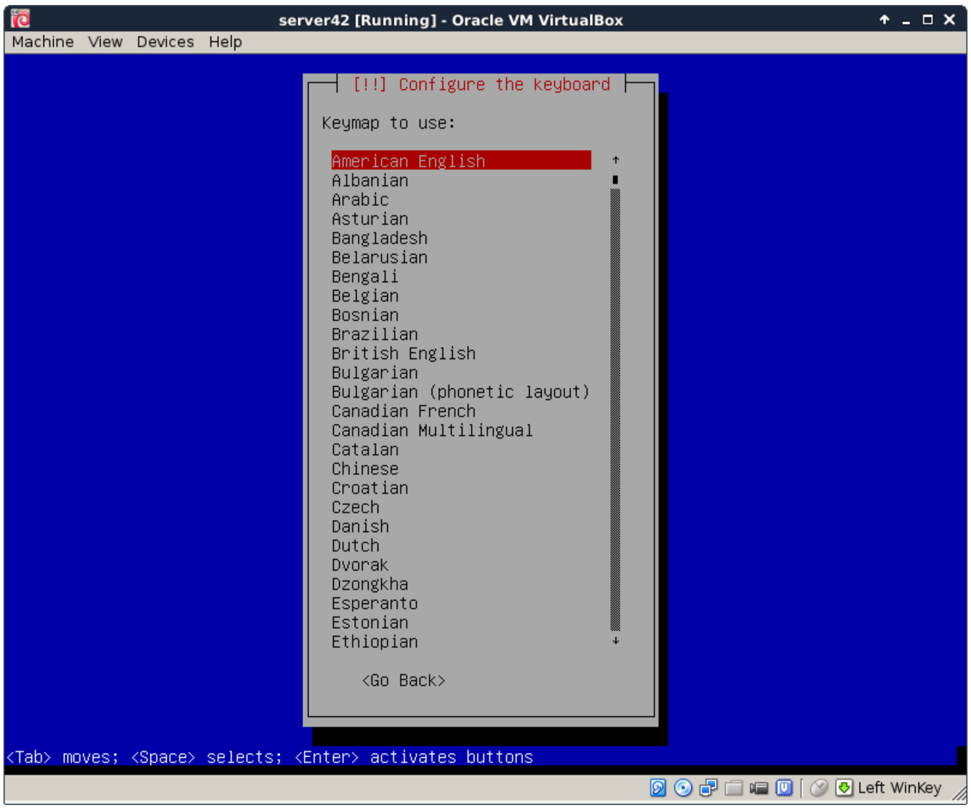
```
[!!] Select a language

Choose the language to be used for the installation process. The selected language will
also be the default language for the installed system.

Language:

                     C                       -  No localization
                     Albanian                -  Shqip
                     Arabic                  -  عربي
                     Asturian                -  Asturianu
                     Basque                  -  Euskara
                     Belarusian              -  Беларуская
                     Bosnian                 -  Bosanski
                     Bulgarian               -  Български
                     Catalan                 -  Català
                     Chinese (Simplified)    -  中文(简体)
                     Chinese (Traditional)   -  中文(繁體)
                     Croatian                -  Hrvatski
                     Czech                   -  Čeština
                     Danish                  -  Dansk
                     Dutch                   -  Nederlands
                     English                 -  English
                     Esperanto               -  Esperanto
                     Estonian                -  Eesti
                     Finnish                 -  Suomi
                     French                  -  Français
                     Galician                -  Galego
                     German                  -  Deutsch
                     Greek                   -  Ελληνικά

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Choose your country. This information will be used to suggest a download mirror.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

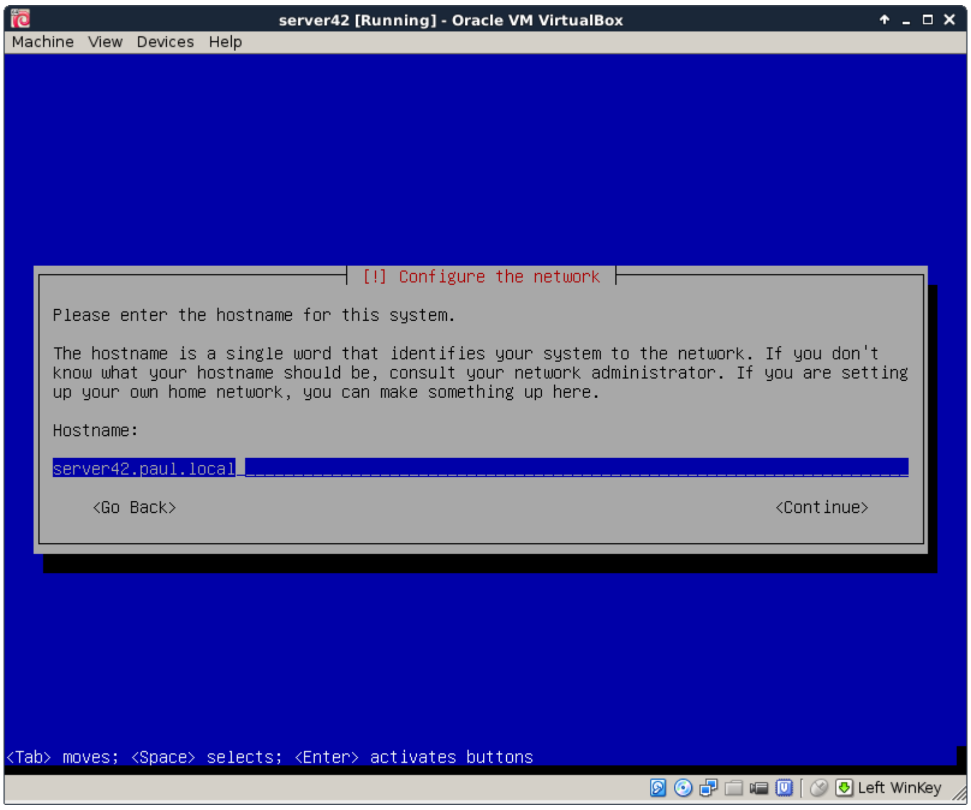
```
[!!] Select your location

The selected location will be used to set your time zone and also for example to help
select the system locale. Normally this should be the country where you live.

This is a shortlist of locations based on the language you selected. Choose "other" if
your location is not listed.

Country, territory or area:

                                  Antigua and Barbuda
                                  Australia
                                  Botswana
                                  Canada
                                  Hong Kong
                                  India
                                  Ireland
                                  New Zealand
                                  Nigeria
                                  Philippines
                                  Singapore
                                  South Africa
                                  United Kingdom
                                  United States
                                  Zambia
                                  Zimbabwe
                                  other

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Choose the correct keyboard. On servers this is of no importance since most servers are remotely managed via `ssh`.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

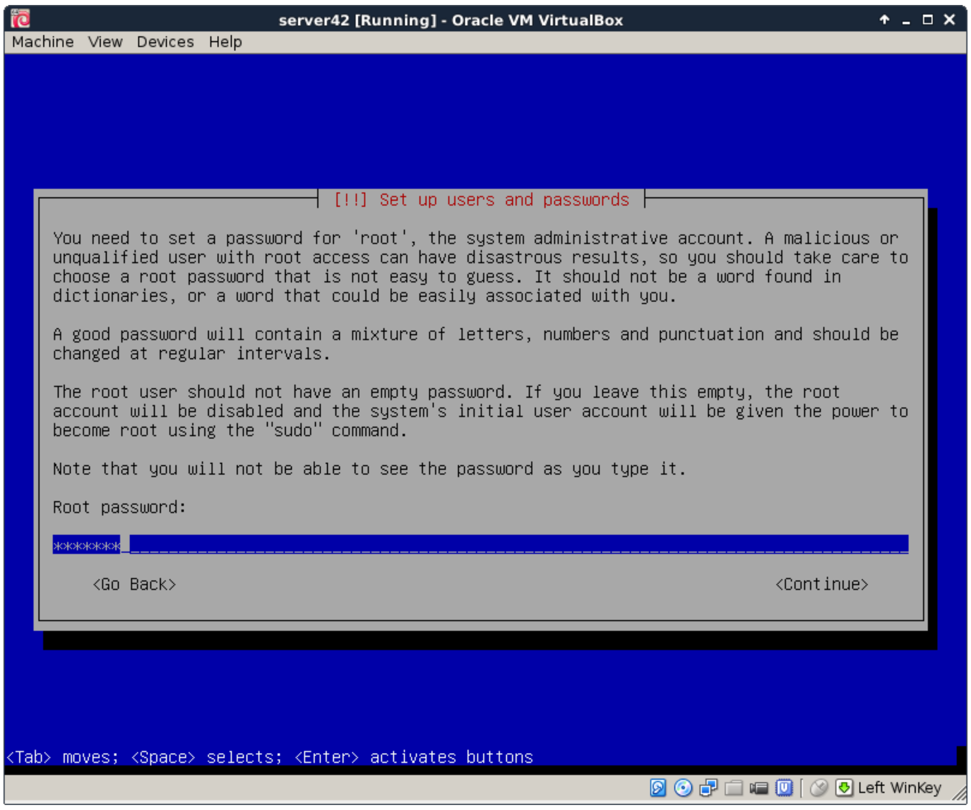
```
[!!] Configure the keyboard

Keymap to use:

 American English
 Albanian
 Arabic
 Asturian
 Bangladesh
 Belarusian
 Bengali
 Belgian
 Bosnian
 Brazilian
 British English
 Bulgarian
 Bulgarian (phonetic layout)
 Canadian French
 Canadian Multilingual
 Catalan
 Chinese
 Croatian
 Czech
 Danish
 Dutch
 Dvorak
 Dzongkha
 Esperanto
 Estonian
 Ethiopian

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Enter a `hostname` (with `fqdn` to set a `dnsdomainname`).


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

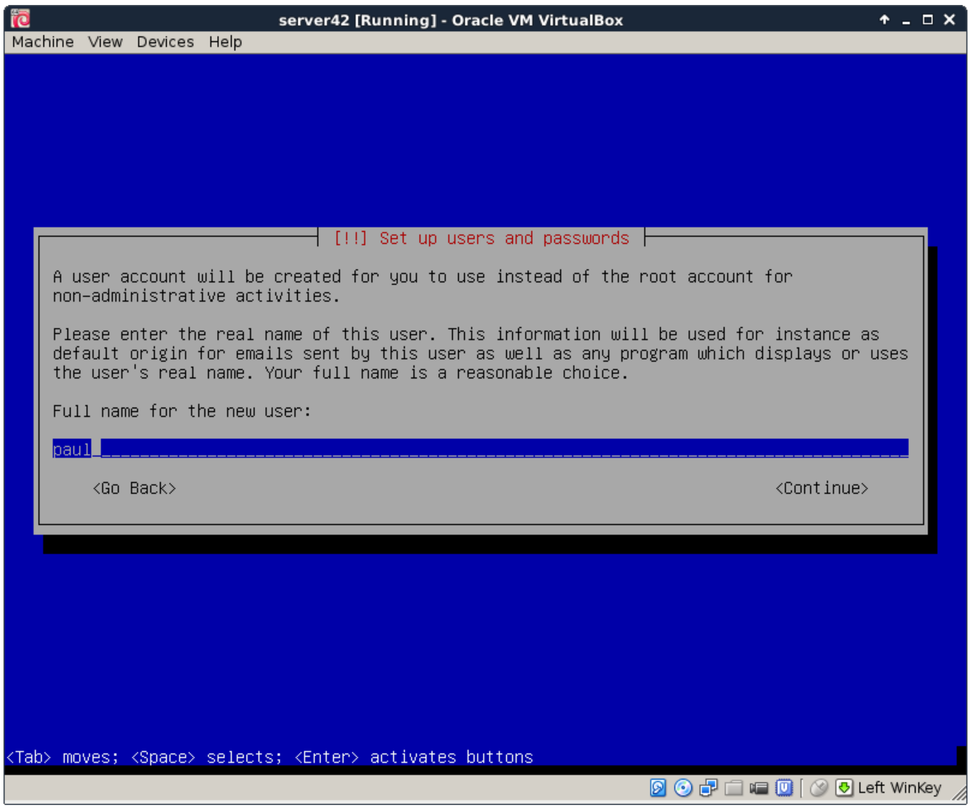
```
[!] Configure the network

Please enter the hostname for this system.

The hostname is a single word that identifies your system to the network. If you don't
know what your hostname should be, consult your network administrator. If you are setting
up your own home network, you can make something up here.

Hostname:

server42.paul.local

    <Go Back>                                                        <Continue>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Give the `root` user a password. Remember this password (or use `hunter2`).


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

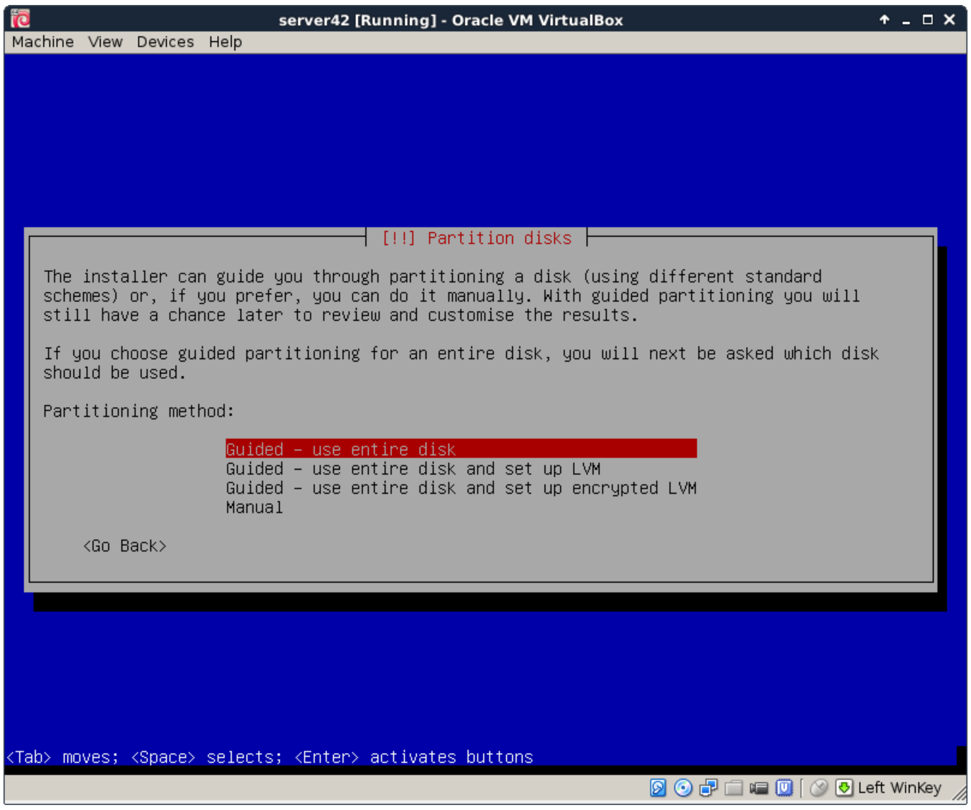
```
[!!] Set up users and passwords

You need to set a password for 'root', the system administrative account. A malicious or
unqualified user with root access can have disastrous results, so you should take care to
choose a root password that is not easy to guess. It should not be a word found in
dictionaries, or a word that could be easily associated with you.

A good password will contain a mixture of letters, numbers and punctuation and should be
changed at regular intervals.

The root user should not have an empty password. If you leave this empty, the root
account will be disabled and the system's initial user account will be given the power to
become root using the "sudo" command.

Note that you will not be able to see the password as you type it.

Root password:

*******

    <Go Back>                                                        <Continue>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

It is adviced to also create a normal user account. I don't give my full name, Debian 8 accepts an identical username and full name `paul`.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

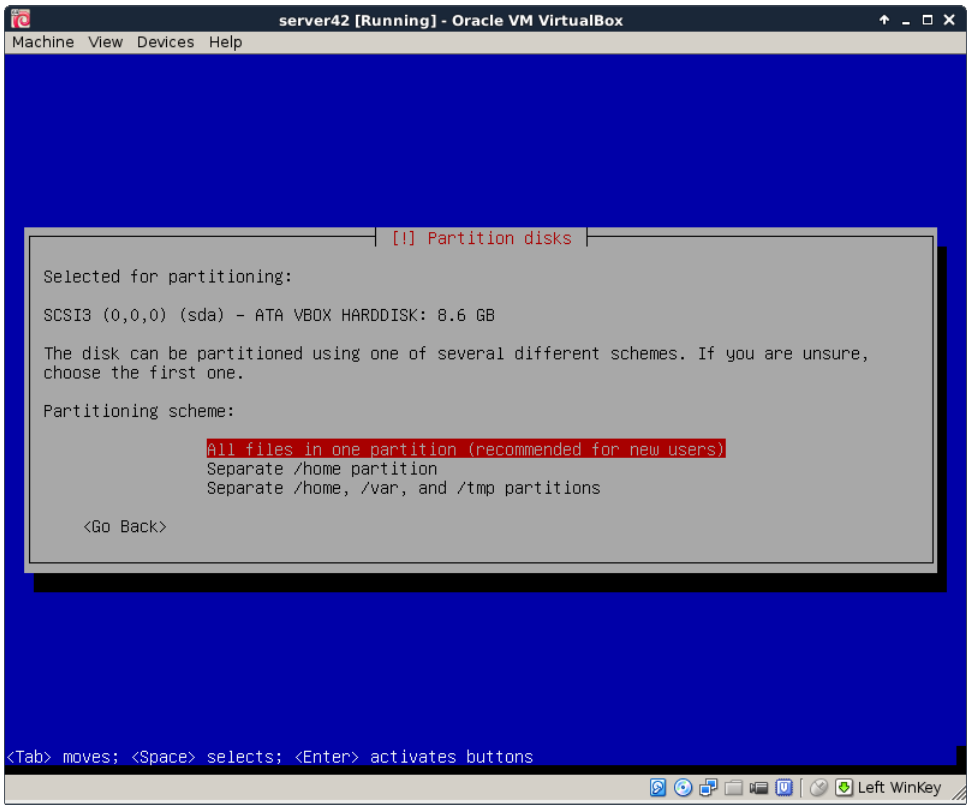
```
[!!] Set up users and passwords

A user account will be created for you to use instead of the root account for
non-administrative activities.

Please enter the real name of this user. This information will be used for instance as
default origin for emails sent by this user as well as any program which displays or uses
the user's real name. Your full name is a reasonable choice.

Full name for the new user:

paul

    <Go Back>                                                          <Continue>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

The `use entire disk` refers to the `virtual disk` that you created before in `Virtualbox`..


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

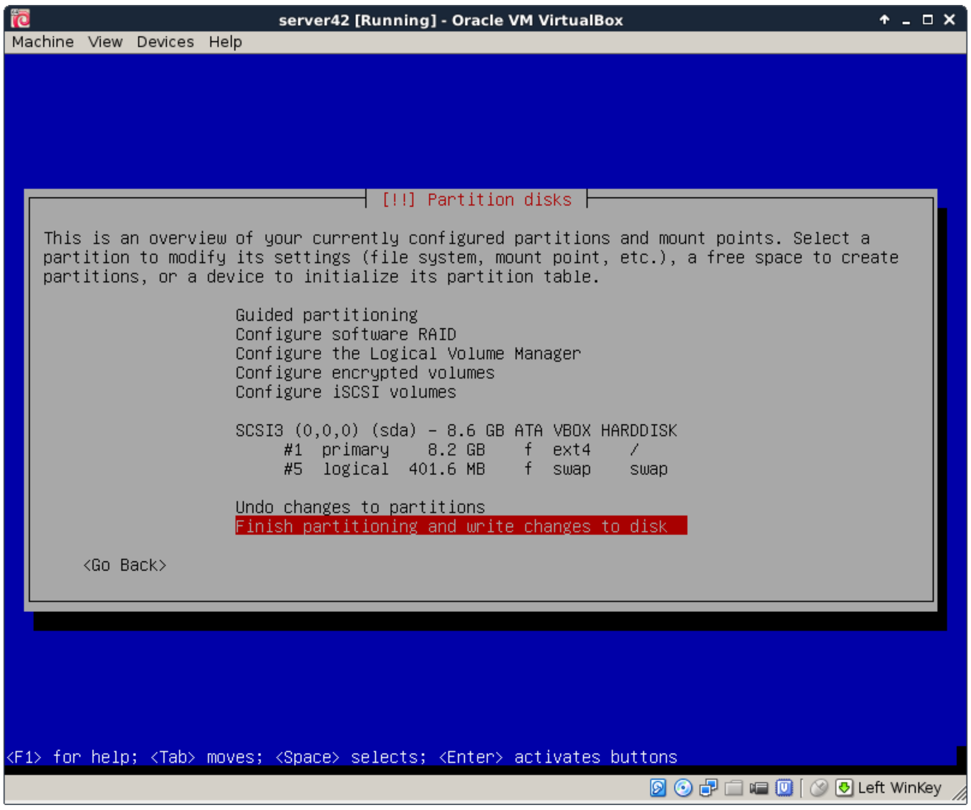
```
[!!] Partition disks

The installer can guide you through partitioning a disk (using different standard
schemes) or, if you prefer, you can do it manually. With guided partitioning you will
still have a chance later to review and customise the results.

If you choose guided partitioning for an entire disk, you will next be asked which disk
should be used.

Partitioning method:

                   Guided - use entire disk
                   Guided - use entire disk and set up LVM
                   Guided - use entire disk and set up encrypted LVM
                   Manual

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Again the default is probably what you want. Only change partitioning if you really know what you are doing.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

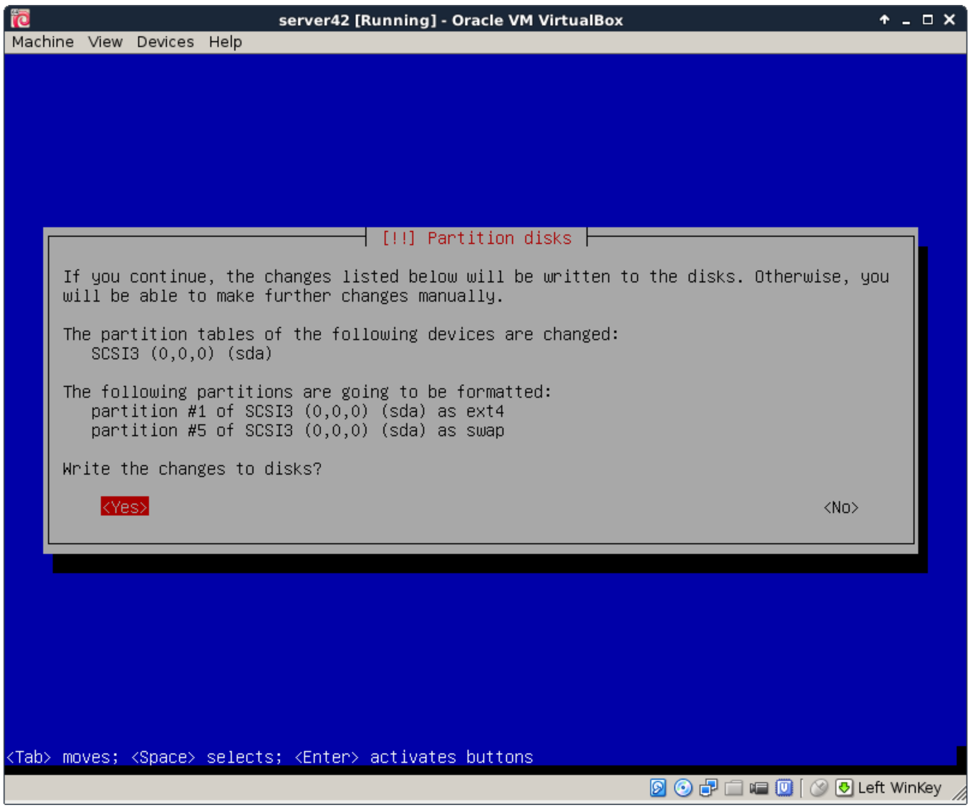
```
[!] Partition disks

Selected for partitioning:

SCSI3 (0,0,0) (sda) - ATA VBOX HARDDISK: 8.6 GB

The disk can be partitioned using one of several different schemes. If you are unsure,
choose the first one.

Partitioning scheme:

                    All files in one partition (recommended for new users)
                    Separate /home partition
                    Separate /home, /var, and /tmp partitions

     <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Accept the partition layout (again only change if you really know what you are doing).


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

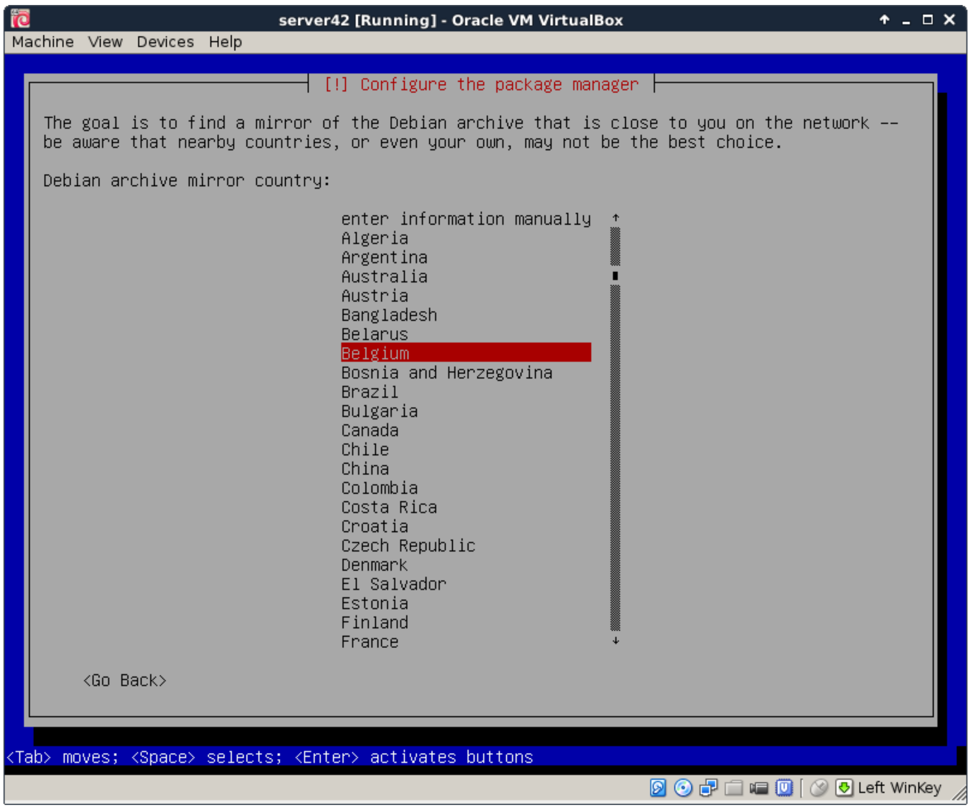
```
[!!] Partition disks

This is an overview of your currently configured partitions and mount points. Select a
partition to modify its settings (file system, mount point, etc.), a free space to create
partitions, or a device to initialize its partition table.

                        Guided partitioning
                        Configure software RAID
                        Configure the Logical Volume Manager
                        Configure encrypted volumes
                        Configure iSCSI volumes

                        SCSI3 (0,0,0) (sda) - 8.6 GB ATA VBOX HARDDISK
                              #1  primary    8.2 GB    f  ext4     /
                              #5  logical  401.6 MB    f  swap     swap

                        Undo changes to partitions
                        Finish partitioning and write changes to disk

     <Go Back>

<F1> for help; <Tab> moves; <Space> selects; <Enter> activates buttons
```

This is the point of no return, the magical moment where pressing yes will forever erase data on the (virtual) computer.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

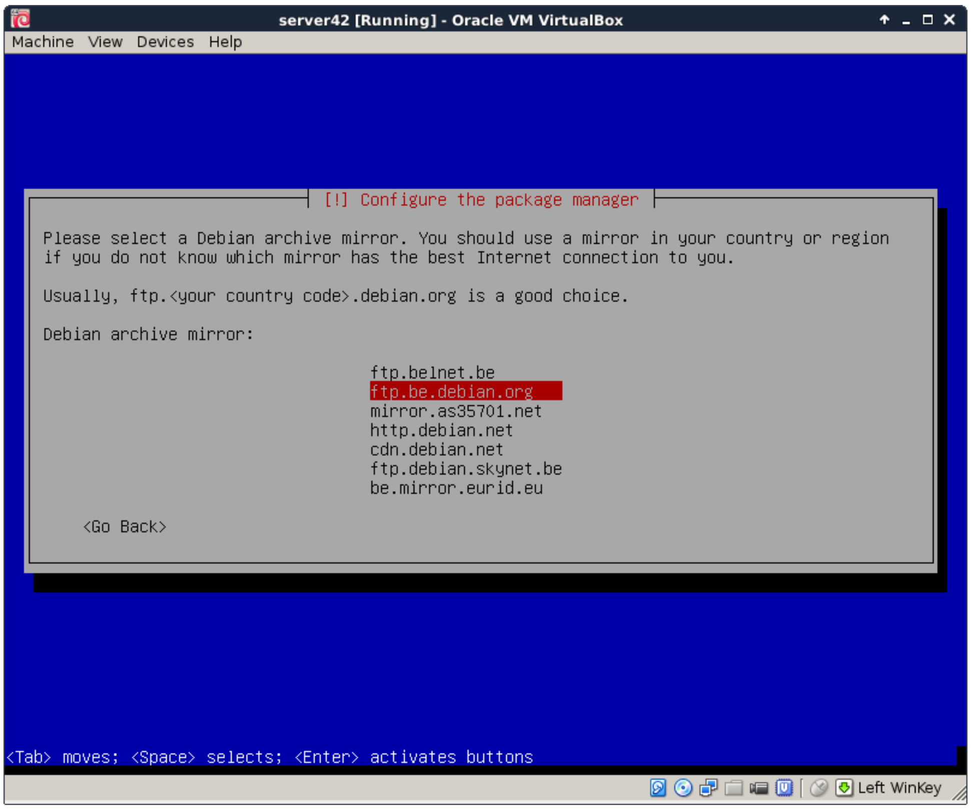
```
[!!] Partition disks

If you continue, the changes listed below will be written to the disks. Otherwise, you
will be able to make further changes manually.

The partition tables of the following devices are changed:
   SCSI3 (0,0,0) (sda)

The following partitions are going to be formatted:
   partition #1 of SCSI3 (0,0,0) (sda) as ext4
   partition #5 of SCSI3 (0,0,0) (sda) as swap

Write the changes to disks?

     <Yes>                                                                    <No>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Software is downloaded from a mirror repository, preferably choose one that is close by (as in the same country).


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

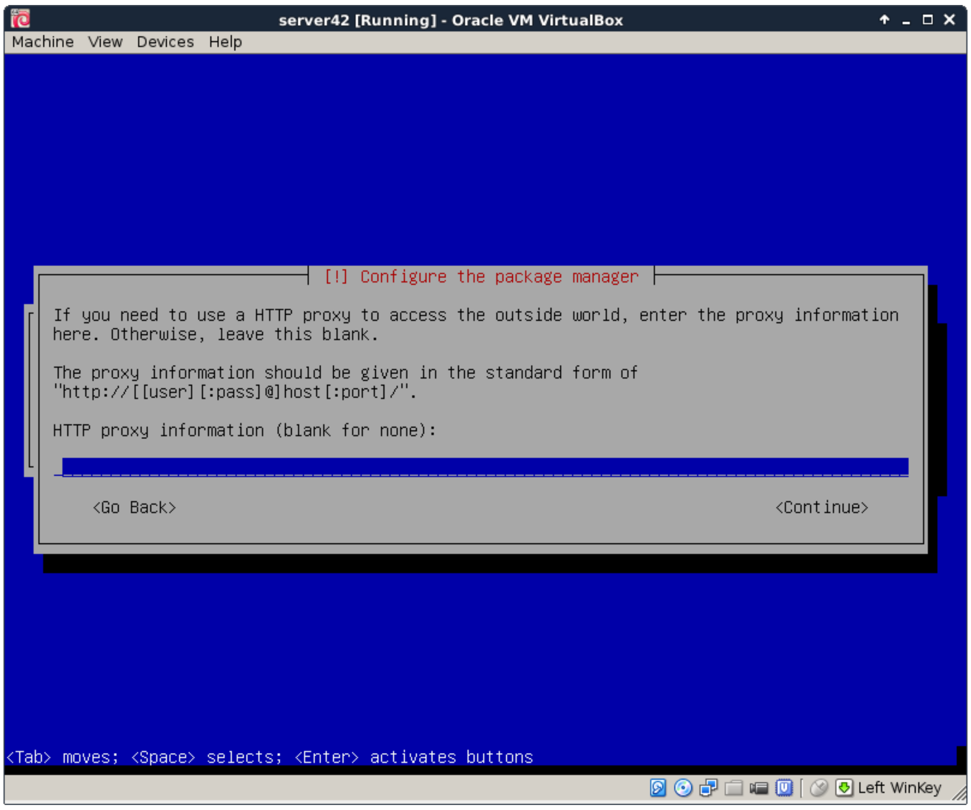
```
[!] Configure the package manager

The goal is to find a mirror of the Debian archive that is close to you on the network --
be aware that nearby countries, or even your own, may not be the best choice.

Debian archive mirror country:

                               enter information manually
                               Algeria
                               Argentina
                               Australia
                               Austria
                               Bangladesh
                               Belarus
                               Belgium
                               Bosnia and Herzegovina
                               Brazil
                               Bulgaria
                               Canada
                               Chile
                               China
                               Colombia
                               Costa Rica
                               Croatia
                               Czech Republic
                               Denmark
                               El Salvador
                               Estonia
                               Finland
                               France

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

This setup was done in Belgium.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

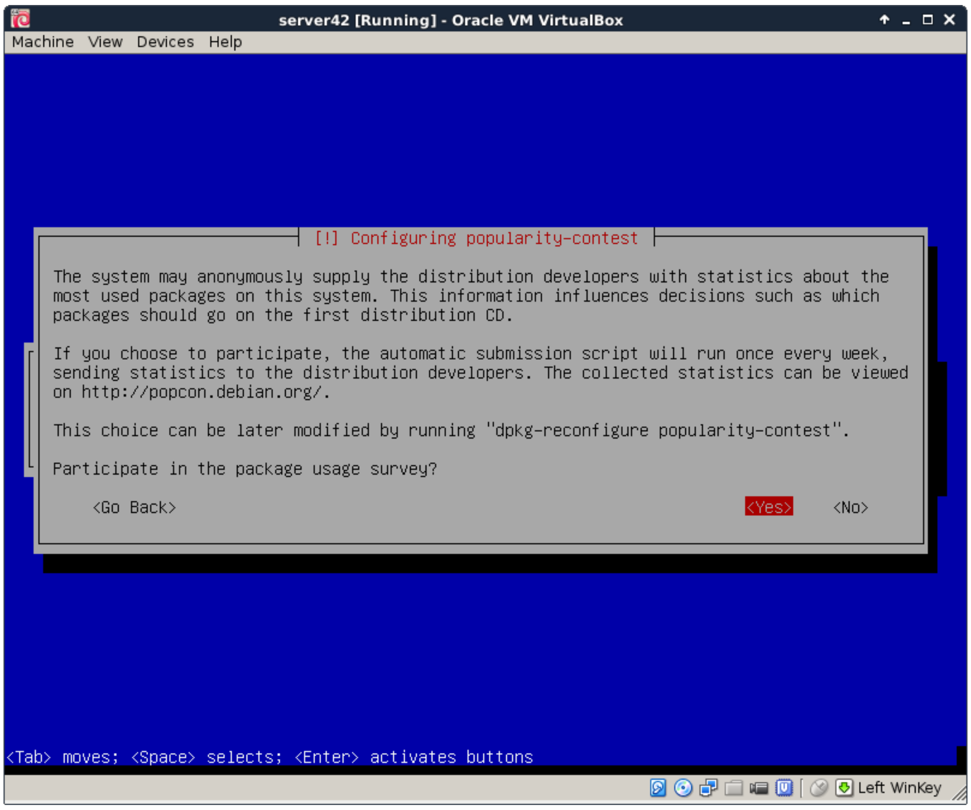
```
[!] Configure the package manager

Please select a Debian archive mirror. You should use a mirror in your country or region
if you do not know which mirror has the best Internet connection to you.

Usually, ftp.<your country code>.debian.org is a good choice.

Debian archive mirror:

                                  ftp.belnet.be
                                  ftp.be.debian.org
                                  mirror.as35701.net
                                  http.debian.net
                                  cdn.debian.net
                                  ftp.debian.skynet.be
                                  be.mirror.eurid.eu

    <Go Back>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Leave the proxy field empty (unless you are sure that you are behind a proxy server).


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

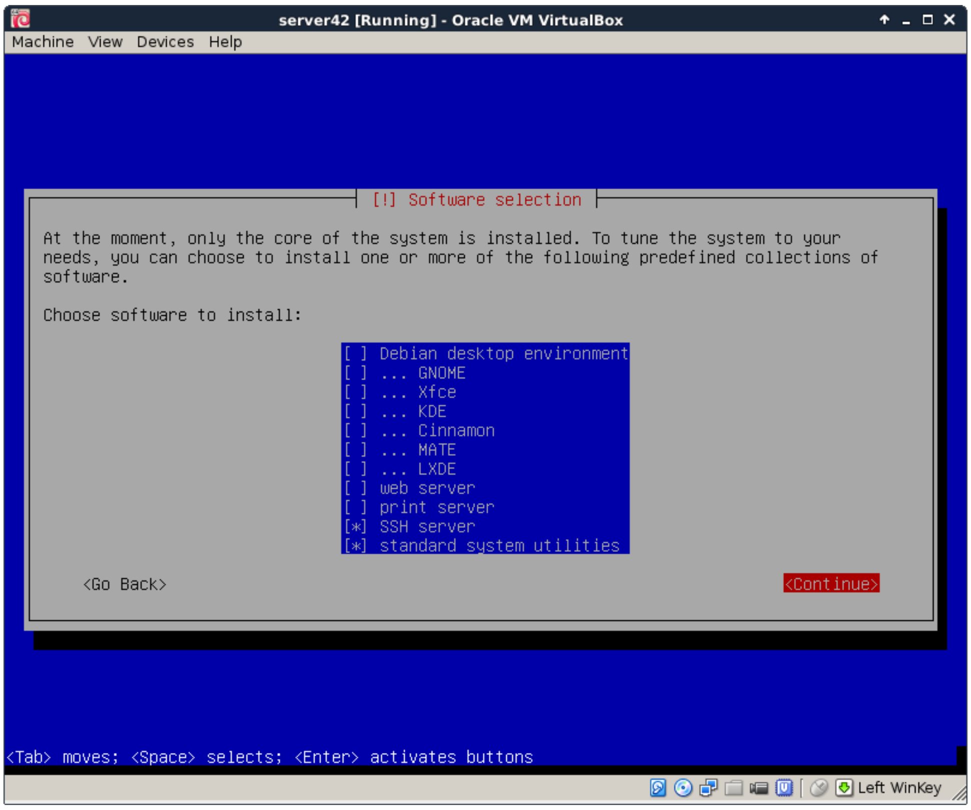
```
[!] Configure the package manager

If you need to use a HTTP proxy to access the outside world, enter the proxy information
here. Otherwise, leave this blank.

The proxy information should be given in the standard form of
"http://[[user][:pass]@]host[:port]/".

HTTP proxy information (blank for none):

    <Go Back>                                                           <Continue>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Choose whether you want to send anonymous statistics to the Debian project (it gathers data about installed packages). You can view the statistics here `http://popcon.debian.org/`.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

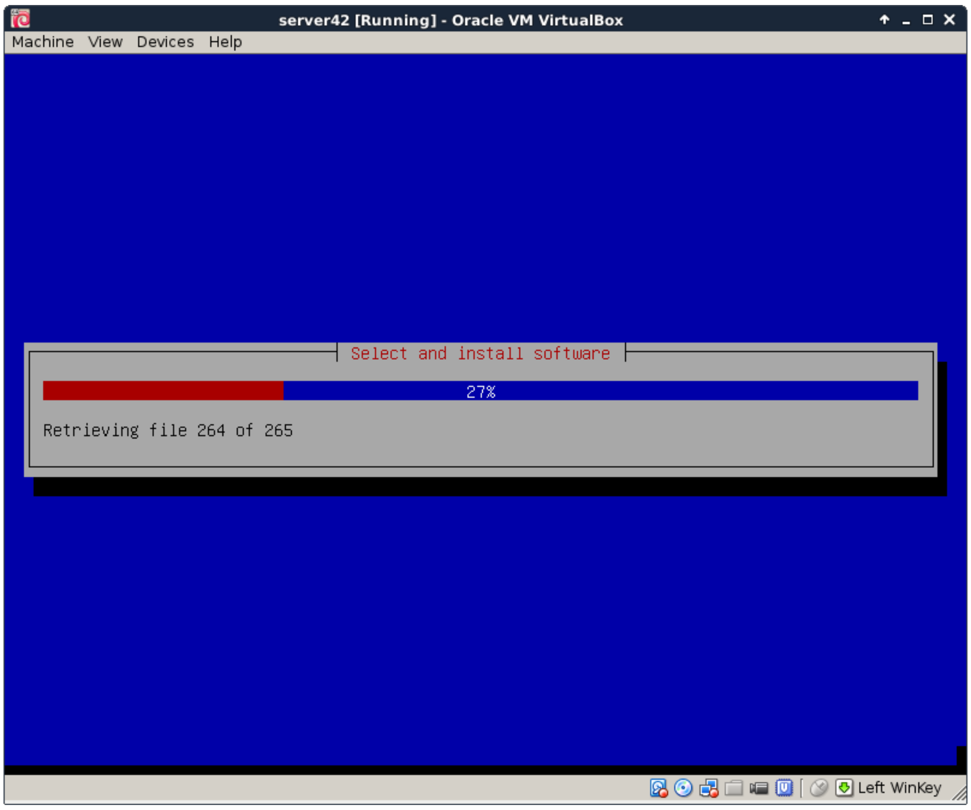
```
[!] Configuring popularity-contest

The system may anonymously supply the distribution developers with statistics about the
most used packages on this system. This information influences decisions such as which
packages should go on the first distribution CD.

If you choose to participate, the automatic submission script will run once every week,
sending statistics to the distribution developers. The collected statistics can be viewed
on http://popcon.debian.org/.

This choice can be later modified by running "dpkg-reconfigure popularity-contest".

Participate in the package usage survey?

    <Go Back>                                                       <Yes>     <No>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Choose what software to install, we do not need any graphical stuff for this training.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

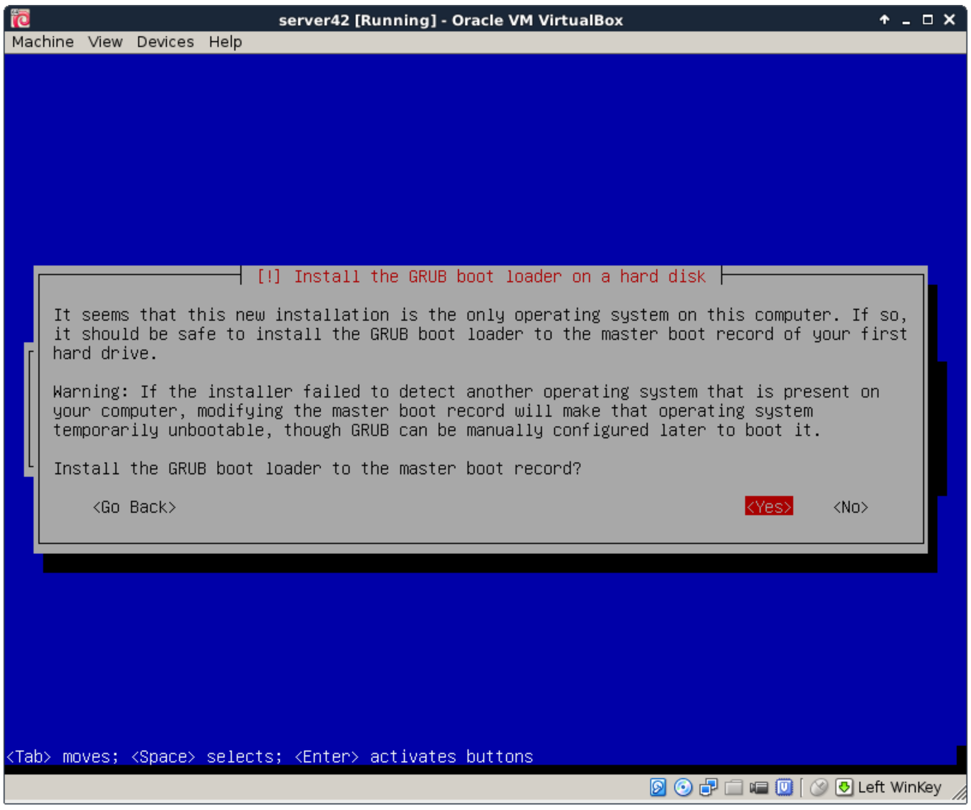
```
[!] Software selection

At the moment, only the core of the system is installed. To tune the system to your
needs, you can choose to install one or more of the following predefined collections of
software.

Choose software to install:

                              [ ] Debian desktop environment
                              [ ] ... GNOME
                              [ ] ... Xfce
                              [ ] ... KDE
                              [ ] ... Cinnamon
                              [ ] ... MATE
                              [ ] ... LXDE
                              [ ] web server
                              [ ] print server
                              [*] SSH server
                              [*] standard system utilities

    <Go Back>                                                          <Continue>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

The latest versions are being downloaded.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

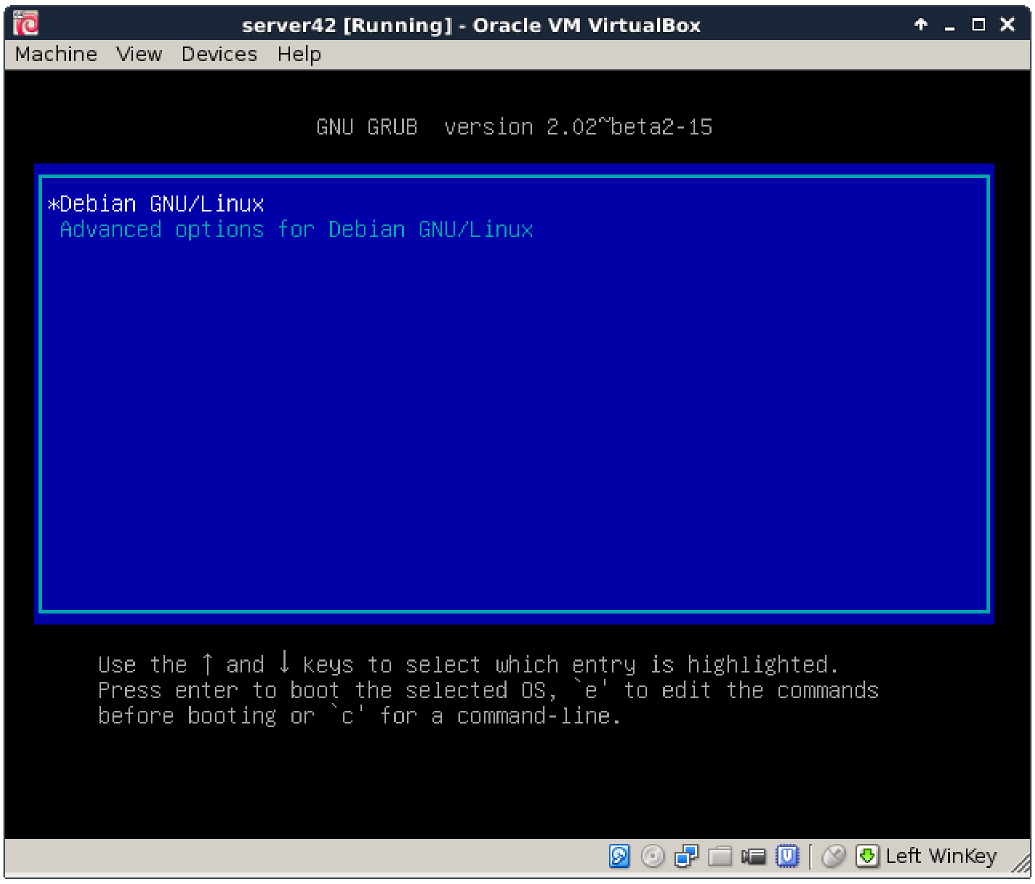
```
Select and install software

27%

Retrieving file 264 of 265
```

Say yes to install the bootloader on the virtual machine.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

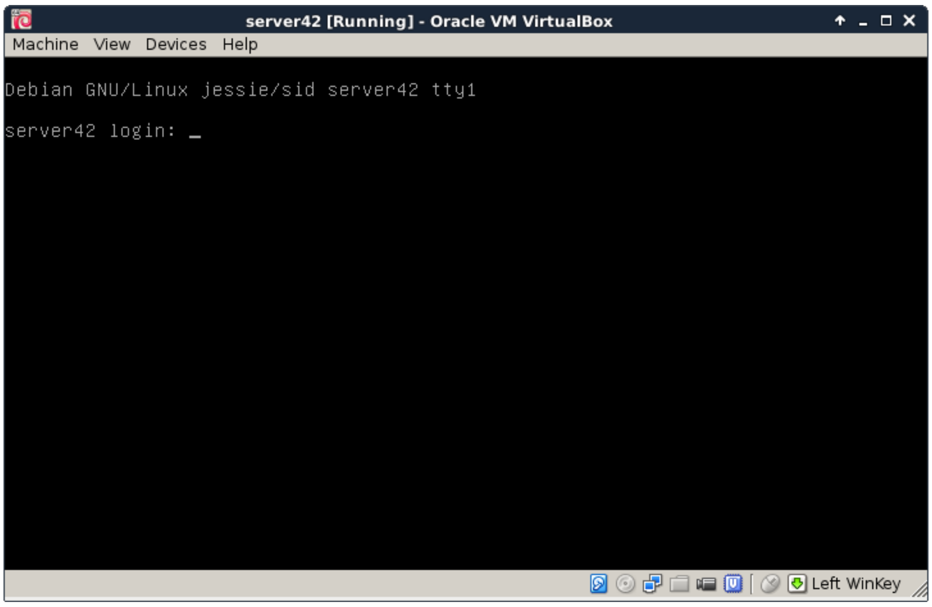
```
[!] Install the GRUB boot loader on a hard disk

It seems that this new installation is the only operating system on this computer. If so,
it should be safe to install the GRUB boot loader to the master boot record of your first
hard drive.

Warning: If the installer failed to detect another operating system that is present on
your computer, modifying the master boot record will make that operating system
temporarily unbootable, though GRUB can be manually configured later to boot it.

Install the GRUB boot loader to the master boot record?

    <Go Back>                                                       <Yes>     <No>

<Tab> moves; <Space> selects; <Enter> activates buttons
```

Booting for the first time shows the grub screen


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

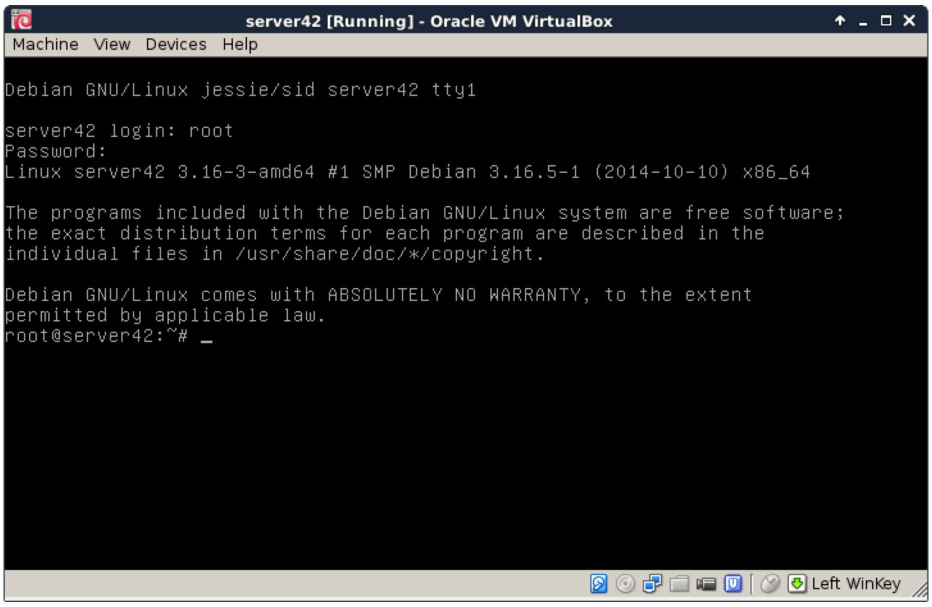
```
                    GNU GRUB  version 2.02~beta2-15

 *Debian GNU/Linux
  Advanced options for Debian GNU/Linux

      Use the ↑ and ↓ keys to select which entry is highlighted.
      Press enter to boot the selected OS, `e' to edit the commands
      before booting or `c' for a command-line.
```

A couple seconds later you should see a lot of text scrolling of the screen (`dmesg`). After which you are presented with this `getty` and are allowed your first logon.


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
Debian GNU/Linux jessie/sid server42 tty1

server42 login: _
```

You should now be able to log on to your virtual machine with the `root` account. Do you remember the password ? Was it `hunter2` ?


server42 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
Debian GNU/Linux jessie/sid server42 tty1

server42 login: root
Password:
Linux server42 3.16-3-amd64 #1 SMP Debian 3.16.5-1 (2014-10-10) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@server42:~# _
```

The screenshots in this book will look like this from now on. You can just type those commands in the terminal (after you logged on).

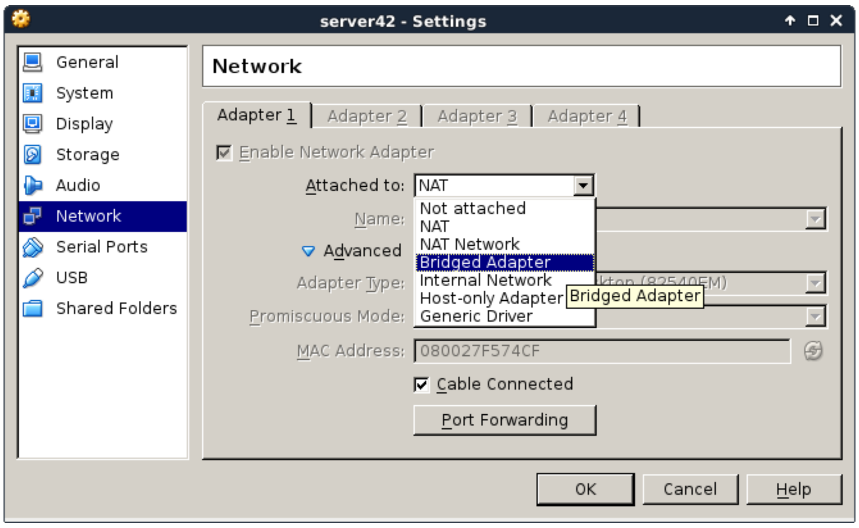
```
root@linux:~# who am i
root     tty1         2014-11-10 18:21
root@linux:~# hostname
server42
root@linux:~# date
Mon Nov 10 18:21:56 CET 2014
```

4.3. virtualbox networking

You can also log on from remote (or from your Windows/Mac/Linux host computer) using `ssh` or `putty`. Change the `network` settings in the virtual machine to `bridge`. This will enable your virtual machine to receive an ip address from your local dhcp server.

The default virtualbox networking is to attach virtual network cards to `nat`. This screenshiot shows the ip address `10.0.2.15` when on `nat`:

```
root@linux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:f5:74:cf
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fef5:74cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2352 (2.2 KiB)  TX bytes:1988 (1.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```



By shutting down the network interface and enabling it again, we force Debian to renew an ip address from the bridged network.

```
root@linux:~# # do not run ifdown while connected over ssh!
root@linux:~# ifdown eth0
Killed old client process
Internet Systems Consortium DHCP Client 4.3.1
Copyright 2004-2014 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/08:00:27:f5:74:cf
Sending on   LPF/eth0/08:00:27:f5:74:cf
Sending on   Socket/fallback
DHCPRELEASE on eth0 to 10.0.2.2 port 67
root@linux:~# # now enable bridge in virtualbox settings
root@linux:~# ifup eth0
Internet Systems Consortium DHCP Client 4.3.1
Copyright 2004-2014 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/08:00:27:f5:74:cf
Sending on   LPF/eth0/08:00:27:f5:74:cf
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 8
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 8
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPOFFER from 192.168.1.42
DHCPACK from 192.168.1.42
bound to 192.168.1.111 -- renewal in 2938 seconds.
root@linux:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:f5:74:cf
          inet addr:192.168.1.111  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fef5:74cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3156 (3.0 KiB)  TX bytes:3722 (3.6 KiB)
root@linux:~#
```

Here is an example of ssh to this freshly installed computer. Note that Debian 8 has disabled remote root access, so i need to use the normal user account.

```
student@linux:~$ ssh paul@192.168.1.111
student@192.168.1.111's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
student@linux:~$
student@linux:~$ su -
Password:
root@linux:~#
```

TODO: putty screenshot here...

4.4. setting the hostname

The hostname of the server is asked during installation, so there is no need to configure this manually.

```
root@linux:~# hostname
server42
root@linux:~# cat /etc/hostname
server42
root@linux:~# dnsdomainname
paul.local
root@linux:~# grep server42 /etc/hosts
127.0.1.1       server42.paul.local     server42
root@linux:~#
```

4.5. adding a static ip address

This example shows how to add a static ip address to your server.

You can use ifconfig to set a static address that is active until the next reboot (or until the next ifdown).

a

```
root@linux:~# ifconfig eth0:0 10.104.33.39
```

Adding a couple of lines to the /etc/network/interfaces file to enable an extra ip address forever.

```
root@linux:~# vi /etc/network/interfaces
root@linux:~# tail -4 /etc/network/interfaces
auto eth0:0
iface eth0:0 inet static
address 10.104.33.39
netmask 255.255.0.0
```

```
root@linux:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:f5:74:cf
          inet addr:192.168.1.111  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fef5:74cf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:528 errors:0 dropped:0 overruns:0 frame:0
          TX packets:333 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:45429 (44.3 KiB)  TX bytes:48763 (47.6 KiB)

eth0:0    Link encap:Ethernet  HWaddr 08:00:27:f5:74:cf
          inet addr:10.104.33.39  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@linux:~#
```

4.6. Debian package management

To get all information about the newest packages form the online repository:

```
root@linux:~# aptitude update
Get: 1 http://ftp.be.debian.org jessie InRelease [191 kB]
Get: 2 http://security.debian.org jessie/updates InRelease [84.1 kB]
Get: 3 http://ftp.be.debian.org jessie-updates InRelease [117 kB]
Get: 4 http://ftp.be.debian.org jessie-backports InRelease [118 kB]
Get: 5 http://security.debian.org jessie/updates/main Sources [14 B]
Get: 6 http://ftp.be.debian.org jessie/main Sources/DiffIndex [7,876 B]
... (output truncated)
```

To download and apply all updates for all installed packages:

```
root@linux:~# aptitude upgrade
Resolving dependencies...
The following NEW packages will be installed:
    firmware-linux-free{a}  irqbalance{a}  libnuma1{a}  linux-image-3.16.0-4-
amd64{a}
The following packages will be upgraded:
  busybox file libc-bin libc6 libexpat1 libmagic1 libpaper-utils libpaper1 libsqlite3-
0
  linux-image-amd64 locales multiarch-support
12 packages upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 44.9 MB of archives. After unpacking 161 MB will be used.
Do you want to continue? [Y/n/?]
... (output truncated)
```

To install new software (`vim` and `tmux` in this example):

```
root@linux:~# aptitude install vim tmux
The following NEW packages will be installed:
  tmux vim vim-runtime{a}
0 packages upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 6,243 kB of archives. After unpacking 29.0 MB will be used.
Do you want to continue? [Y/n/?]
Get: 1 http://ftp.be.debian.org/debian/ jessie/main tmux amd64 1.9-6 [245 kB]
Get: 2 http://ftp.be.debian.org/debian/ jessie/main vim-runtime all 2:7.4.488-
1 [5,046 kB]
Get:  3  http://ftp.be.debian.org/debian/  jessie/main  vim  amd64  2:7.4.488-
1 [952 kB]
```

Refer to the `package management` chapter in LinuxAdm.pdf for more information.

5. installing CentOS 8

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

This module is a step by step demonstration of an actual installation of `CentOS 8`.

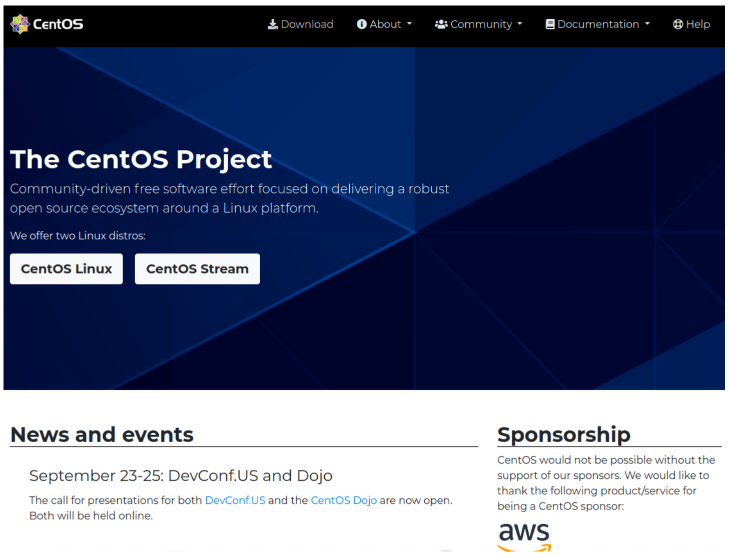
We start by downloading an image from the internet and install `CentOS 8` as a virtual machine in `Virtualbox`. We will also do some basic configuration of this new machine like setting an `ip address` and fixing a `hostname`.

This procedure should be very similar for other versions of `CentOS`, and also for distributions like `RHEL` (Red Hat Enterprise Linux) or `Fedora`. This procedure can also be helpful if you are using another virtualization solution.

5.1. download a CentOS 7 image

This demonstration uses a laptop computer with `Virtualbox` to install `CentOS 7` as a virtual machine. The first task is to download an `.iso` image of `CentOS 7`.

The `CentOS 7` website looks like this today (November 2014). They change the look regularly, so it may look different when you visit it.



The CentOS Project

Community-driven free software effort focused on delivering a robust open source ecosystem around a Linux platform.

We offer two Linux distros:

News and events

September 23-25: DevConf.US and Dojo

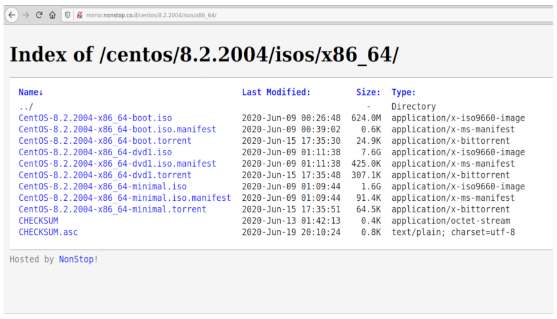
The call for presentations for both DevConf.US and the CentOS Dojo are now open. Both will be held online.

Sponsorship

CentOS would not be possible without the support of our sponsors. We would like to thank the following product/service for being a CentOS sponsor:

You can download a full DVD, which allows for an off line installation of a graphical `CentOS 7` desktop. You can select this because it should be easy and complete, and should get you started with a working `CentOS 7` virtual machine.

But I clicked instead on 'alternative downloads', selected `CentOS 7` and `x86_64` and ended up on a `mirror list`. Each mirror is a server that contains copies of `CentOS 7` media. I selected a Belgian mirror because I currently am in Belgium.



```
mirror.nonstop.co.il/centos/8.2.2004/isos/x86_64/

Index of /centos/8.2.2004/isos/x86_64/

Name↓                                      Last Modified:         Size:  Type:
../                                                                   -  Directory
CentOS-8.2.2004-x86_64-boot.iso            2020-Jun-09 00:26:48  624.0M  application/x-iso9660-image
CentOS-8.2.2004-x86_64-boot.iso.manifest   2020-Jun-09 00:39:02    0.6K  application/x-ms-manifest
CentOS-8.2.2004-x86_64-boot.torrent        2020-Jun-15 17:35:30   24.9K  application/x-bittorrent
CentOS-8.2.2004-x86_64-dvd1.iso            2020-Jun-09 01:11:38    7.6G  application/x-iso9660-image
CentOS-8.2.2004-x86_64-dvd1.iso.manifest   2020-Jun-09 01:11:38  425.0K  application/x-ms-manifest
CentOS-8.2.2004-x86_64-dvd1.torrent        2020-Jun-15 17:35:48  307.1K  application/x-bittorrent
CentOS-8.2.2004-x86_64-minimal.iso         2020-Jun-09 01:09:44    1.6G  application/x-iso9660-image
CentOS-8.2.2004-x86_64-minimal.iso.manifest 2020-Jun-09 01:09:44  91.4K  application/x-ms-manifest
CentOS-8.2.2004-x86_64-minimal.torrent     2020-Jun-15 17:35:51   64.5K  application/x-bittorrent
CHECKSUM                                   2020-Jun-13 01:42:13    0.4K  application/octet-stream
CHECKSUM.asc                               2020-Jun-19 20:10:24    0.8K  text/plain; charset=utf-8

Hosted by NonStop!
```

There is again the option for full DVD's and more. This demonstration will use the `minimal` .iso file, because it is much smaller in size. The download takes a couple of minutes.

Verify the size of the file after download to make sure it is complete. Probably a right click on the file and selecting 'properties' (if you use Windows or Mac OSX).

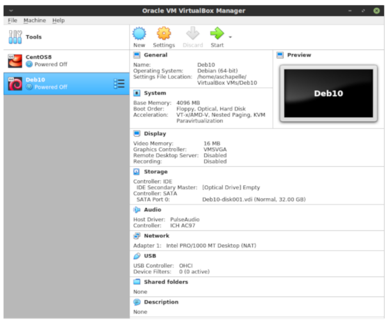
I use Linux on the laptop already:

```
student@linux:~$ ls -lh CentOS-7.0-1406-x86_64-Minimal.iso
-rw-r--r-- 1 paul paul 566M Nov  1 14:45 CentOS-7.0-1406-x86_64-Minimal.iso
```

Do not worry if you do no understand the above command. Just try to make sure that the size of this file is the same as the size that is mentioned on the `CentOS 7` website.

5.2. Virtualbox

This screenshot shows up when I start Virtualbox. I already have four virtual machines, you might have none.



General
Name: Deb10
Operating System: Debian (64-bit)
Settings File Location: /home/aschapelle/VirtualBox VMs/Deb10

System
Base Memory: 4096 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Deb10-disk001.vdi (Normal, 32.00 GB)

Audio
Host Driver: PulseAudio
Controller: ICH AC97

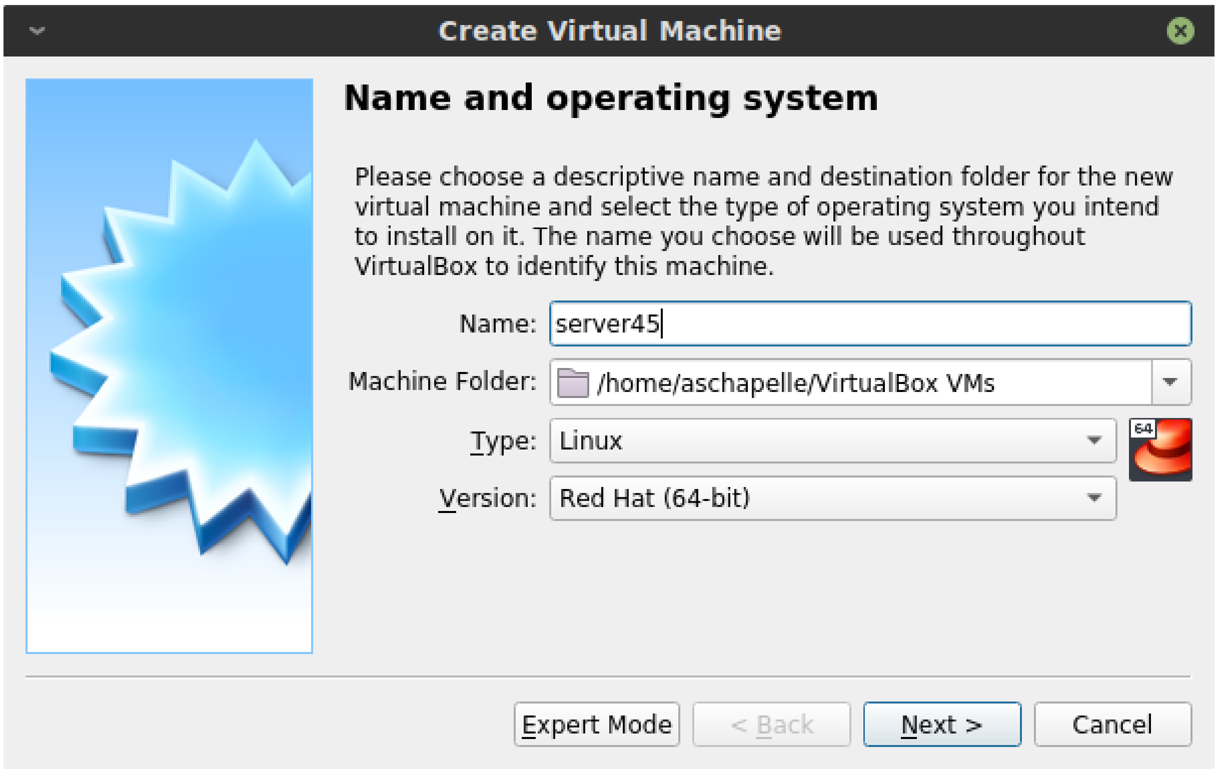
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB
USB Controller: OHCI
Device Filters: 0 (0 active)

Shared folders
None

Description
None

Below are the steps for creating a new virtual machine. Start by clicking `New` and give your machine a name (I chose `server33`). Click `Next`.



Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

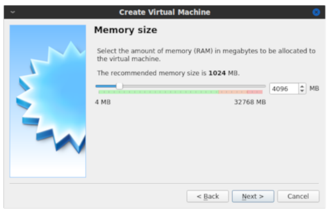
Name: server45

Machine Folder: /home/aschapelle/VirtualBox VMs

Type: Linux

Version: Red Hat (64-bit)

A Linux computer without graphical interface will run fine on `half a gigabyte` of RAM.



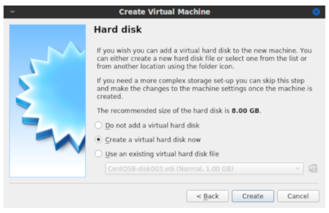
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

4096 MB

A Linux virtual machine will need a `virtual hard drive`.



Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

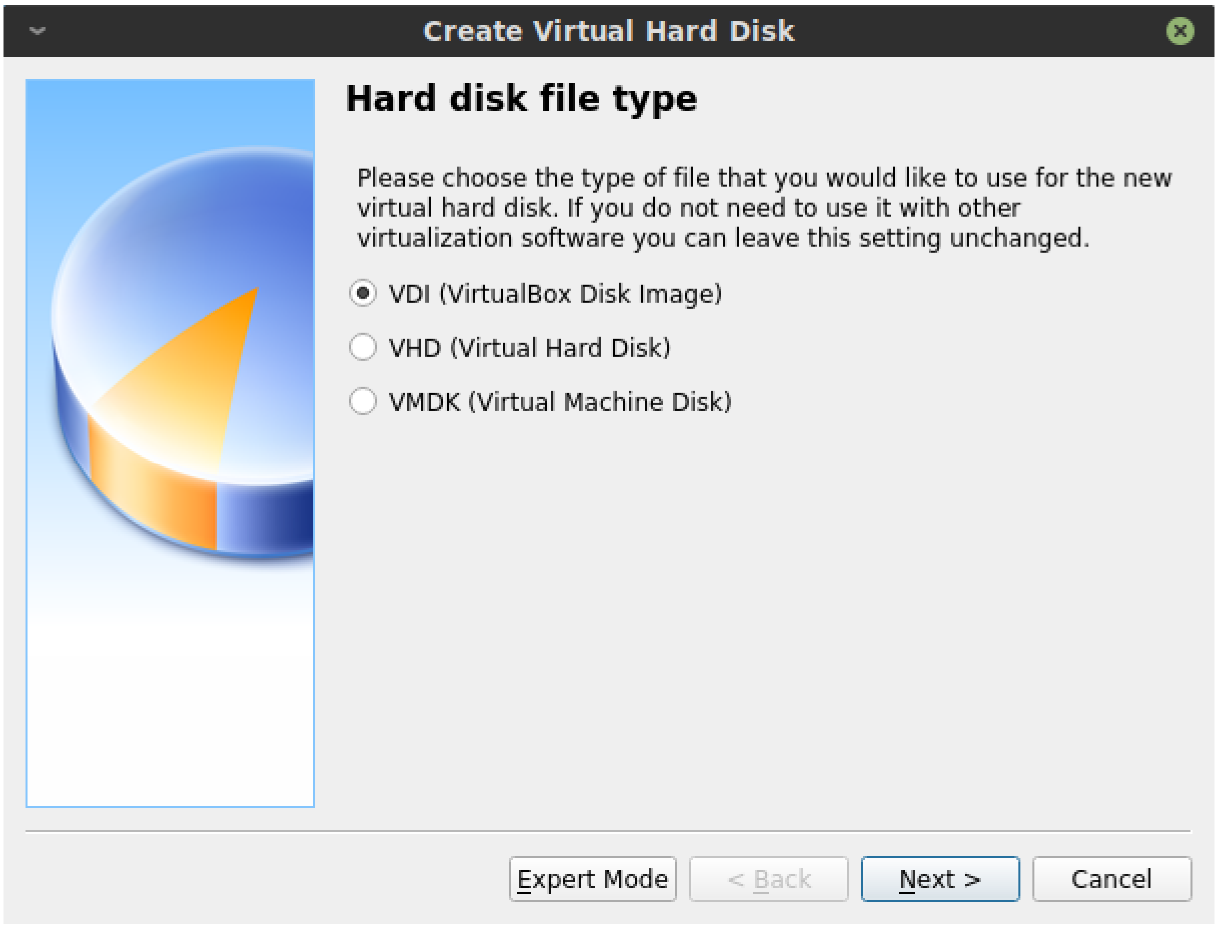
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

Do not add a virtual hard disk [✓ Create a virtual hard disk now] Use an existing virtual hard disk file

CentOS8-disk003.vdi (Normal, 1.00 GB)

Any format will do for our purpose, so I left the default `vdi`.

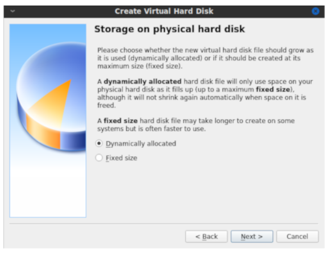

Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

[✓ VDI (VirtualBox Disk Image)]
VHD (Virtual Hard Disk)
VMDK (Virtual Machine Disk)

The default `dynamically allocated` type will save disk space (until we fill the virtual disk up to 100 percent). It makes the virtual machine a bit slower than `fixed size`, but the `fixed size` speed improvement is not worth it for our purpose.


Create Virtual Hard Disk

Storage on physical hard disk

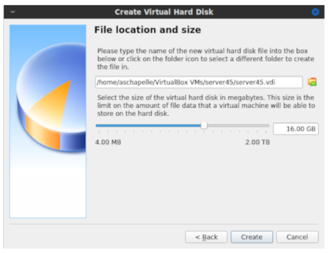
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

[✓ Dynamically allocated]
Fixed size

The name of the virtual disk file on the host computer will be `server33.vdi` in my case (I left it default and it uses the vm name). Also 16 GB should be enough to practice Linux. The file will stay much smaller than 16GB, unless you copy a lot of files to the virtual machine.


Create Virtual Hard Disk

File location and size

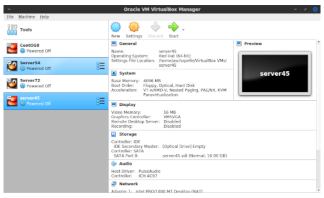
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

/home/aschapelle/VirtualBox VMs/server45/server45.vdi

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

16.00 GB

You should now be back to the start screen of `Virtualbox`. If all went well, then you should see the machine you just created in the list.


Oracle VM VirtualBox Manager

File Machine Help

Tools

CentOS8 Powered Off
Server54 Powered Off
Server72 Powered Off
server45 Powered Off

New Settings Discard Start

General
Name: server45
Operating System: Red Hat (64-bit)
Settings File Location: /home/aschapelle/VirtualBox VMs/server45

Preview
server45

System
Base Memory: 4096 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

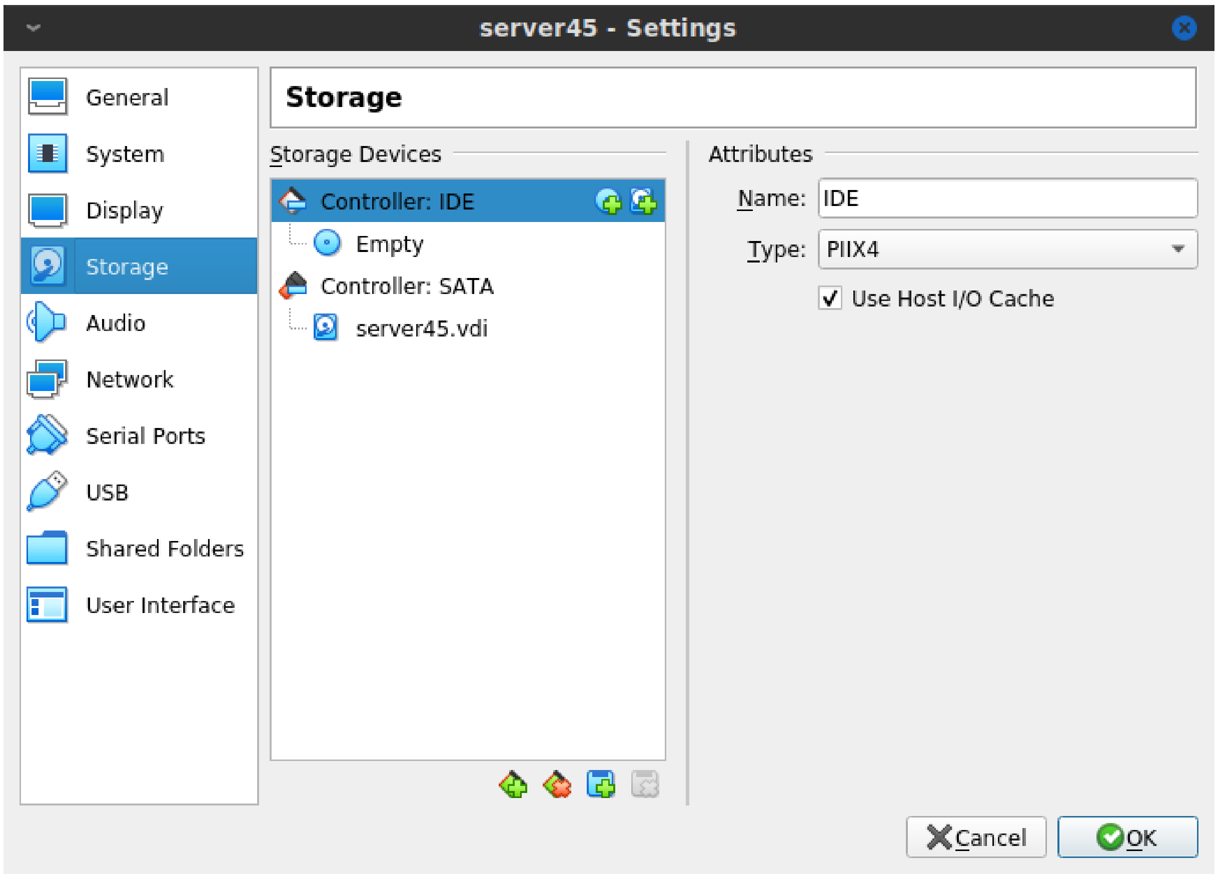
Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: server45.vdi (Normal, 16.00 GB)

Audio
Host Driver: PulseAudio
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

After finishing the setup, we go into the `Settings` of our virtual machine and attach the `.iso` file we downloaded before. Below is the default screenshot.


server45 - Settings

General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

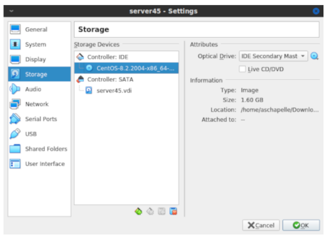
Storage

Storage Devices
Controller: IDE
Empty
Controller: SATA
server45.vdi

Attributes
Name: IDE
Type: PIIX4
✓ Use Host I/O Cache

Cancel OK

This is a screenshot with the `.iso` file properly attached.


server45 - Settings

General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface

Storage

Storage Devices
Controller: IDE
CentOS-8.2.2004-x86_64-...
Controller: SATA
server45.vdi

Attributes
Optical Drive: IDE Secondary Mast
Live CD/DVD

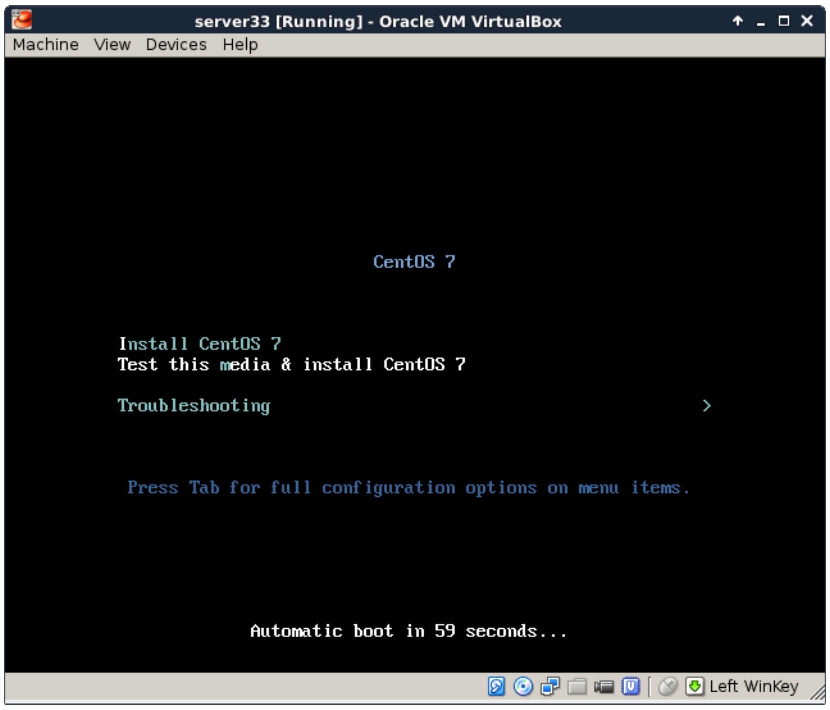
Information
Type: Image
Size: 1.60 GB
Location: /home/aschapelle/Downlo...
Attached to: --

Cancel OK

5.3. CentOS 7 installing

The screenshots below will show every step from starting the virtual machine for the first time (with the .iso file attached) until the first logon.

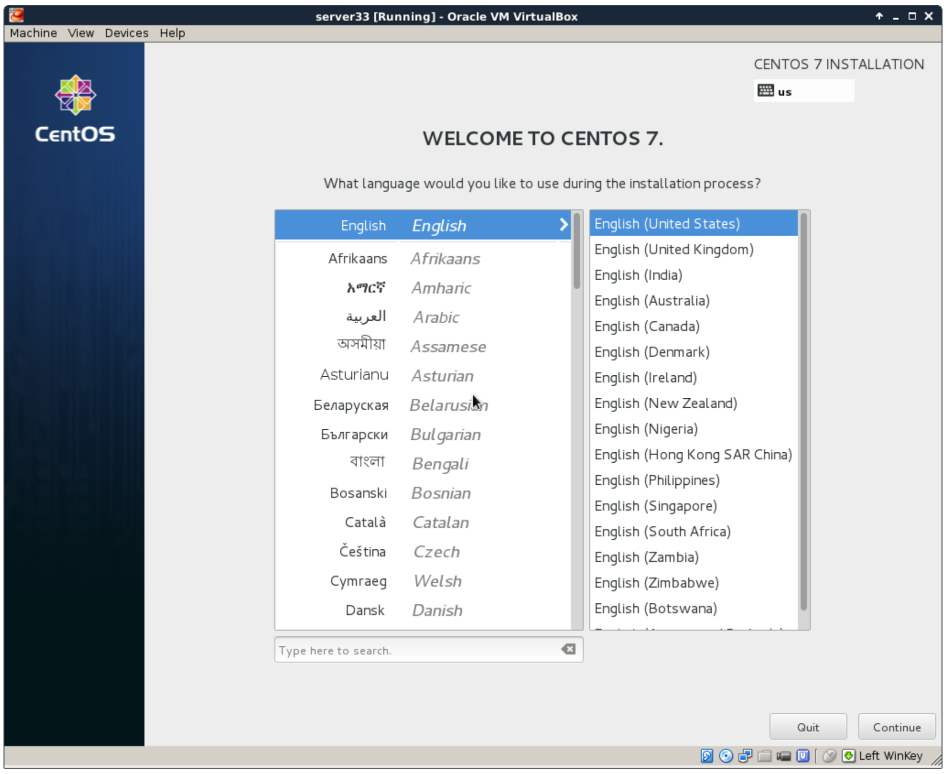
You should see this when booting, otherwise verify the attachment of the .iso file form the previous steps. Select `Test this media and install CentOS 7`.


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
CentOS 7

Install CentOS 7
Test this media & install CentOS 7

Troubleshooting                              >

Press Tab for full configuration options on menu items.

Automatic boot in 59 seconds...
```

Carefully select the language in which you want your `CentOS`. I always install operating systems in English, even though my native language is not English.

Also select the right keyboard, mine is a US qwerty, but yours may be different.

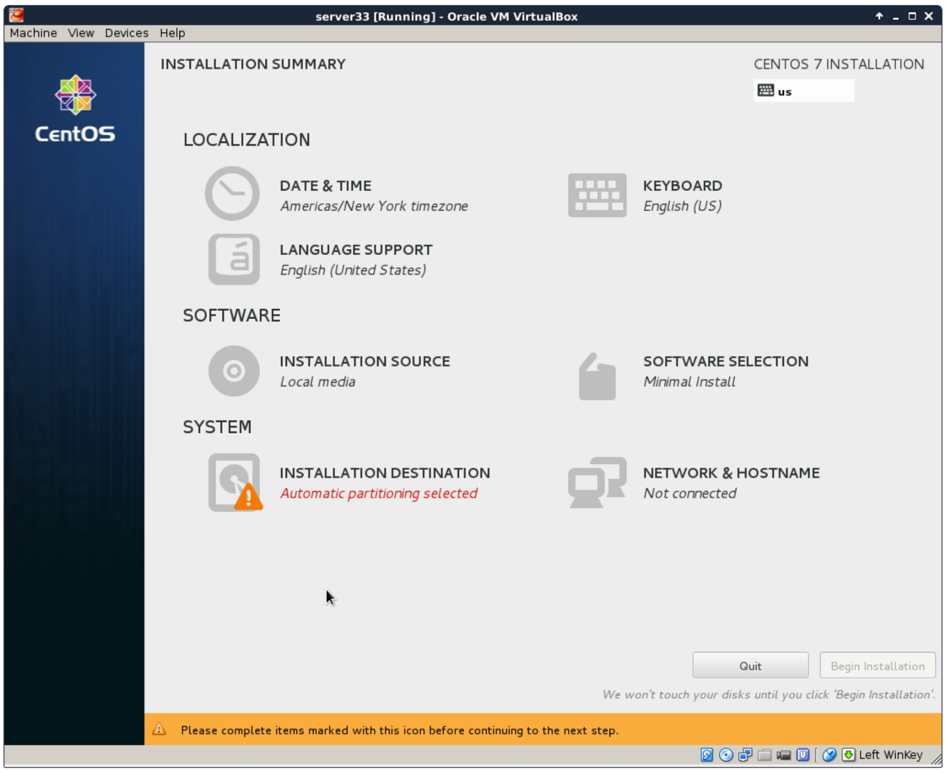


WELCOME TO CENTOS 7.

What language would you like to use during the installation process?

English	*English*	English (United States)
Afrikaans	*Afrikaans*	English (United Kingdom)
አማርኛ	*Amharic*	English (India)
العربية	*Arabic*	English (Australia)
অসমীয়া	*Assamese*	English (Canada)
Asturianu	*Asturian*	English (Denmark)
Беларуская	*Belarusian*	English (Ireland)
Български	*Bulgarian*	English (New Zealand)
বাংলা	*Bengali*	English (Nigeria)
Bosanski	*Bosnian*	English (Hong Kong SAR China)
Català	*Catalan*	English (Philippines)
Čeština	*Czech*	English (Singapore)
Cymraeg	*Welsh*	English (South Africa)
Dansk	*Danish*	English (Zambia)
		English (Zimbabwe)
		English (Botswana)

You should arrive at a summary page (with one or more warnings).


server33 [Running] - Oracle VM VirtualBox

INSTALLATION SUMMARY

CENTOS 7 INSTALLATION

LOCALIZATION

DATE & TIME
Americas/New York timezone

KEYBOARD
English (US)

LANGUAGE SUPPORT
English (United States)

SOFTWARE

INSTALLATION SOURCE
Local media

SOFTWARE SELECTION
Minimal Install

SYSTEM

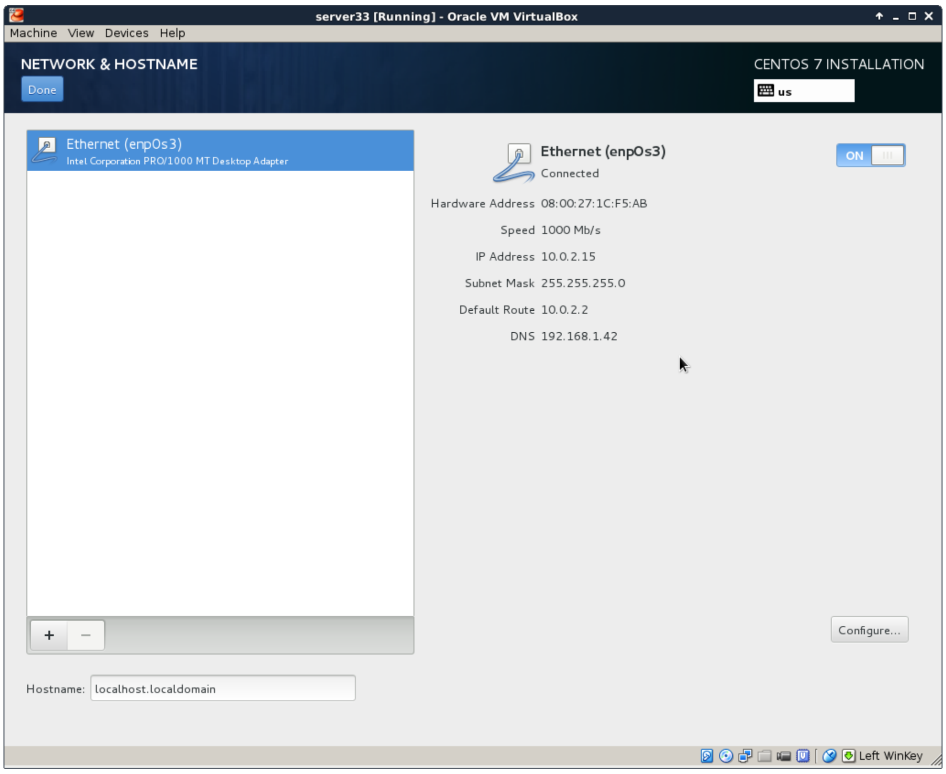
INSTALLATION DESTINATION
Automatic partitioning selected

NETWORK & HOSTNAME
Not connected

We won't touch your disks until you click 'Begin Installation'.

Please complete items marked with this icon before continuing to the next step.

Start by configuring the network. During this demonstration I had a DHCP server running at 192.168.1.42, yours is probably different. Ask someone (a network administator ?) for help if this step fails.


server33 [Running] - Oracle VM VirtualBox

NETWORK & HOSTNAME

CENTOS 7 INSTALLATION

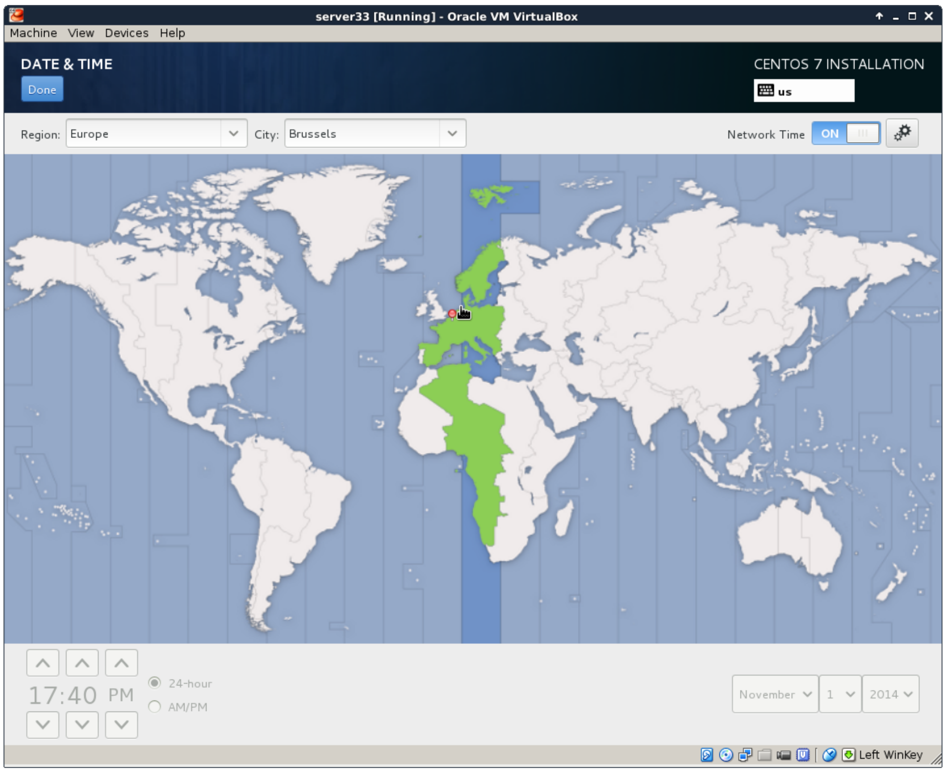
Ethernet (enp0s3)
Intel Corporation PRO/1000 MT Desktop Adapter

Ethernet (enp0s3)
Connected

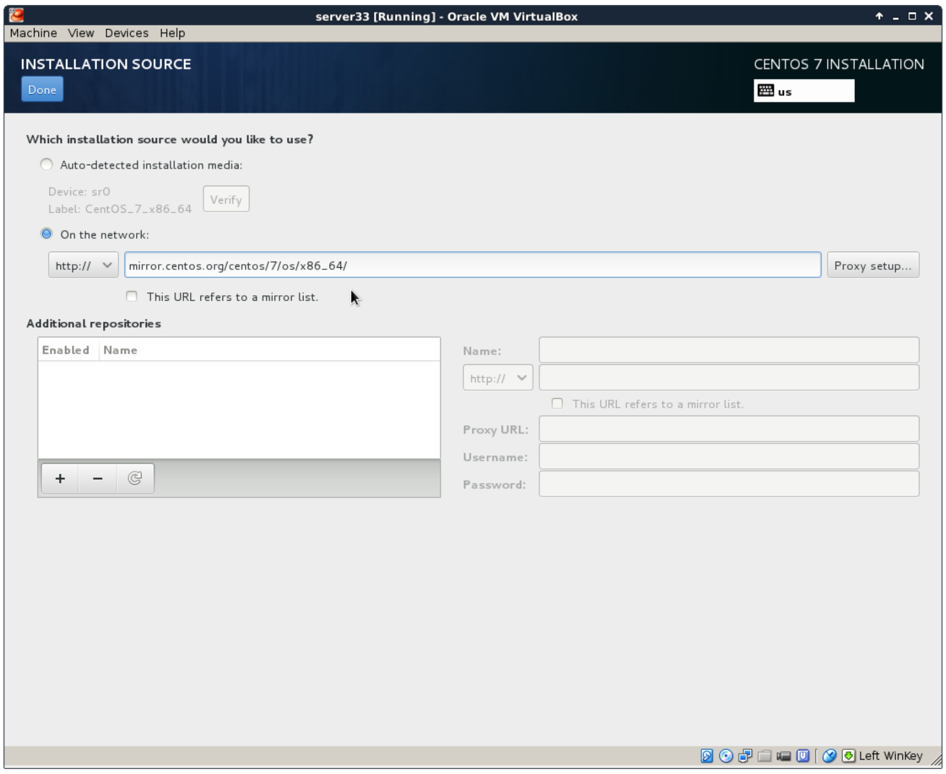
Hardware Address	08:00:27:1C:F5:AB
Speed	1000 Mb/s
IP Address	10.0.2.15
Subnet Mask	255.255.255.0
Default Route	10.0.2.2
DNS	192.168.1.42

Hostname: localhost.localdomain

Select your time zone, and activate `ntp`.



Choose a mirror that is close to you. If you can't find a local mirror, then you can copy the one from this screenshot (it is a general `CentOS` mirror).


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
INSTALLATION SOURCE
Done
CENTOS 7 INSTALLATION
us

Which installation source would you like to use?

Auto-detected installation media:

Device: sr0
Label: CentOS_7_x86_64 Verify

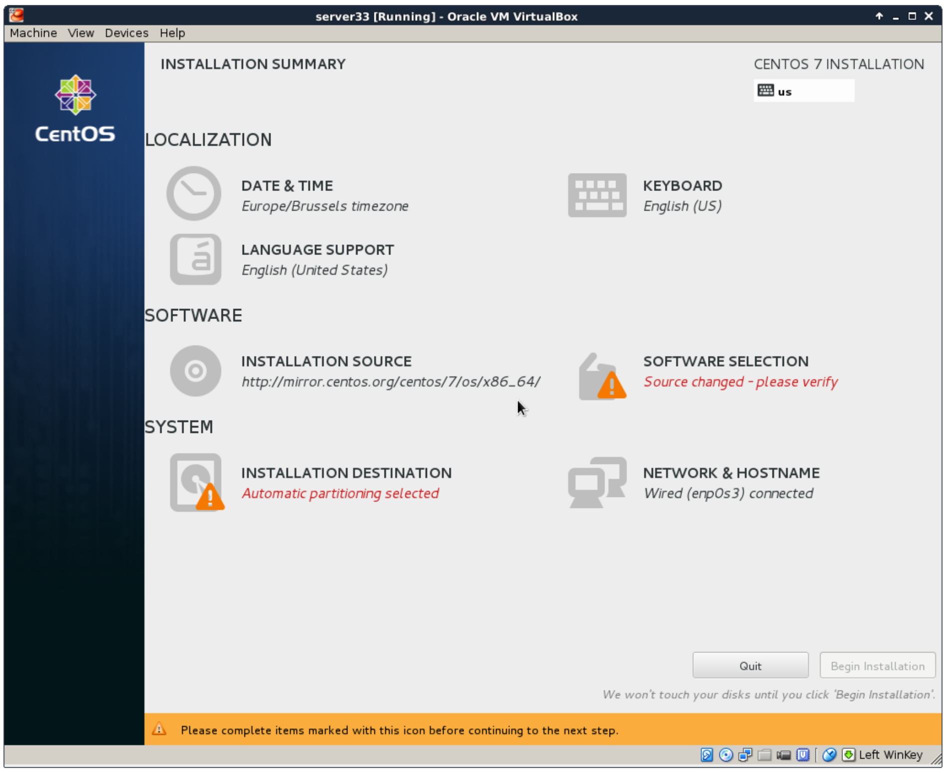
On the network:

http:// mirror.centos.org/centos/7/os/x86_64/ Proxy setup...

This URL refers to a mirror list.

Additional repositories

It can take a couple of seconds before the mirror is verified.


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
CentOS
Quit
Begin Installation
Left WinKey

INSTALLATION SUMMARY

CENTOS 7 INSTALLATION

us

LOCALIZATION

DATE & TIME
Europe/Brussels timezone

KEYBOARD
English (US)

LANGUAGE SUPPORT
English (United States)

SOFTWARE

INSTALLATION SOURCE
http://mirror.centos.org/centos/7/os/x86_64/

SOFTWARE SELECTION
Source changed - please verify

SYSTEM

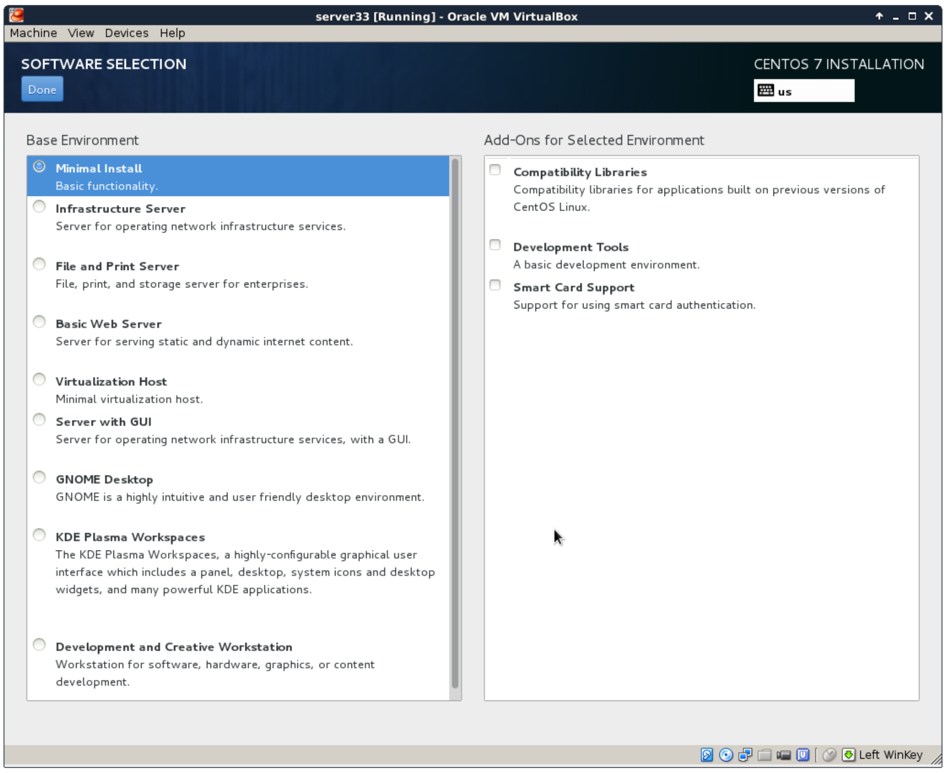
INSTALLATION DESTINATION
Automatic partitioning selected

NETWORK & HOSTNAME
Wired (enp0s3) connected

We won't touch your disks until you click 'Begin Installation'.

Please complete items marked with this icon before continuing to the next step.

I did not select any software here (because I want to show it all in this training).


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Done
Left WinKey

SOFTWARE SELECTION

CENTOS 7 INSTALLATION

us

Base Environment

[✓ Minimal Install]
Basic functionality.

Infrastructure Server
Server for operating network infrastructure services.

File and Print Server
File, print, and storage server for enterprises.

Basic Web Server
Server for serving static and dynamic internet content.

Virtualization Host
Minimal virtualization host.

Server with GUI
Server for operating network infrastructure services, with a GUI.

GNOME Desktop
GNOME is a highly intuitive and user friendly desktop environment.

KDE Plasma Workspaces
The KDE Plasma Workspaces, a highly-configurable graphical user interface which includes a panel, desktop, system icons and desktop widgets, and many powerful KDE applications.

Development and Creative Workstation
Workstation for software, hardware, graphics, or content development.

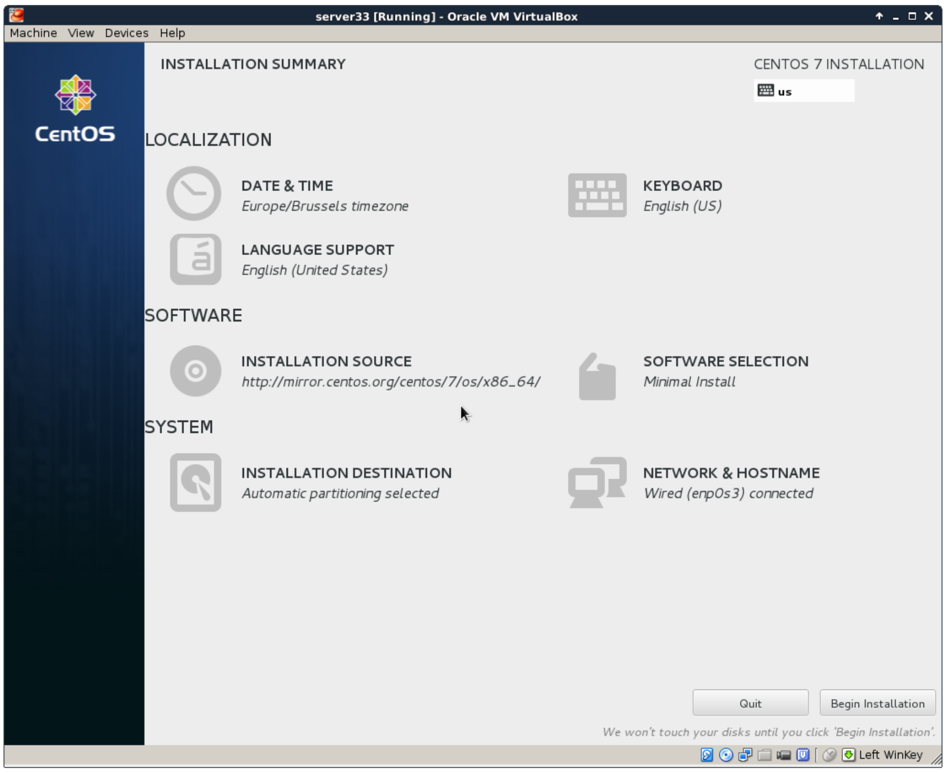
Add-Ons for Selected Environment

Compatibility Libraries
Compatibility libraries for applications built on previous versions of CentOS Linux.

Development Tools
A basic development environment.

Smart Card Support
Support for using smart card authentication.

After configuring network, location, software and all, you should be back on this page. Make sure there are no warnings anymore (and that you made the correct choice everywhere).


server33 [Running] - Oracle VM VirtualBox

Machine View Devices Help

INSTALLATION SUMMARY

CENTOS 7 INSTALLATION

us

LOCALIZATION

DATE & TIME
Europe/Brussels timezone

KEYBOARD
English (US)

LANGUAGE SUPPORT
English (United States)

SOFTWARE

INSTALLATION SOURCE
http://mirror.centos.org/centos/7/os/x86_64/

SOFTWARE SELECTION
Minimal Install

SYSTEM

INSTALLATION DESTINATION
Automatic partitioning selected

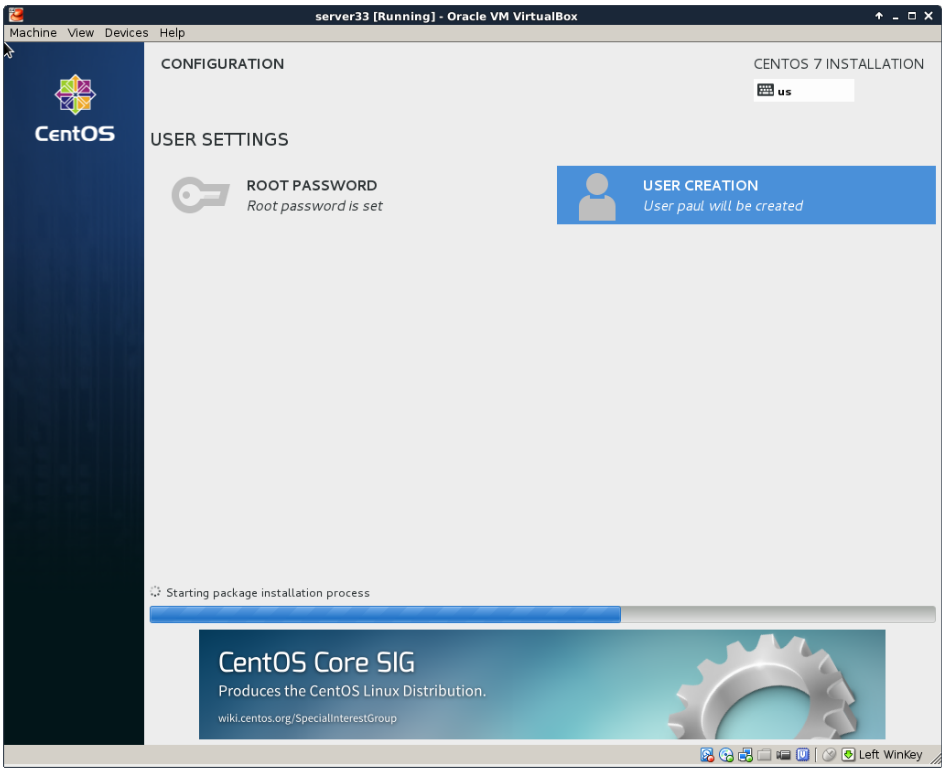
NETWORK & HOSTNAME
Wired (enp0s3) connected

Quit | Begin Installation

We won't touch your disks until you click 'Begin Installation'.

Left WinKey

You can enter a `root password` and create a `user account` while the installation is downloading from the internet. This is the longest step, it can take several minutes (or up to an hour if you have a slow internet connection).


server33 [Running] - Oracle VM VirtualBox

Machine View Devices Help

CONFIGURATION

CENTOS 7 INSTALLATION

us

USER SETTINGS

ROOT PASSWORD
Root password is set

USER CREATION
User paul will be created

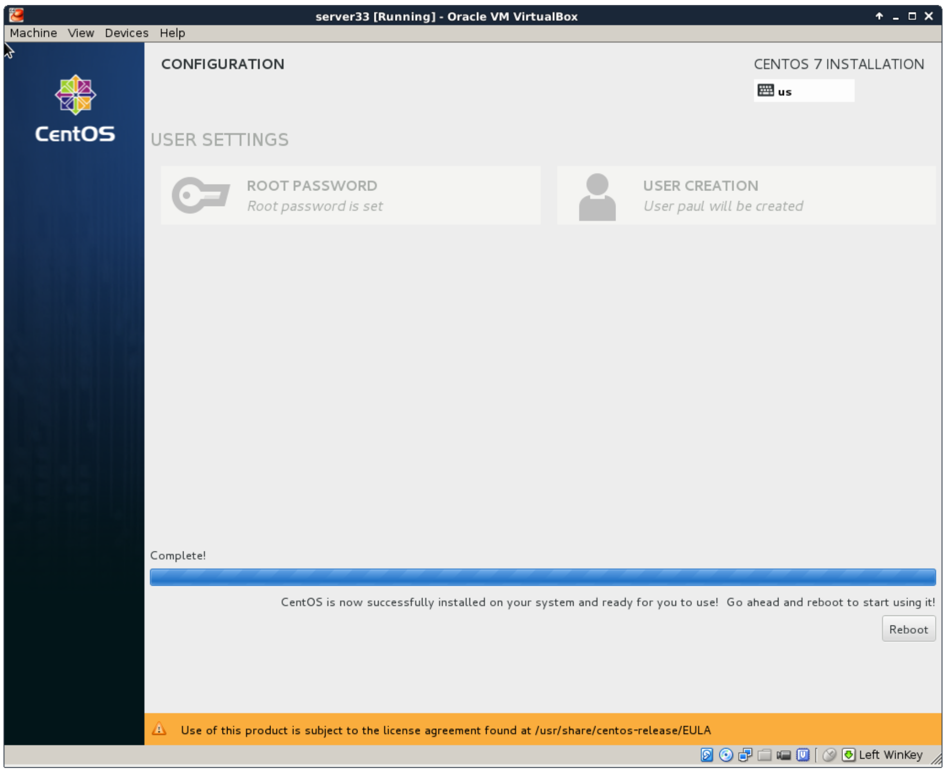
Starting package installation process

CentOS Core SIG
Produces the CentOS Linux Distribution.
wiki.centos.org/SpecialInterestGroup

Left WinKey

If you see this, then the installation was successful.

Time to reboot the computer and start `CentOS 7` for the first time.

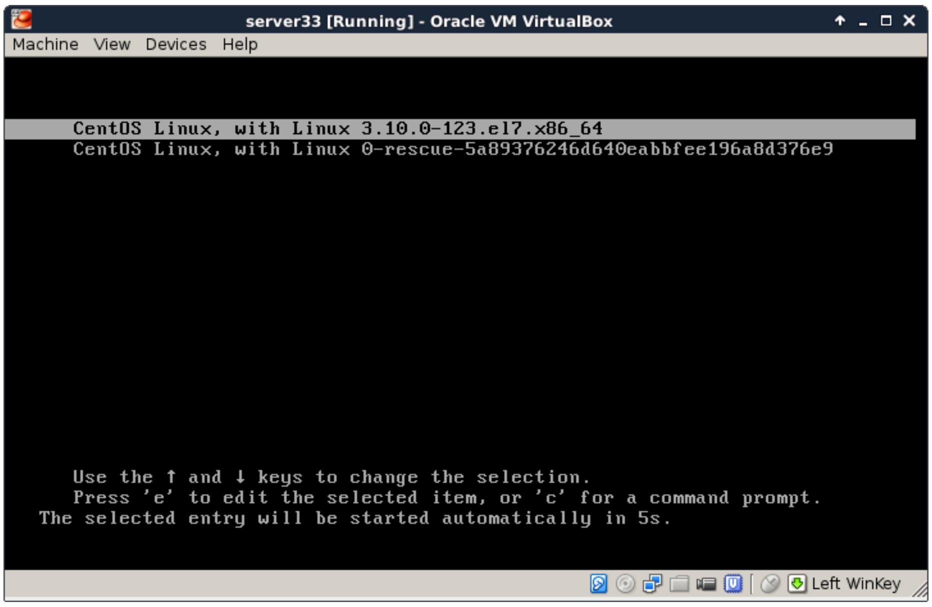


Complete!

CentOS is now successfully installed on your system and ready for you to use! Go ahead and reboot to start using it!

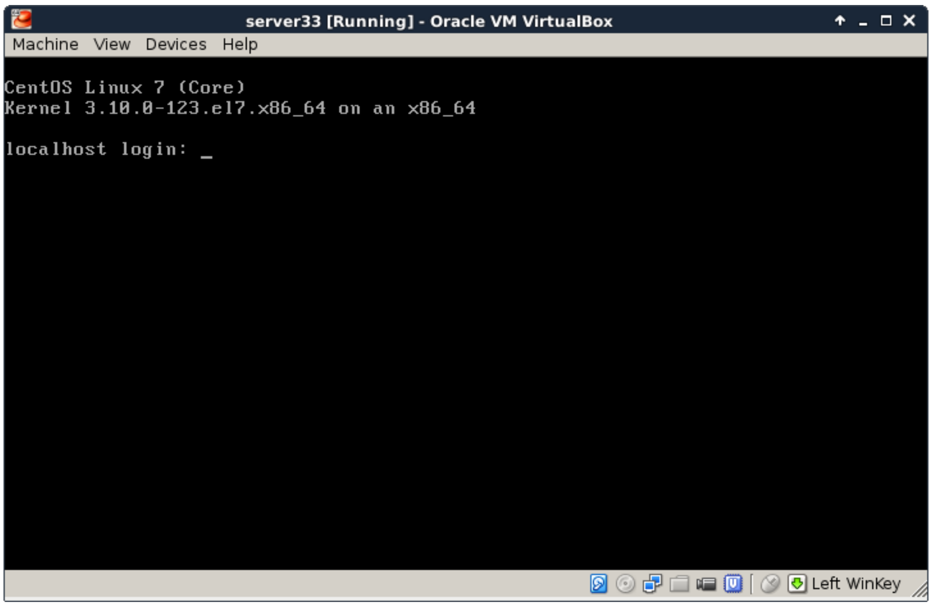
Use of this product is subject to the license agreement found at /usr/share/centos-release/EULA

This screen will appear briefly when the virtual machines starts. You don't have to do anything.


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

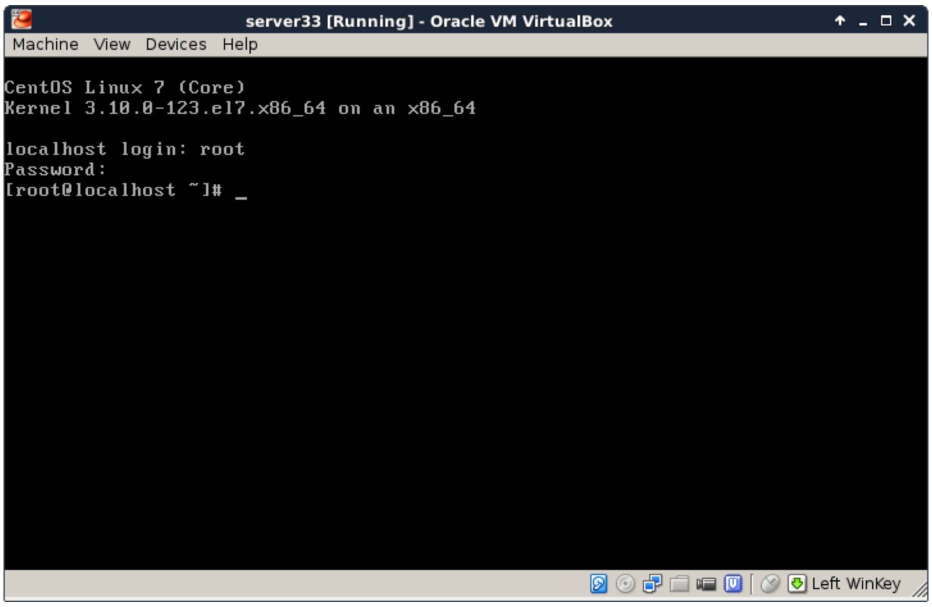
```
      CentOS Linux, with Linux 3.10.0-123.el7.x86_64
      CentOS Linux, with Linux 0-rescue-5a89376246d640eabbfee196a8d376e9









      Use the ↑ and ↓ keys to change the selection.
      Press 'e' to edit the selected item, or 'c' for a command prompt.
   The selected entry will be started automatically in 5s.
```

After a couple of seconds, you should see a logon screen. This is called a `tty` or a `getty`. Here you can type `root` as username. The `login process` will then ask your password (nothing will appear on screen when you type your password).


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
CentOS Linux 7 (Core)
Kernel 3.10.0-123.el7.x86_64 on an x86_64

localhost login: _
```

And this is what it looks like after logon. You are logged on to your own Linux machine, very good.


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
CentOS Linux 7 (Core)
Kernel 3.10.0-123.el7.x86_64 on an x86_64

localhost login: root
Password:
[root@localhost ~]# _
```

All subsequent screenshots will be text only, no images anymore.

For example this screenshot shows three commands being typed on my new CentOS 7 install.

```
[root@localhost ~]# who am i
root     pts/0        2014-11-01 22:14
[root@localhost ~]# hostname
localhost.localdomain
[root@localhost ~]# date
Sat Nov  1 22:14:37 CET 2014
```

When using `ssh` the same commands will give this screenshot:

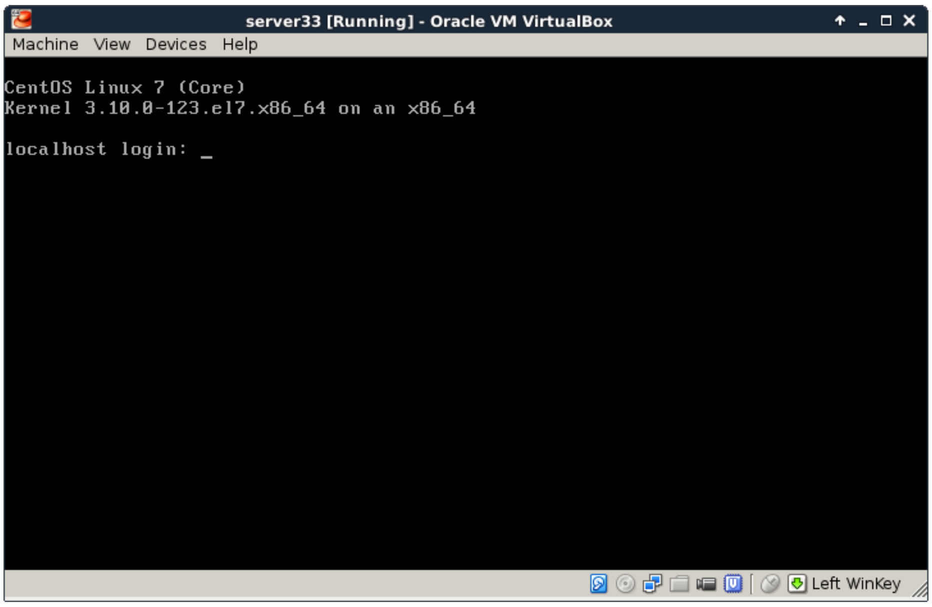
```
[root@localhost ~]# who am i
root     pts/0        2014-11-01 21:00 (192.168.1.35)
[root@localhost ~]# hostname
```

```
localhost.localdomain
[root@localhost ~]# date
Sat Nov  1 22:10:04 CET 2014
[root@localhost ~]#
```

If the last part is a bit too fast, take a look at the next topic `CentOS 7 first logon`.

5.4. CentOS 7 first logon

All you have to log on, after finishing the installation, is this screen in Virtualbox.


server33 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Left WinKey

```
CentOS Linux 7 (Core)
Kernel 3.10.0-123.el7.x86_64 on an x86_64

localhost login: _
```

This is workable to learn Linux, and you will be able to practice a lot. But there are more ways to access your virtual machine, the next chapters discuss some of these and will also introduce some basic system configuration.

5.4.1. setting the hostname

Setting the hostname is a simple as changing the `/etc/hostname` file. As you can see here, it is set to `localhost.localdomain` by default.

```
[root@localhost ~]# cat /etc/hostname
localhost.localdomain
```

You could do `echo server33.netsec.local > /etc/hostname` followed by a `reboot`. But there is also the new `CentOS 7` way of setting a new hostname.

```
[root@localhost ~]# nmtui
```

The above command will give you a menu to choose from with a `set system hostname` option. Using this `nmtui` option will edit the `/etc/hostname` file for you.

```
[root@localhost ~]# cat /etc/hostname
server33.netsec.local
[root@localhost ~]# hostname
server33.netsec.local
[root@localhost ~]# dnsdomainname
netsec.local
```

For some reason the documentation on the `centos.org` and `docs.redhat.com` websites tell you to also execute this command:

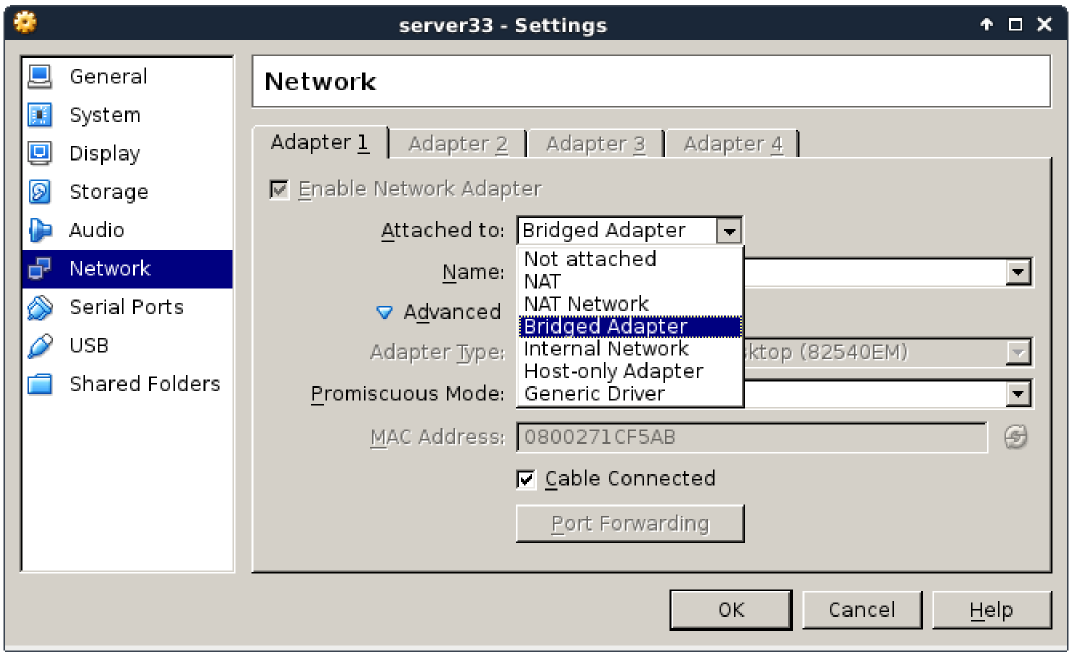
```
[root@localhost ~]# systemctl restart systemd-hostnamed
```

5.5. Virtualbox network interface

By default `Virtualbox` will connect your virtual machine over a `nat` interface. This will show up as a 10.0.2.15 (or similar).

```
[root@server33 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast s\
tate UP qlen 1000
    link/ether 08:00:27:1c:f5:ab brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 86399sec preferred_lft 86399sec
    inet6 fe80::a00:27ff:fe1c:f5ab/64 scope link
       valid_lft forever preferred_lft forever
```

You can change this to `bridge` (over your wi-fi or over the ethernet cable) and thus make it appear as if your virtual machine is directly on your local network (receiving an ip address from your real dhcp server).


server33 - Settings

Network

Adapter 1 | Adapter 2 | Adapter 3 | Adapter 4

☑ Enable Network Adapter

Attached to: Bridged Adapter

Name:

Advanced

Adapter Type: ...ktop (82540EM)

Promiscuous Mode:

MAC Address: 0800271CF5AB

☑ Cable Connected

You can make this change while the vm is running, provided that you execute this command:

```
[root@server33 ~]# systemctl restart network
[root@server33 ~]# ip a s dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast s\
tate UP qlen 1000
    link/ether 08:00:27:1c:f5:ab brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.110/24 brd 192.168.1.255 scope global dynamic enp0s3
       valid_lft 7199sec preferred_lft 7199sec
    inet6 fe80::a00:27ff:fe1c:f5ab/64 scope link
       valid_lft forever preferred_lft forever
[root@server33 ~]#
```

5.6. configuring the network

The new way of changing network configuration is through the `nmtui` tool. If you want to manually play with the files in `/etc/sysconfig/network-scripts` then you will first need to verify (and disable) `NetworkManager` on that interface.

Verify whether an interface is controlled by `NetworkManager` using the `nmcli` command (connected means managed bu NM).

```
[root@server33 ~]# nmcli dev status
DEVICE  TYPE      STATE      CONNECTION
enp0s3  ethernet  connected  enp0s3
lo      loopback  unmanaged  --
```

Disable `NetworkManager` on an interface (enp0s3 in this case):

```
echo 'NM_CONTROLLED=no' >> /etc/sysconfig/network-scripts/ifcfg-enp0s3
```

You can restart the network without a reboot like this:

```
[root@server33 ~]# systemctl restart network
```

Also, forget `ifconfig` and instead use `ip a`.

```
[root@server33 ~]# ip a s dev enp0s3 | grep inet
    inet 192.168.1.110/24 brd 192.168.1.255 scope global dynamic enp0s3
    inet6 fe80::a00:27ff:fe1c:f5ab/64 scope link
[root@server33 ~]#
```

5.7. adding one static ip address

This example shows how to add one static ip address to your computer.

```
[root@server33 ~]# nmtui edit enp0s3
```

In this interface leave the IPv4 configuration to automatic, and add an ip address just below.

```
          IPv4 CONFIGURATION <Automatic>                         <Hide>
          Addresses 10.104.33.32/16__________ <Remove>
```

Execute this command after exiting `nmtui`.

```
[root@server33 ~]# systemctl restart network
```

And verify with `ip` (not with `ifconfig`):

```
[root@server33 ~]# ip a s dev enp0s3 | grep inet
    inet 192.168.1.110/24 brd 192.168.1.255 scope global dynamic enp0s3
    inet 10.104.33.32/16 brd 10.104.255.255 scope global enp0s3
    inet6 fe80::a00:27ff:fe1c:f5ab/64 scope link
[root@server33 ~]#
```

5.8. package management

Even with a network install, `CentOS 7` did not install the latest version of some packages. Luckily there is only one command to run (as root). This can take a while.

```
[root@server33 ~]# yum update
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.weepeetelecom.be
 * extras: centos.weepeetelecom.be
 * updates: centos.weepeetelecom.be
Resolving Dependencies
--> Running transaction check
---> Package NetworkManager.x86_64 1:0.9.9.1-13.git20140326.4dba720.el7 \
will be updated
... (output truncated)
```

You can also use `yum` to install one or more packages. Do not forget to run `yum update` from time to time.

```
[root@server33 ~]# yum update -y && yum install vim -y
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.weepeetelecom.be
... (output truncated)
```

Refer to the package management chapter for more information on installing and removing packages.

5.9. logon from Linux and MacOSX

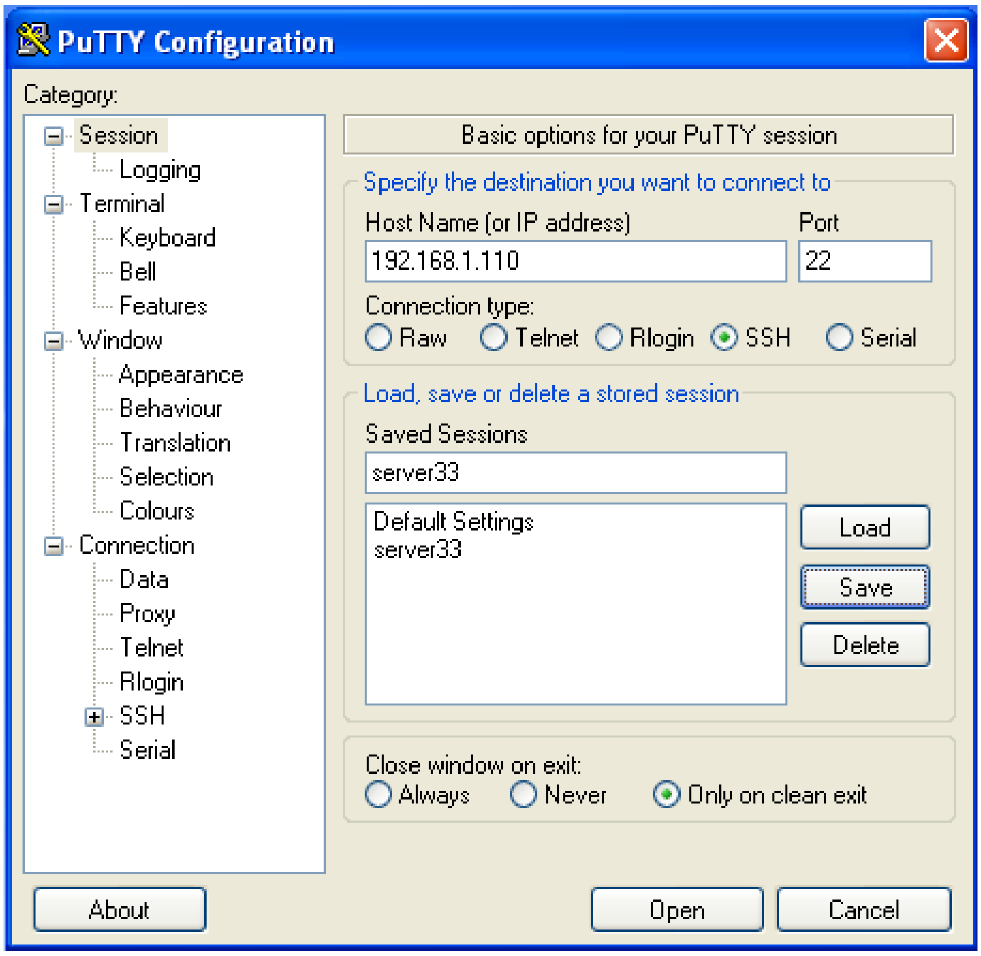
You can now open a terminal on Linux or MacOSX and use `ssh` to log on to your virtual machine.

```
student@linux:~$ ssh root@192.168.1.110
root@192.168.1.110's password:
Last login: Sun Nov  2 11:53:57 2014
[root@server33 ~]# hostname
server33.netsec.local
[root@server33 ~]#
```

5.10. logon from MS Windows

There is no ssh installed on MS Windows, but you can download putty.exe from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html (just Google it).

Use putty.exe as shown in this screenshot (I saved the ip address by giving it a name 'server33' and presing the 'save' button).


PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.1.110

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

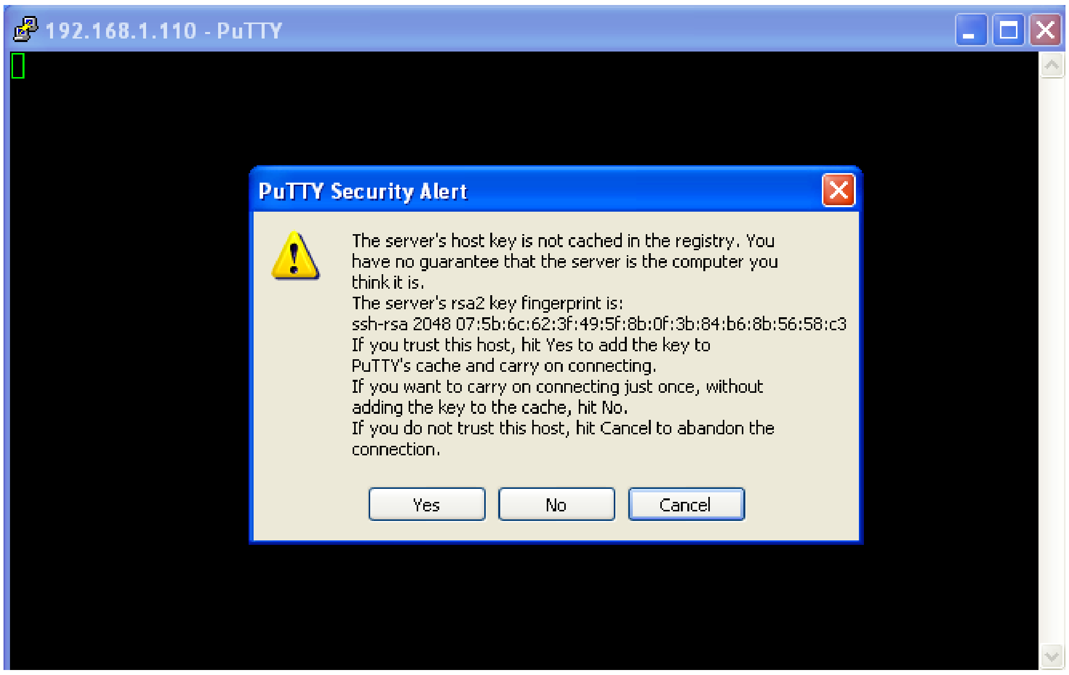
Load, save or delete a stored session

Saved Sessions: server33

Default Settings
server33

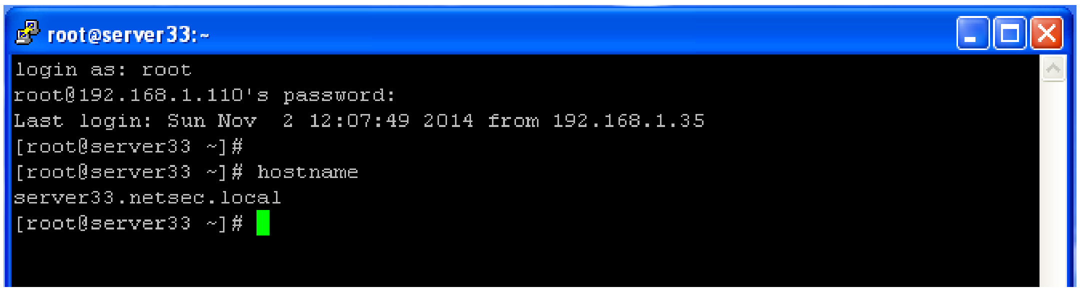
Close window on exit: Always Never [✓ Only on clean exit]

The first time you will get a message about keys, accept this (this is explained in the ssh chapter).



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 07:5b:6c:62:3f:49:5f:8b:0f:3b:84:b6:8b:56:58:c3
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

Enter your userid (or root) and the correct password (nothing will appear on the screen when typing a password).



```
login as: root
root@192.168.1.110's password:
Last login: Sun Nov  2 12:07:49 2014 from 192.168.1.35
[root@server33 ~]#
[root@server33 ~]# hostname
server33.netsec.local
[root@server33 ~]#
```

6. getting Linux at home

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

`This chapter shows a Ubuntu install in Virtualbox. Consider it legacy and use CentOS7 or Debian8 instead (each have their own chapter now).`

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

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

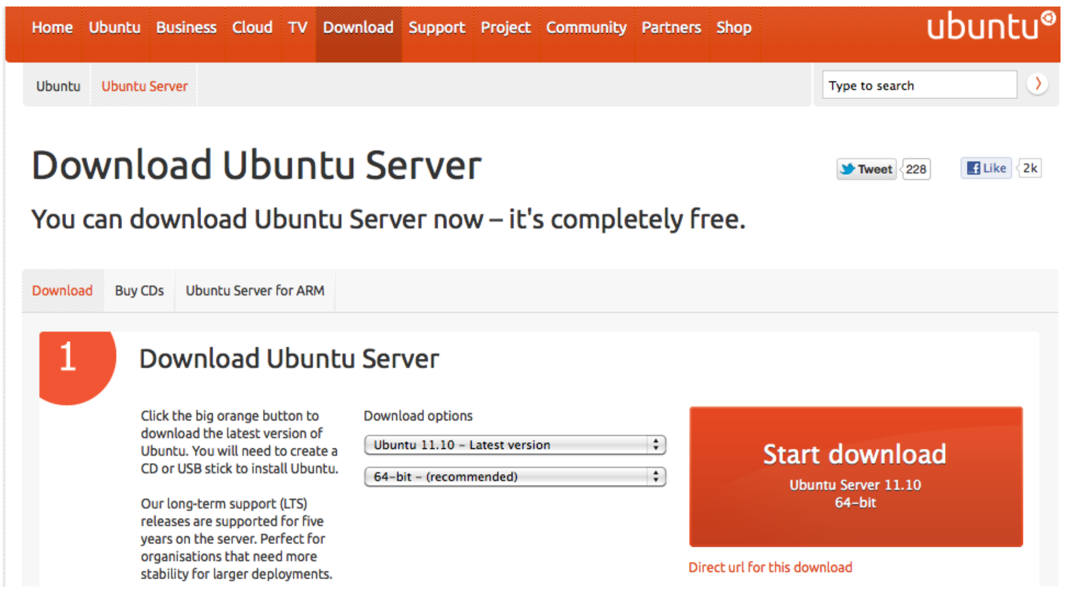
If you do not have access to a Linux computer at the moment, and if you are unable or unsure about installing Linux on your computer, then this chapter proposes a third option: installing Linux in a virtual machine.

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

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

6.1. download a Linux CD image

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



Download Ubuntu Server

You can download Ubuntu Server now – it's completely free.

Download Ubuntu Server

Click the big orange button to download the latest version of Ubuntu. You will need to create a CD or USB stick to install Ubuntu.

Our long-term support (LTS) releases are supported for five years on the server. Perfect for organisations that need more stability for larger deployments.

Download options

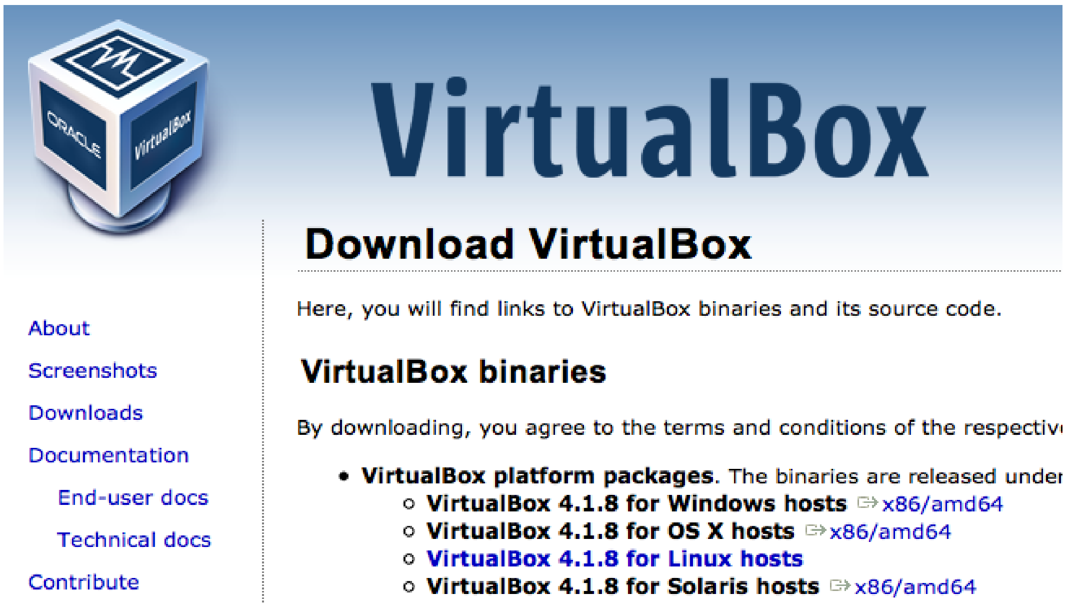
Ubuntu 11.10 – Latest version

64-bit – (recommended)

Ubuntu Server 11.10
64-bit

6.2. download Virtualbox

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



VirtualBox

Download VirtualBox

About
Screenshots
Downloads
Documentation
End-user docs
Technical docs
Contribute

Here, you will find links to VirtualBox binaries and its source code.

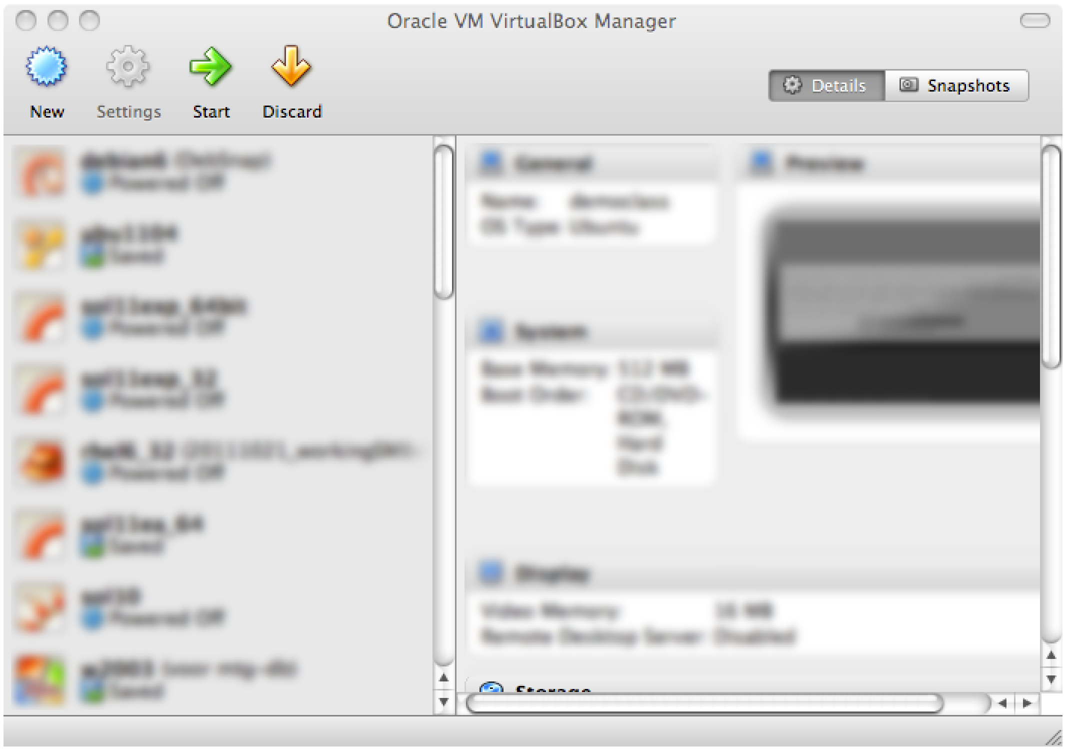
VirtualBox binaries

By downloading, you agree to the terms and conditions of the respectiv

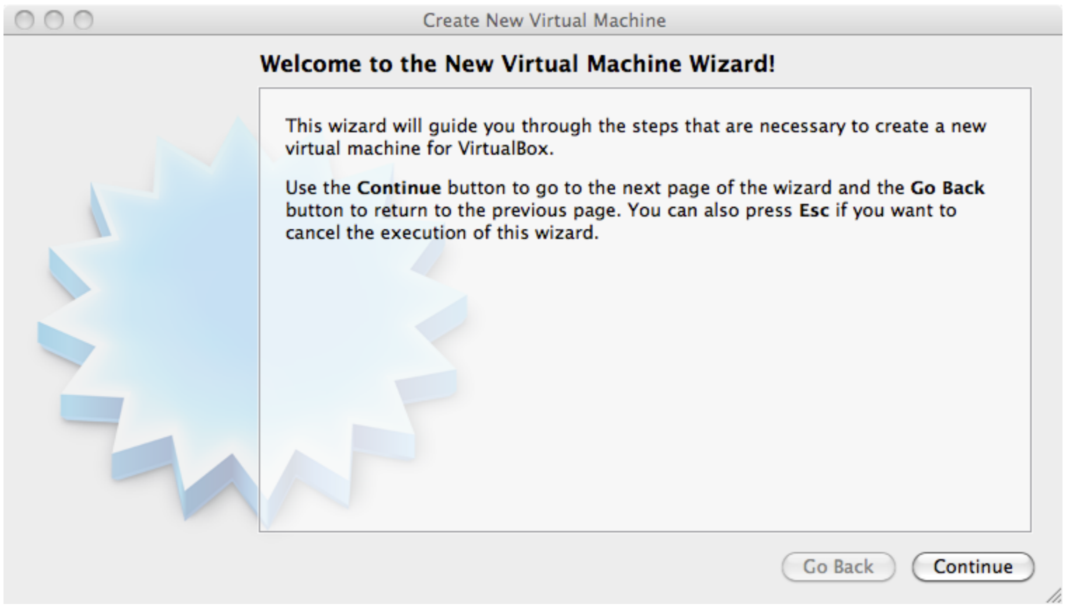
- **VirtualBox platform packages**. The binaries are released under
  - **VirtualBox 4.1.8 for Windows hosts** ⇨x86/amd64
  - **VirtualBox 4.1.8 for OS X hosts** ⇨x86/amd64
  - **VirtualBox 4.1.8 for Linux hosts**
  - **VirtualBox 4.1.8 for Solaris hosts** ⇨x86/amd64

6.3. create a virtual machine

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



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

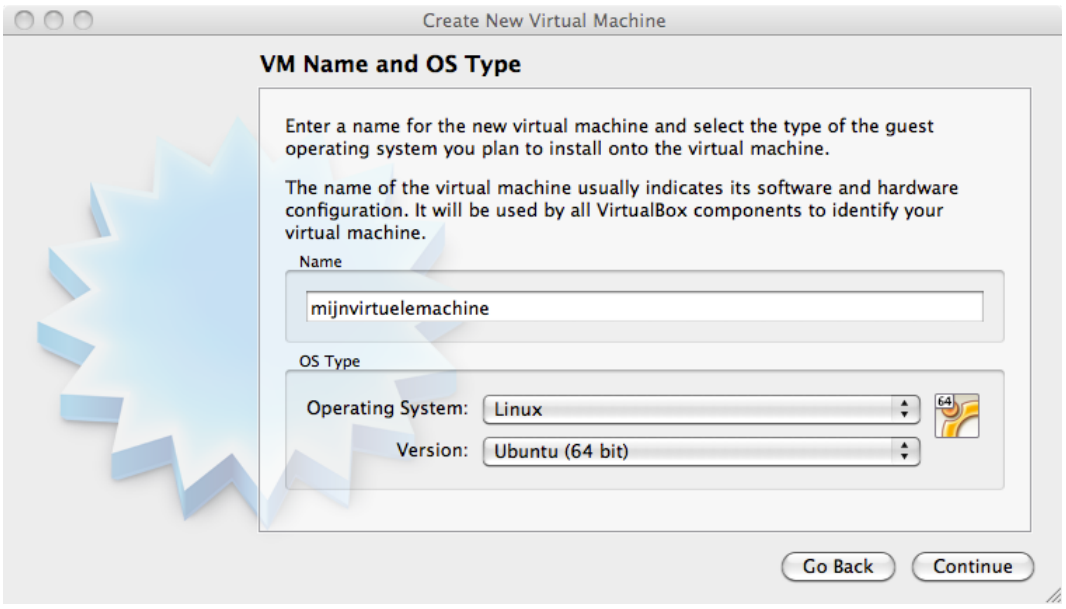


Welcome to the New Virtual Machine Wizard!

This wizard will guide you through the steps that are necessary to create a new virtual machine for VirtualBox.

Use the **Continue** button to go to the next page of the wizard and the **Go Back** button to return to the previous page. You can also press **Esc** if you want to cancel the execution of this wizard.

Name your virtual machine (and maybe select 32-bit or 64-bit).



VM Name and OS Type

Enter a name for the new virtual machine and select the type of the guest operating system you plan to install onto the virtual machine.

The name of the virtual machine usually indicates its software and hardware configuration. It will be used by all VirtualBox components to identify your virtual machine.

Name

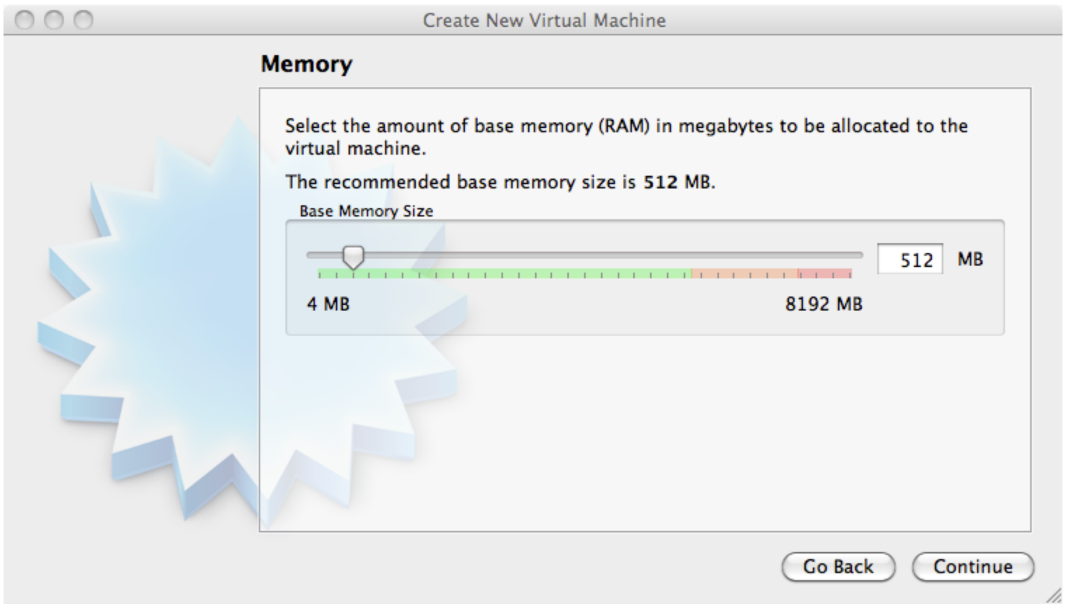
mijnvirtuelemachine

OS Type

Operating System: Linux

Version: Ubuntu (64 bit)

Give the virtual machine some memory (512MB if you have 2GB or more, otherwise select 256MB).



Memory

Select the amount of base memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended base memory size is **512** MB.

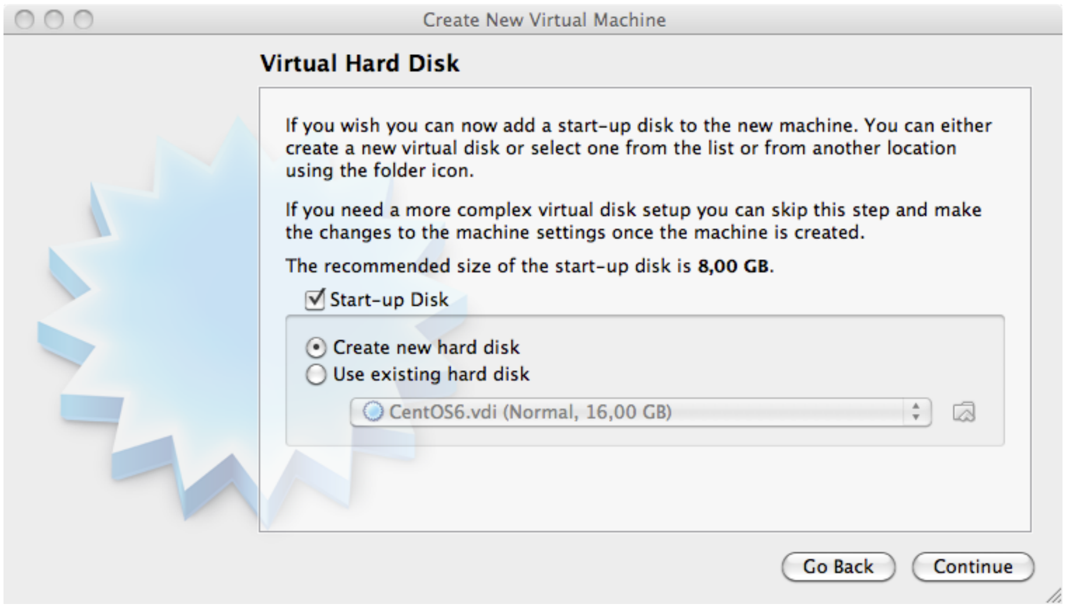
Base Memory Size

512 MB

4 MB

8192 MB

Select to create a new disk (remember, this will be a virtual disk).



Virtual Hard Disk

If you wish you can now add a start-up disk to the new machine. You can either create a new virtual disk or select one from the list or from another location using the folder icon.

If you need a more complex virtual disk setup you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the start-up disk is **8,00 GB**.

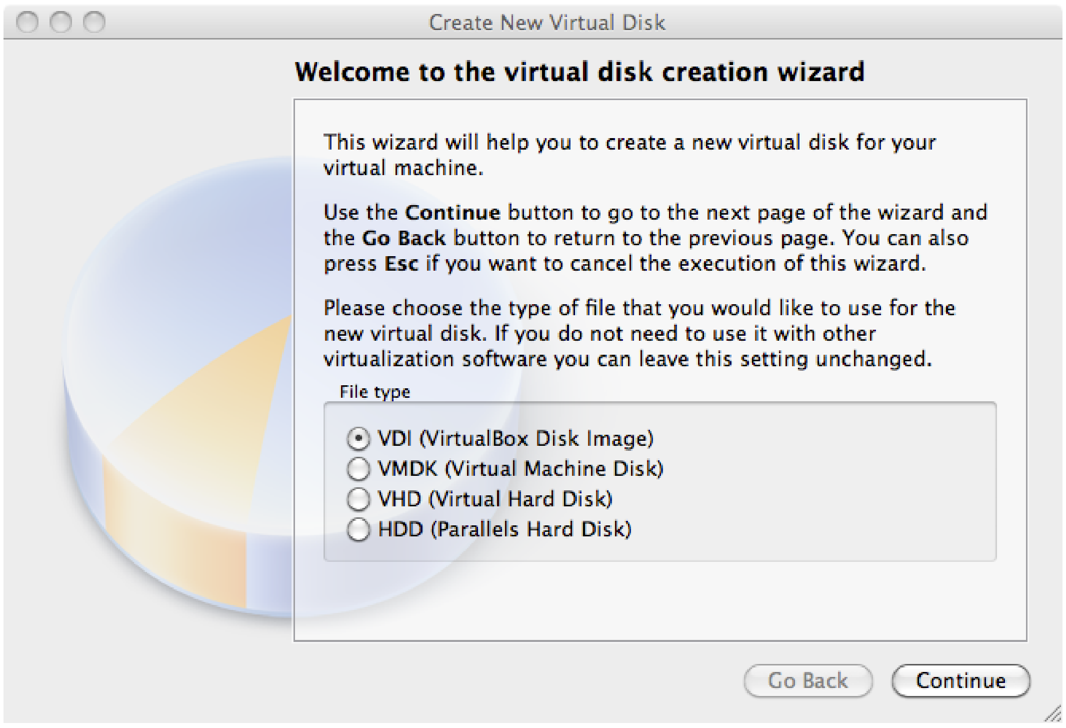
✓ Start-up Disk

● Create new hard disk

○ Use existing hard disk

CentOS6.vdi (Normal, 16,00 GB)

If you get the question below, choose vdi.


Create New Virtual Disk

Welcome to the virtual disk creation wizard

This wizard will help you to create a new virtual disk for your virtual machine.

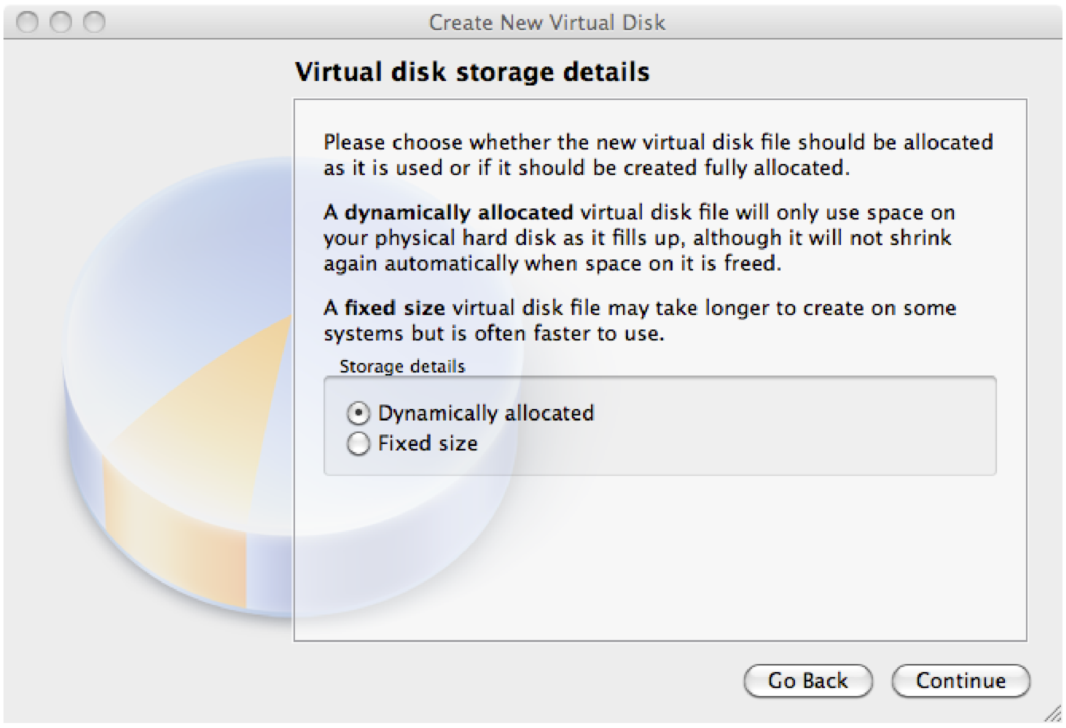
Use the **Continue** button to go to the next page of the wizard and the **Go Back** button to return to the previous page. You can also press **Esc** if you want to cancel the execution of this wizard.

Please choose the type of file that you would like to use for the new virtual disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

File type

[✓ VDI (VirtualBox Disk Image)] VMDK (Virtual Machine Disk) VHD (Virtual Hard Disk) HDD (Parallels Hard Disk)

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


Create New Virtual Disk

Virtual disk storage details

Please choose whether the new virtual disk file should be allocated as it is used or if it should be created fully allocated.

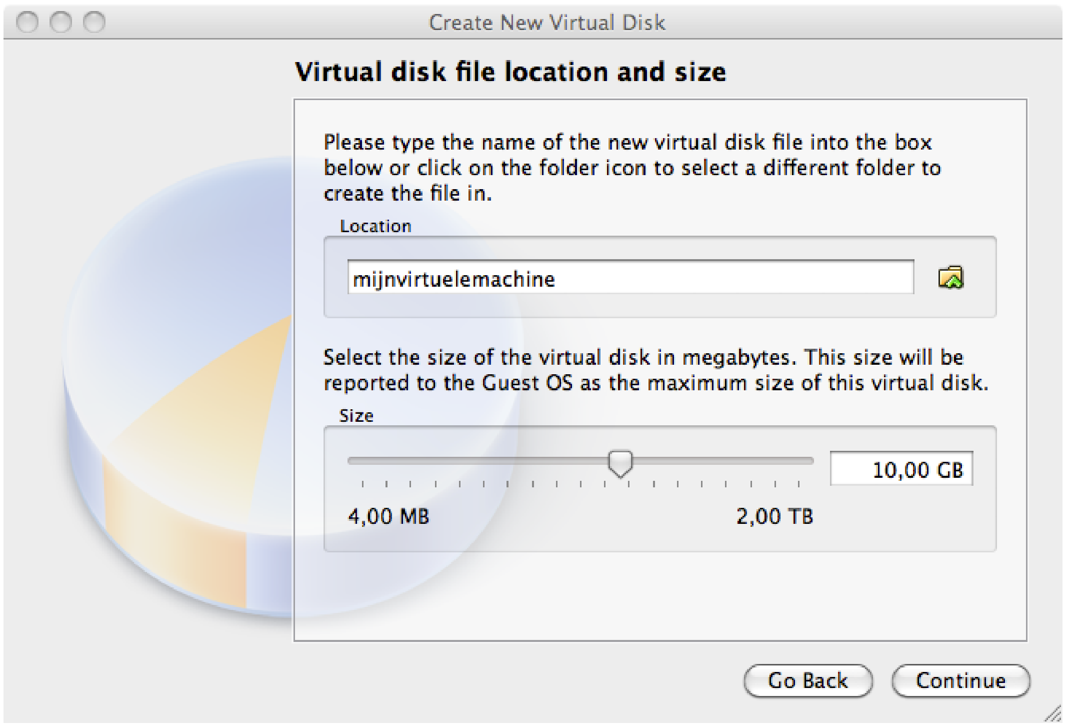
A **dynamically allocated** virtual disk file will only use space on your physical hard disk as it fills up, although it will not shrink again automatically when space on it is freed.

A **fixed size** virtual disk file may take longer to create on some systems but is often faster to use.

Storage details

[✓ Dynamically allocated] Fixed size

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



Virtual disk file location and size

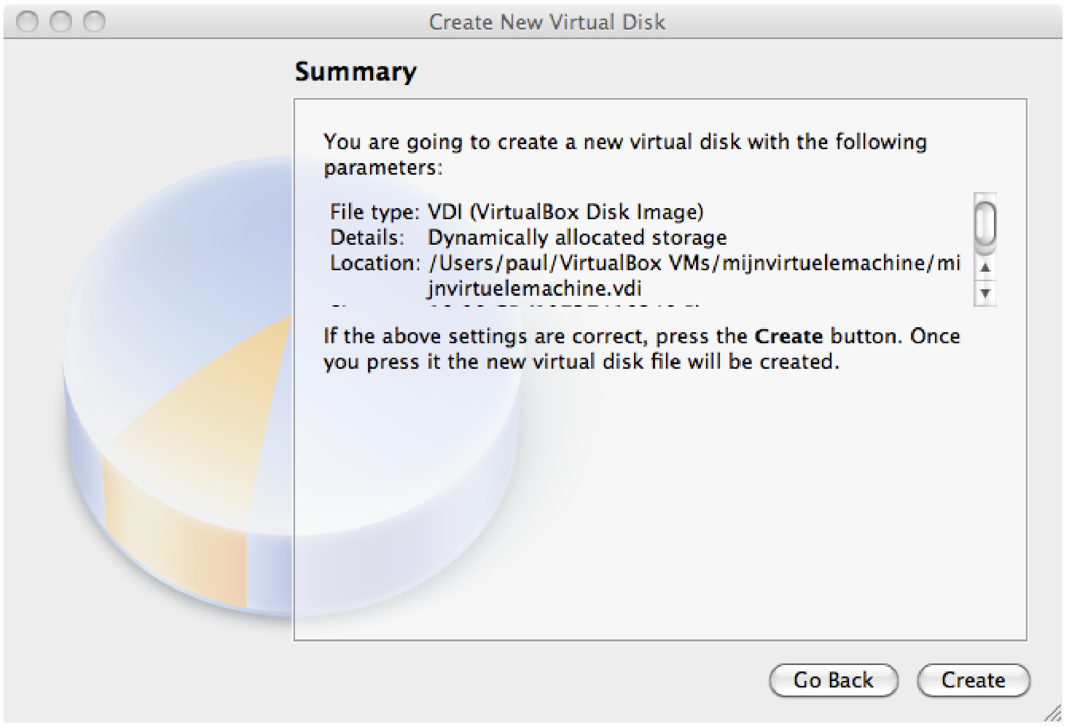
Please type the name of the new virtual disk file into the box below or click on the folder icon to select a different folder to create the file in.

Location: mijnvirtuelemachine

Select the size of the virtual disk in megabytes. This size will be reported to the Guest OS as the maximum size of this virtual disk.

Size: 10,00 GB

Click `create` to create the virtual disk.



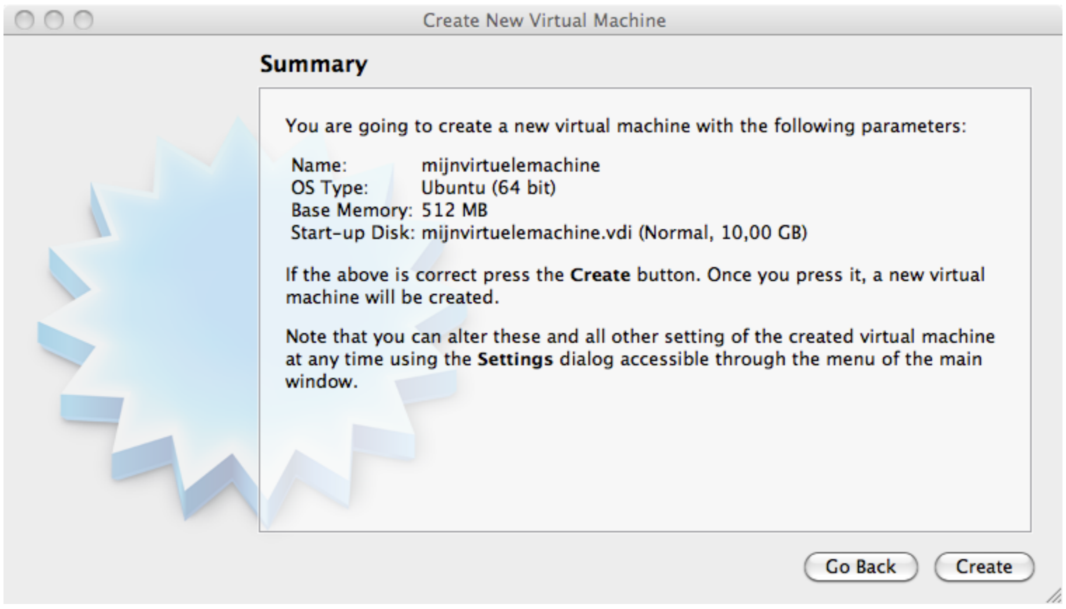
Summary

You are going to create a new virtual disk with the following parameters:

```
File type: VDI (VirtualBox Disk Image)
Details:   Dynamically allocated storage
Location:  /Users/paul/VirtualBox VMs/mijnvirtuelemachine/mi
           jnvirtuelemachine.vdi
```

If the above settings are correct, press the **Create** button. Once you press it the new virtual disk file will be created.

Click `create` to create the virtual machine.



Summary

You are going to create a new virtual machine with the following parameters:

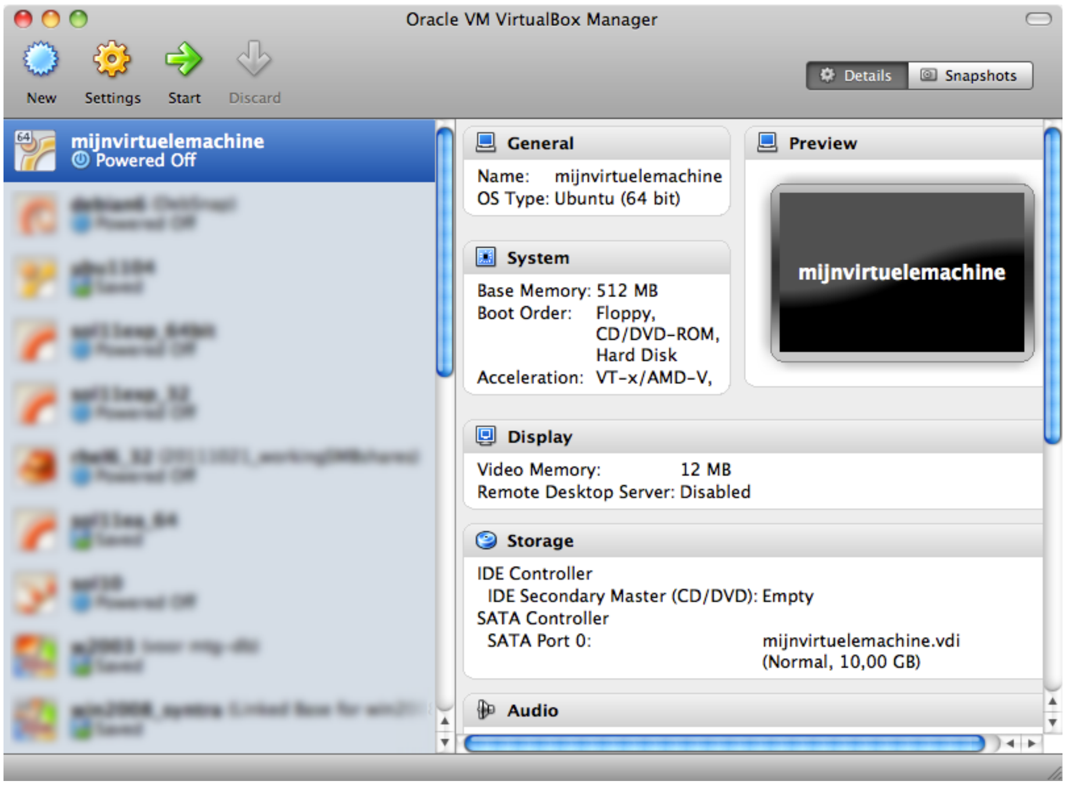
Name:	mijnvirtuelemachine
OS Type:	Ubuntu (64 bit)
Base Memory:	512 MB
Start-up Disk:	mijnvirtuelemachine.vdi (Normal, 10,00 GB)

If the above is correct press the **Create** button. Once you press it, a new virtual machine will be created.

Note that you can alter these and all other setting of the created virtual machine at any time using the **Settings** dialog accessible through the menu of the main window.

6.4. attach the CD image

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



General

Name: mijnvirtuelemachine
OS Type: Ubuntu (64 bit)

System

Base Memory: 512 MB
Boot Order: Floppy, CD/DVD-ROM, Hard Disk
Acceleration: VT-x/AMD-V,

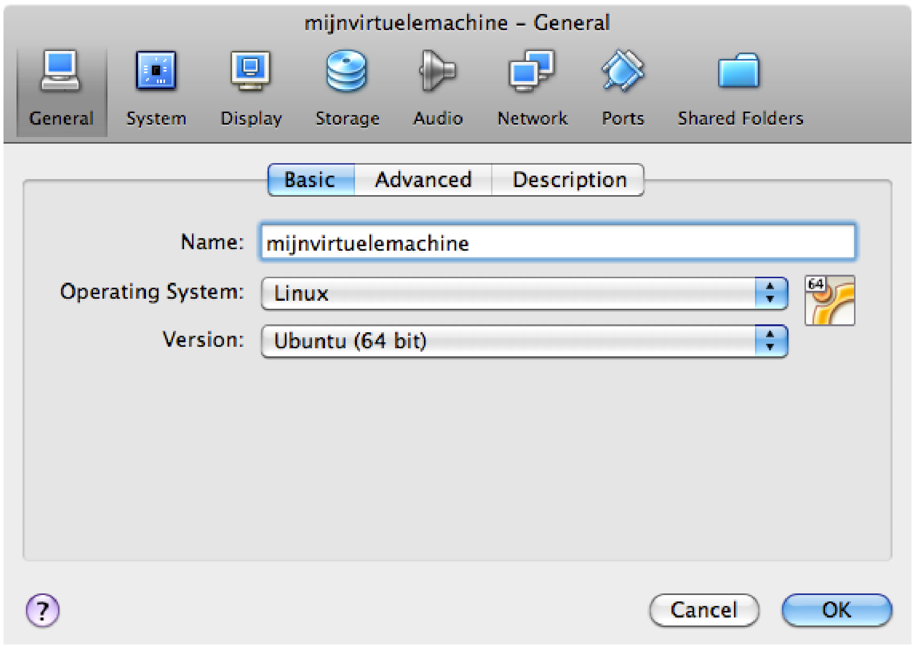
Display

Video Memory: 12 MB
Remote Desktop Server: Disabled

Storage

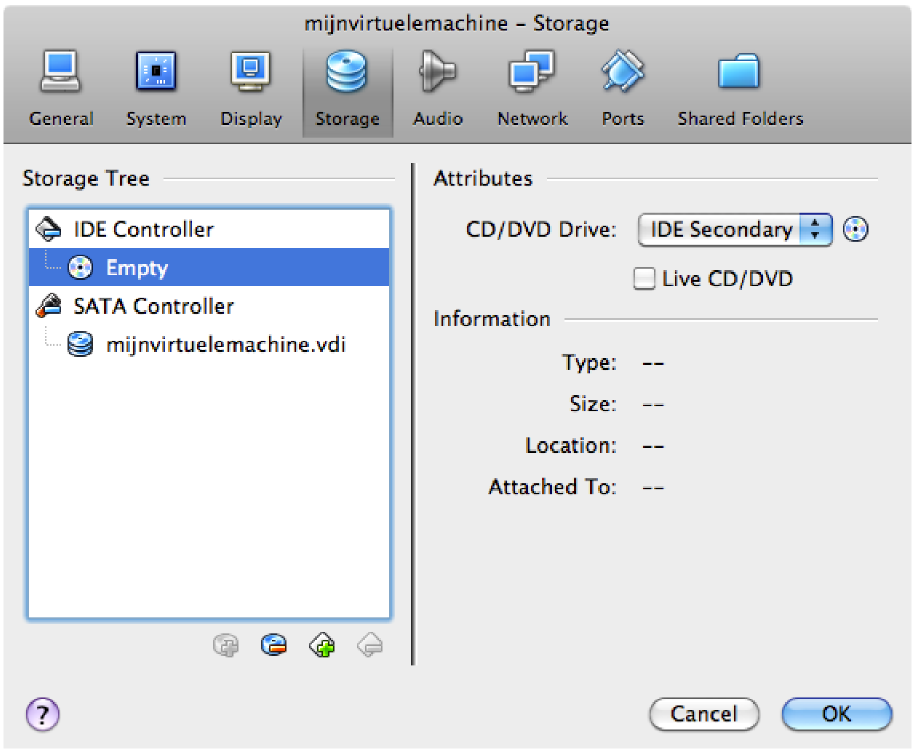
IDE Controller
IDE Secondary Master (CD/DVD): Empty
SATA Controller
SATA Port 0: mijnvirtuelemachine.vdi (Normal, 10,00 GB)

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



Name: mijnvirtuelemachine
Operating System: Linux
Version: Ubuntu (64 bit)

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

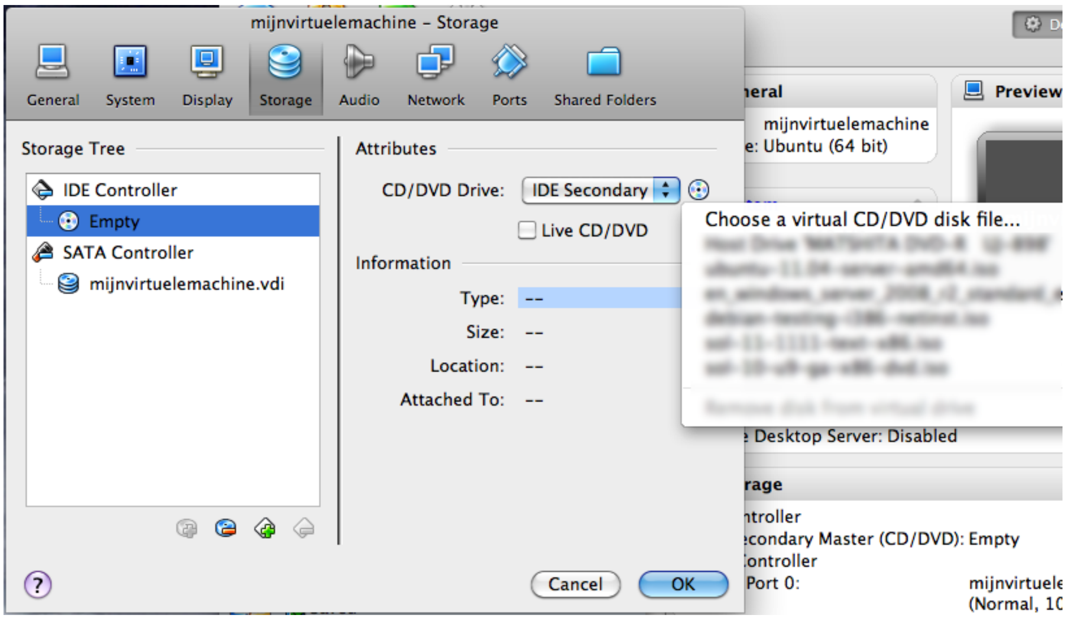


IDE Controller
Empty
SATA Controller
mijnvirtuelemachine.vdi

CD/DVD Drive: IDE Secondary
Live CD/DVD

Type: --
Size: --
Location: --
Attached To: --

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



Storage Tree

IDE Controller
Empty
SATA Controller
mijnvirtuelemachine.vdi

Attributes

CD/DVD Drive: IDE Secondary

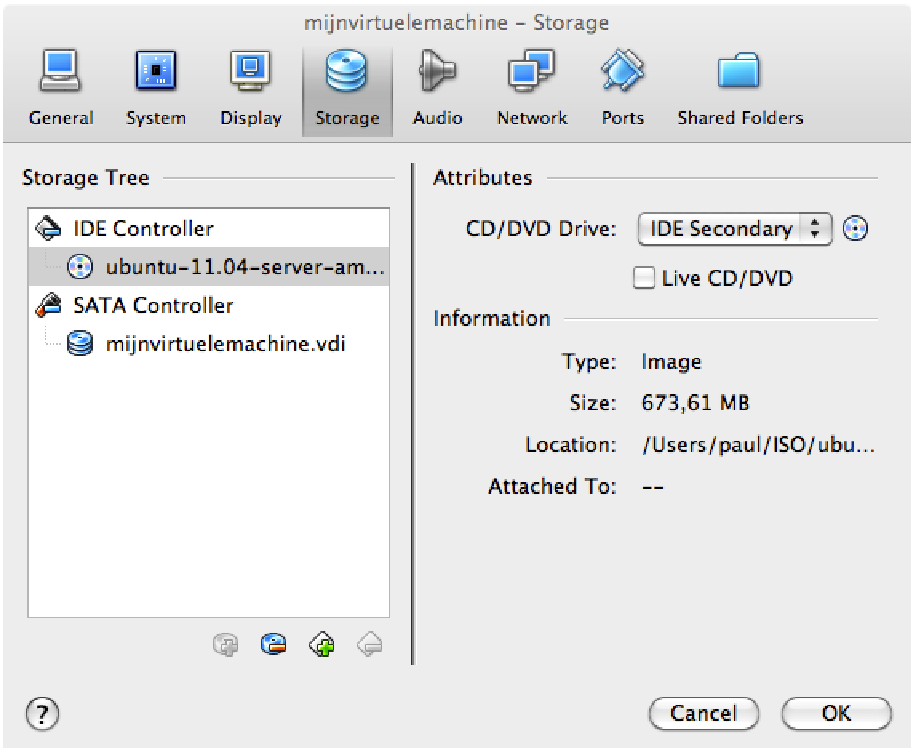
Live CD/DVD

Information

Type:	--
Size:	--
Location:	--
Attached To:	--

mijnvirtuelemachine
e: Ubuntu (64 bit)
Desktop Server: Disabled
rage
ntroller
condary Master (CD/DVD): Empty
ontroller
Port 0: mijnvirtuele
(Normal, 10

Verify that your download is accepted. If Virtualbox complains at this point, then you probably did not finish the download of the CD (try downloading it again).



Storage Tree

IDE Controller
ubuntu-11.04-server-am...
SATA Controller
mijnvirtuelemachine.vdi

Attributes

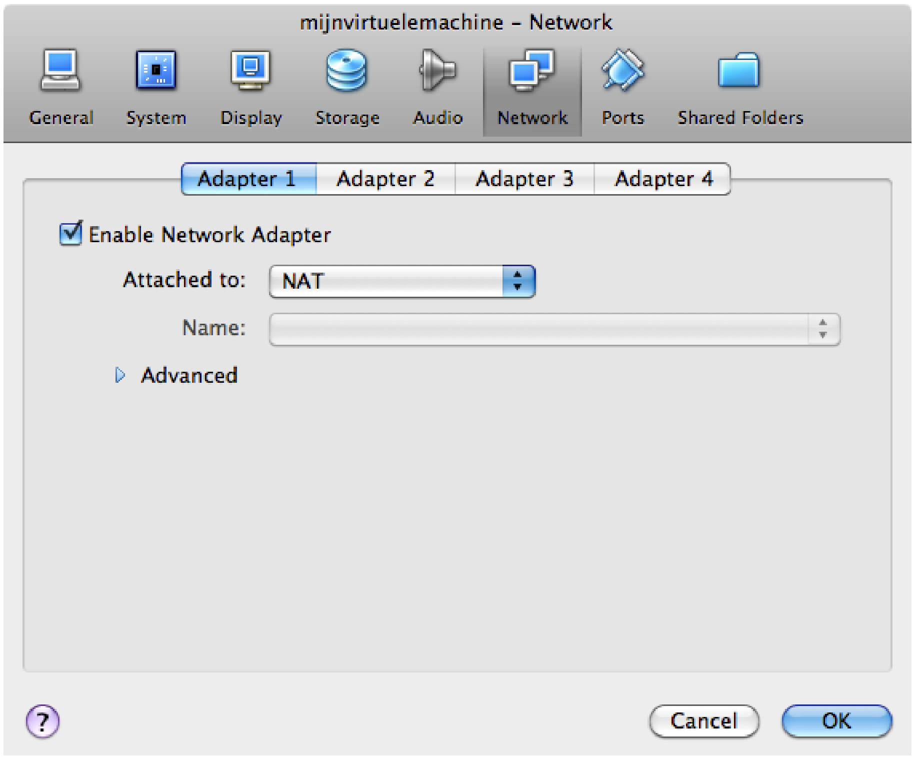
CD/DVD Drive: IDE Secondary

Live CD/DVD

Information

Type:	Image
Size:	673,61 MB
Location:	/Users/paul/ISO/ubu...
Attached To:	--

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


mijnvirtuelemachine – Network

☑ Enable Network Adapter

Attached to: NAT

Name:

6.5. install Linux

The virtual machine is now ready to start. When given a choice at boot, select `install` and follow the instructions on the screen. When the installation is finished, you can log on to the machine and start practising Linux!

Part III.

First steps on the command line

7. 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.

7.1. 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...
```

7.2. man $configfile

Most `configuration files` have their own manual.

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

7.3. man $daemon

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

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

7.4. 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
```

7.5. 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
```

7.6. 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
```

7.7. 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]
```

7.8. 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
```

7.9. 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
```

7.10. 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
```

8. 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.

8.1. 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
```

8.2. 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
```

8.2.1. 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
```

8.2.2. 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.

8.2.3. 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
```

8.3. 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.

8.4. 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!

8.5. ls

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

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

8.5.1. 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:~$
```

8.5.2. 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
```

8.5.3. 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:~$
```

8.6. 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$
```

8.6.1. 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
```

8.7. 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:~$
```

8.7.1. 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:~$
```

8.8. 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.

8.9. solution: working with directories

1. Display your current directory.

```
pwd
```

2. Change to the /etc directory.

```
cd /etc
```

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

```
cd (and the enter key)
```

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

```
cd /boot/grub (use the tab key)
```

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

```
cd .. (with space between cd and ..)
```

6. Go to the root directory.

```
cd /
```

7. List the contents of the root directory.

```
ls
```

8. List a long listing of the root directory.

```
ls -l
```

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

```
ls /etc
```

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

```
ls /bin /sbin
```

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

```
ls ~
```

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

```
ls -al ~
```

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

```
ls -lh /boot
```

14. Create a directory testdir in your home directory.

```
mkdir ~/testdir
```

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

```
cd /etc ; mkdir ~/newdir
```

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

```
mkdir -p ~/dir1/dir2/dir3
```

17. Remove the directory testdir.

```
rmdir testdir
```

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

```
man bash           # 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
```

9. 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`.

9.1. 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!
```

9.2. 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`.

9.3. 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=0x83, active, starthead...
root@linux~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@linux~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

9.4. touch

9.4.1. 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:~$
```

9.4.2. 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:~$
```

9.5. rm

9.5.1. 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:~$
```

9.5.2. 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:~$
```

9.5.3. 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:~$
```

9.6. cp

9.6.1. 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
```

9.6.2. 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
```

9.6.3. 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
```

9.6.4. 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
```

9.6.5. 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:~$
```

9.7. mv

9.7.1. 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.

9.7.2. 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:~$
```

9.7.3. 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:~$
```

9.8. rename

9.8.1. 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.

9.8.2. 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$
```

9.8.3. 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 ~]$
```

9.9. 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!)

9.10. 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
```

10. 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.

10.1. 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~$
```

10.2. 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.

10.3. 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
```

10.3.1. 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:~$
```

10.3.2. 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.

10.3.3. 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:~$
```

10.3.4. 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:~$
```

10.4. 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
```

10.5. 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`.

10.6. 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
...
```

10.7. 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`.

10.8. 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`.

```
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
```

11. 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.

11.1. 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."

11.2. 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.

11.3. 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
```

11.4. 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`.

11.4.1. /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
...
```

11.4.2. 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.

11.4.3. /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
```

11.4.4. /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
```

11.4.4.1. /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.

11.4.4.2. /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`.

11.4.5. /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`.

11.5. configuration directories

11.5.1. /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.

11.5.2. /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`.

11.5.2.1. /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`.

11.5.2.2. /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`.

11.5.2.3. /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.

11.5.2.4. /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)`, `load-keys(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.

11.6. data directories

11.6.1. /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
```

11.6.2. /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.

11.6.3. /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.

11.6.4. /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
```

11.6.5. /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.

11.6.6. /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.

11.7. in memory directories

11.7.1. /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.

11.7.1.1. 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.

11.7.1.2. /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).

11.7.1.3. /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 ;-).*

11.7.2. /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.

11.7.2.1. /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
```

11.7.2.2. /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:~$
```

11.7.3. /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.

11.8. /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).

11.8.1. /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`.)

11.8.2. /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
...
```

11.8.3. /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
```

11.8.4. /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/
```

11.8.5. /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
```

11.8.6. /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
```

11.9. /var variable data

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

11.9.1. /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
```

11.9.2. /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
```

11.9.3. /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
```

11.9.4. /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).

11.9.5. /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/.

11.9.6. /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.

11.10. 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.

11.11. 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.
```

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 IV.

Shell expansion

12. 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
```

12.1. 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.

12.2. 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.

12.3. 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 ~]$
```

12.4. 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.

12.5. 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.

12.6. commands

12.6.1. 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.

12.6.2. 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'
```

12.6.3. 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...
```

12.6.4. 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
```

12.7. aliases

12.7.1. 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
```

12.7.2. 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:~$
```

12.7.3. 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')

12.7.4. 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'
```

12.7.5. 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 ~]$
```

12.8. 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
```

12.9. 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.

12.10. 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)
```

12. commands and arguments

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"
```

13. 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(&).

13.1. ; 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 ~]$
```

13.2. & 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`.

13.3. $? 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 $?
1
student@linux:~/test$
```

13.4. && 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
```

13.5. || 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
```

13.6. 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$
```

13.7. # 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$
```

13.8. \ 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 \?*"'
```

13.8.1. 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 ~]$
```

13.9. 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)

13.10. 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 \ * [ } ~ \\ . "\"
```

14. 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.

14.1. $ 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
```

14.2. case sensitive

This example shows that shell variables are case sensitive!

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

14.3. 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 $MyVar
555
[student@linux gen]$
```

14.4. 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.
```

14.5. 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.

14.6. 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 ~]$
```

14.7. $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\$ "
```

14.8. $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 ~]#
```

14.9. 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]$
```

14.10. 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 $var5
three four
[student@linux ~]$
```

14.11. 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 ${prefix}girl
Hello Superman and Supergirl
[student@linux ~]$
```

14.12. 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`.

14.13. 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.

14.14. 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$ '
```

15. shell embedding and options

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

This chapter takes a brief look at `child shells`, `embedded shells` and `shell options`.

15.1. shell embedding

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

```
[student@linux gen]$ echo $var1

[student@linux gen]$ echo $(var1=5;echo $var1)
5
[student@linux gen]$ echo $var1

[student@linux gen]$
```

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

This screenshot shows an embedded shell inside an embedded shell.

```
student@linux:~$ A=shell
student@linux:~$ echo $C$B$A $(B=sub;echo $C$B$A; echo $(C=sub;echo $C$B$A))
shell subshell subsubshell
```

15.1.1. backticks

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

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

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

15.1.2. backticks or single quotes

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

```
[student@linux gen]$ echo `var1=5;echo $var1`
5
[student@linux gen]$ echo 'var1=5;echo $var1'
var1=5;echo $var1
[student@linux gen]$
```

15.2. shell options

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

```
[student@linux ~]$ echo $var123

[student@linux ~]$ set -u
[student@linux ~]$ echo $var123
-bash: var123: unbound variable
[student@linux ~]$ set +u
[student@linux ~]$ echo $var123

[student@linux ~]$
```

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

```
[student@linux ~]$ echo $-
himBH
[student@linux ~]$ set -C ; set -u
[student@linux ~]$ echo $-
himuBCH
[student@linux ~]$ set +C ; set +u
[student@linux ~]$ echo $-
himBH
[student@linux ~]$
```

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

15.3. practice: shell embedding

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

2. Activate `nounset` in your shell. Test that it shows an error message when using non-existing variables.

3. Deactivate nounset.

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

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

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

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

(optional)7. Given the following screenshot, add exactly four characters to that command line so that the total output is FirstMiddleLast.

```
[student@linux ~]$ echo  First; echo  Middle; echo  Last
```

8. Display a `long listing` (ls -l) of the `passwd` command using the `which` command inside an embedded shell.

15.4. solution: shell embedding

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

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

2. Activate `nounset` in your shell. Test that it shows an error message when using non-existing variables.

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

Both these lines have the same effect.

3. Deactivate nounset.

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

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

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

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

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

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

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

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

```
It displays shell expansion for troubleshooting your command.
```

(optional)7. Given the following screenshot, add exactly four characters to that command line so that the total output is FirstMiddleLast.

```
[student@linux ~]$ echo  First; echo  Middle; echo  Last
```

```
echo -n First; echo -n Middle; echo Last
```

8. Display a `long listing` (ls -l) of the `passwd` command using the `which` command inside an embedded shell.

```
ls -l $(which passwd)
```

16. 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.

16.1. 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$
```

16.2. 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$
```

16.3. 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
```

16.4. !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
```

16.5. 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
```

16.6. $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
```

16.7. $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.*

16.8. $HISTFILESIZE

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

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

16.9. 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
```

16.10. (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$
```

16.11. (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
$
```

16.12. 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 ?

16.13. 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
```

17. file globbing

(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 will explain **file globbing**. Typing `man 7 glob` (on Debian) will tell you that long ago there was a program called `/etc/glob` that would expand *wildcard patterns*. Soon afterward, this became a shell built-in.

A string is a wildcard pattern if it contains `?`, `*` or `[`. *Globbing* (or dynamic filename generation) is the operation that expands a wildcard pattern into a list of pathnames that match the pattern.

17.1. * 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.

```
student@linux:~/gen$ ls
file1  file2  file3  File4  File55  FileA  fileå  fileab  Fileab  FileAB  fileabc  fileæ  fileø  filex  filey  filez
student@linux:~/gen$ ls File*
File4  File55  FileA  Fileab  FileAB
student@linux:~/gen$ ls file*
file1  file2  file3  fileå  fileab  fileabc  fileæ  fileø  filex  filey  filez
student@linux:~/gen$ ls *ile55
File55
student@linux:~/gen$ ls F*ile55
File55
student@linux:~/gen$ ls F*55
File55
```

17.2. ? 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 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
```

17.3. [ ] 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 File[5A]
FileA
student@linux:~/gen$ ls File[A5]3
ls: cannot access 'File[A5]3': No such file or directory
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: cannot access 'File[a5][5b][abcdefghijklm]': No such file or directory
student@linux:~/gen$ ls file[a5][5b][abcdefghijklm]
fileabc
```

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

```
student@linux:~/gen$ ls file[a5][!Z]
fileab
student@linux:~/gen$ ls file[!5]*
file1  file2  file3  fileå  fileab  fileabc  fileæ  fileø  filex  filey  filez
student@linux:~/gen$ ls file[!5]?
fileab
```

17.4. a-z and 0-9 ranges

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

```
student@linux:~/gen$ ls file[a-z]*
fileab  fileabc  filex  filey  filez
student@linux:~/gen$ ls file[0-9]
file1  file2  file3
student@linux:~/gen$ ls file[a-z][a-z][0-9]*
ls: cannot access 'file[a-z][a-z][0-9]*': No such file or directory
student@linux:~/gen$ ls file[a-z][a-z][a-z]*
fileabc
```

17.5. named character classes

Instead of ranges, you can also specify named character classes: [[:alnum:]],, [[:alpha:]], [[:blank:]], [[:cntrl:]], [[:digit:]], [[:graph:]], [[:lower:]], [[:print:]], [[:punct:]], [[:space:]], [[:upper:]], [[:xdigit:]]. Instead of, e.g. [a-z], you can also use [[:lower:]].

```
student@linux:~/gen$ ls file[a-z]*
fileab  fileabc  filex  filey  filez
student@linux:~/gen$ ls file[[:lower:]]*
fileå  fileab  fileabc  fileæ  fileø  filex  filey  filez
```

Remark that the named character classes work better for international characters. In the example above, [a-z] does not match the Danish characters æ, ø, and å, but [[:lower:]] does.

17.6. $LANG and square brackets

But, don't forget the influence of the $LANG variable. Depending on the selected language or locale, the shell will interpret the square brackets and named character classes differently. Sort order may also be affected.

For example, when we select the default locale called C:

```
student@linux:~/gen$ sudo localectl set-locale C
[ ... log out and log in again ... ]
student@linux:~/gen$ echo $LANG
C
student@linux:~/gen$ ls
 File4   File55   FileA   FileAB   Fileab   file1   file2   file3   fileab
 ↪  fileabc   filex   filey   filez  'file'$'\303\245'  'file'$'\303\246'
 ↪  'file'$'\303\270'
student@linux:~/gen$ ls file[[:lower:]]*
fileab  fileabc  filex  filey  filez
```

The Danish characters can't be displayed properly and don't match the [[:lower:]] character class.

Let us change the locale to da_DK.UTF-8 (Danish/Denmark with UTF-8 support) and see what happens:

```
student@linux:~/gen$ sudo localectl set-locale da_DK.UTF-8
[ ... log out and log in again ... ]
student@linux:~/gen$ echo $LANG
da_DK.UTF-8
student@linux:~/gen$ ls
file1  file2  file3  File4  File55  FileA  FileAB  Fileab  fileab  fileabc
 ↪  filex  filey  filez  fileæ  fileø  fileå
student@linux:~/gen$ ls file[[:lower:]]*
fileab  fileabc  filex  filey  filez  fileæ  fileø  fileå
```

Now the Danish characters are displayed properly and match the [[:lower:]] character class.

In the en_US.UTF-8 locale (US English, with UTF-8 support), the Danish characters are displayed properly, and also match the [[:lower:]] character class. However, they are sorted differently:

```
student@linux:~/gen$ sudo localectl set-locale en_US.UTF-8
[ ... log out and log in again ... ]
student@linux:~/gen$ echo $LANG
en_US.UTF-8
student@linux:~/gen$ ls
file1  file2  file3  File4  File55  FileA  fileå  fileab  Fileab  FileAB
 ↪  fileabc  fileæ  fileø  filex  filey  filez
student@linux:~/gen$ ls file[[:lower:]]*
fileå  fileab  fileabc  fileæ  fileø  filex  filey  filez
```

17.7. preventing file globbing

If a wildcard pattern does not match any filenames, the shell will not expand the pattern. Consequently, when in an empty directory, `echo *` will display a `*`. 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 test{1,2,3}
student@linux:~/test42$ echo *
test1 test2 test3
```

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

```
student@linux:~/test42$ echo *
test1 test2 test3
student@linux:~/test42$ echo \*
*
student@linux:~/test42$ echo '*'
*
student@linux:~/test42$ echo "*"
*
```

17.8. practice: shell globbing

In the questions below, use the `ls` command with globbing patterns to list the specified files. Don't pipe the output to `grep` or another tool to filter on regular expressions!

1. Create a test directory `glob` and enter it.
2. Create the following files :

```
vagrant@ubuntu:~/glob$ ls
'file('   file10  'file 2'   File2   file33   fileA   fileà   fileAAA
 file1    file11   file2     File3   filea    fileá   fileå   fileAB
```

   (remark that `file 2` has a space in the name!)

3. List all files starting with `file`
4. List all files starting with `File`
5. List all files starting with `file` and ending in *a number*.
6. List all files starting with `file` and ending with *a letter*
7. List all files starting with `File` and having a *digit* as *fifth* character.
8. List all files starting with `File` and having a *digit* as *fifth* and last character (i.e. the name consists of five characters).
9. List all files starting with *a letter* and ending in *a number*.
10. List all files that have *exactly five characters*.
11. List all files that start with `f` or `F` and end with `3` or `A`.
12. List 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. Show the influence of `$LANG` (the system locale) in listing `A-Z` or `a-z` ranges.
15. You receive information that one of your servers was cracked. The cracker probably replaced the `ls` command with a rootkit so it can no longer be used safely. 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`?

17.9. solution: shell globbing

1. Create a test directory `glob` and enter it.

```
mkdir glob; cd glob
```

2. Create the files:

```
student@ubuntu:~$ touch file1 file10 file11 file2 File2 File3 file33
    ↪ fileAB
student@ubuntu:~$ touch filea fileá fileà fileå fileA fileAAA 'file('
    ↪ 'file 2'
```

3. List all files starting with `file`

```
student@ubuntu:~/glob$ ls file*
'file('   file10  'file 2'   file33   fileA   fileà   fileAAA
 file1    file11   file2     filea    fileá   fileå   fileAB
```

4. List all files starting with `File`

```
student@ubuntu:~/glob$ ls File*
File2  File3
```

5. List all files starting with `file` and ending in *a number*.

```
student@ubuntu:~/glob$ ls file*[0-9]
file1   file10   file11  'file 2'   file2   file33
```

6. List all files starting with `file` and ending with *a letter*

```
student@ubuntu:~/glob$ ls file*[A-Za-z]
filea  fileA  fileAAA  fileAB
student@ubuntu:~/glob$ ls file*[[:alpha:]]
filea  fileA  fileá  fileà  fileå  fileAAA  fileAB
```

Remark that the first solution is not complete, as it does not list the files with special characters in the name! In this case, it's better to use the named class `[:alpha:]`.

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

```
student@ubuntu:~/glob$ ls File[0-9]*
File2  File3
```

8. List all files starting with `File` and having a *digit* as *fifth* and last character (i.e. the name consists of five characters).

```
student@ubuntu:~/glob$ ls File[0-9]
File2
```

9. List all files starting with *a letter* and ending in *a number*.

```
student@ubuntu:~/glob$ ls [[:alpha:]]*[[:digit:]]
file1   file10   file11  'file 2'   file2   File2   File3   file33
```

10. List all files that have *exactly five characters*.

```
student@ubuntu:~/glob$ ls ?????
'file('   file1   file2   File2   File3   filea   fileA   fileá   fileà
 ↪  fileå
```

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

```
student@ubuntu:~/glob$ ls [fF]*[3A]
File3  file33  fileA  fileAAA
```

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

```
student@ubuntu:~/glob$ ls f[iR]*[0-9]
file1   file10   file11  'file 2'   file2   file33
```

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

```
student@ubuntu:~/glob$ ls [^F]*
'file('   file10  'file 2'   file33   fileA   fileà   fileAAA
 file1    file11   file2     filea    fileá   fileå   fileAB
```

14. Show the influence of `$LANG` (the system locale) in listing `A-Z` or `a-z` ranges.

```
student@ubuntu:~/glob$ LANG=C ls file[[:alpha:]]*
fileA   fileAAA   fileAB   filea  'file'$'\303\240'  'file'$'\303\241'
 ↪  'file'$'\303\245'
student@ubuntu:~/glob$ LANG=en_US.UTF-8 ls file[[:alpha:]]*
filea  fileA  fileá  fileà  fileå  fileAAA  fileAB
student@ubuntu:~/glob$ LANG=da_DK.UTF-8 ls file[[:alpha:]]*
fileA  filea  fileá  fileà  fileAB  fileå  fileAAA
```

15. You receive information that one of your servers was cracked. The cracker probably replaced the `ls` command with a rootkit so it can no longer be used safely. 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`?

```
student@ubuntu:~/glob$ echo *
file( file1 file10 file11 file 2 file2 File2 File3 file33 filea fileA
 ↪  fileá fileà fileå fileAAA fileAB
```

A disadvantage is that you can't see properties of the files, like permissions, owner, group, size, and date. For this, you can use `stat`, e.g. `stat -c '%A %h %U %G %s %y %n' *`.

Part V.

Pipes and commands

18. I/O redirection

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

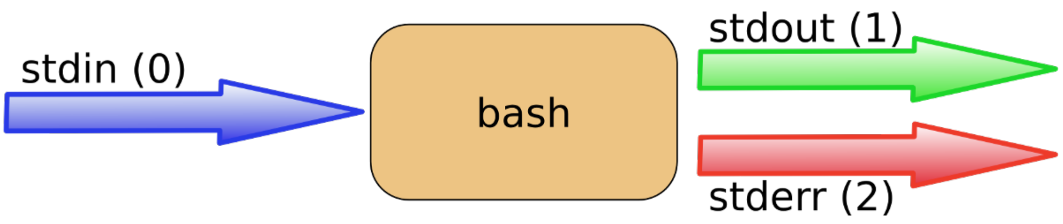
One of the powers of the Unix command line is the use of `input/output redirection` and `pipes`.

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

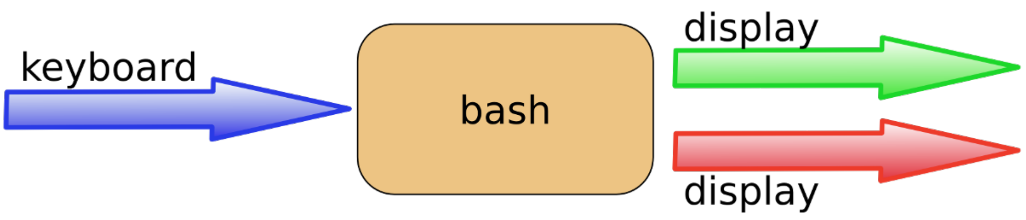
18.1. stdin, stdout, and stderr

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.

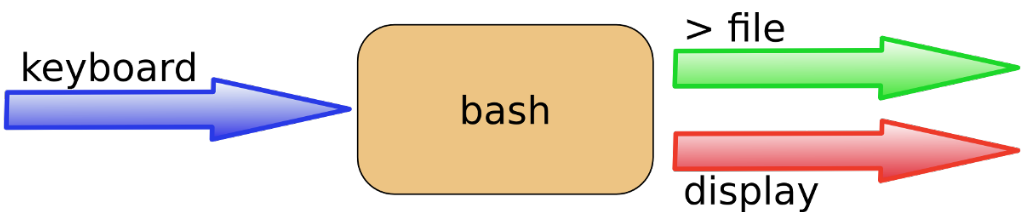


The next sections will explain how to redirect these streams.

18.2. output redirection

18.2.1. > stdout

`stdout` can be redirected to a file 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.

18.2.2. 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 ~]$
```

18.2.3. 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 ~]$
```

18.2.4. 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 ~]$
```

18.2.5. » append

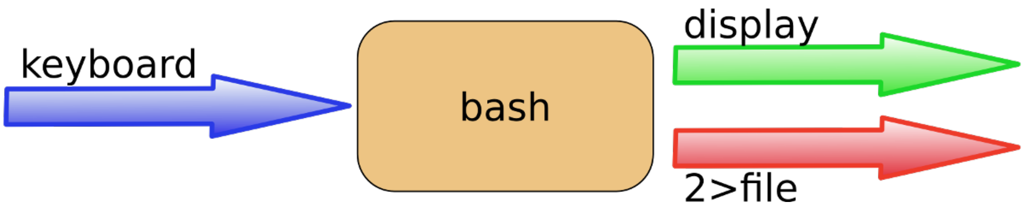
Use >> to append output to a file.

```
[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 ~]$
```

18.3. error redirection

18.3.1. 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 ~]$
```

18.3.2. 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.

18.4. 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
```

18.5. 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:~$
```

18.6. input redirection

18.6.1. < 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 ~]$
```

18.6.2. « 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 ~]$
```

18.6.3. «< 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`.

18.7. 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
```

18.8. 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
```

18.9. 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.

18.10. 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:~$
```

19. filters

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

Filters are commands that take some text input, perform a very specific operation on it, and then output the result. Usually, you can specify a file as argument to take input from. However, they are also designed to be used with a *pipe* (`|`, see I/O redirection) to perform that task on the standard input. The combination of several filters in a *pipeline* allows you to automate quite complex text processing tasks.

This is the essence of the *Unix philosophy*: small, simple tools that do one thing well, and can be combined to perform more complex tasks. Most of the tools discussed in this chapter are in fact older than Linux and are standardised by the POSIX standard. This means that they are available on all Unix-like systems, like Free/Open/NetBSD and macOS.

This chapter will introduce you to the most common filter commands. Be sure to also read each command's `man` page to learn more about their options and features!

19.1. filters and pipes

Typical patterns for using filters are:

- `command file` reads input from a file, specified as an argument.
- `command < file` same, but if the command does not accept a file as argument, you can use input redirection.
- `command1 | command2` uses a *pipe* to send the output of `command1` to `command2`.

Sometimes, in a pipeline, unnecessary commands are used. For example:

- `cat file | command` can always be replaced by `command < file` or `command file` if the command accepts a file as argument.
- `echo "string" | command` can always be replaced by `command <<< "string"` (a here string).

This may seem trivial, but it can make a difference in performance and memory usage. The `cat` and `echo` commands must be loaded into memory and use system resources, while the input redirection and here string are handled by the shell itself.

19.2. cat

Cat is short for *concatenate*. When between two *pipes*, the `cat` command does nothing except repeating `stdin` on `stdout`. The command can also take input from a file.

```
student@linux:~$ cat count.txt
one
two
three
four
five
```

```
student@linux:~$ cat count.txt | cat | cat | cat | cat | cat
one
two
three
four
five
```

In scripts, `cat` is used to print out multiple lines (instead of using `echo` multiple times) using a here document.

```
#!/bin/bash
cat << _EOF_
one
two
three
four
five
_EOF_
```

19.3. tac

The `tac` command is the reverse of `cat`. It prints the lines of a file in reverse order.

```
student@linux:~$ tac count.txt
five
four
three
two
one
```

19.4. shuf

The `shuf` command is used to randomly shuffle the lines of a file.

```
student@linux:~$ shuf count.txt
four
one
three
five
two
```

19.5. tee

Writing long *pipes* in Unix is fun, but sometimes you may want intermediate results. Or, you may want to see the standard output of a command and also save it to a file. This is were `tee` comes in handy.

The `tee` filter puts `stdin` on `stdout` and also into a file (specified as an argument). So `tee` is almost the same as `cat`, except that it has two identical outputs.

```
[student@linux pipes]$ ls
[student@linux pipes]$ cal | tee month.txt
    October 2024
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 31

[student@linux pipes]$ ls
month.txt
[student@linux pipes]$ cat month.txt
    October 2024
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 31
```

19.6. head and tail

The `head` and `tail` commands are used to display the first and last lines of a file, respectively. By default, they displays 10 lines, unless you specify another amount.

```
[student@linux pipes]$ head -3 count.txt
one
two
three
[student@linux pipes]$ tail -3 count.txt
three
four
five
```

With `tail -n+NUM` you can start at line `NUM`.

```
[student@linux pipes]$ tail -n+3 count.txt
three
four
five
```

With `head -n-NUM` you can stop at line `NUM`.

```
[student@linux pipes]$ head -n-3 count.txt
one
two
```

The `tail` command has an option `-f` to follow a file. That is, the command will display the last lines of the specified file, and then wait for new lines to be added to the file. This is useful for monitoring log files.

```
[student@linux ~]$ sudo tail -f /var/log/httpd/access_log
```

19.7. cut

The cut filter can select columns from files, depending on a delimiter or a count of bytes. The screenshot below uses cut to filter for the username and userid in the /etc/passwd file. It uses the colon as a delimiter, and selects fields 1 and 3.

```
[student@linux pipes]$ cut -d: -f1,3 /etc/passwd | tail -4
Figo:510
Pfaff:511
Harry:516
Hermione:517
```

When using a space as the delimiter for cut, you have to quote the space.

```
[student@linux pipes]$ cut -d' ' -f1 tennis.txt
Amelie
Kim
Justine
Serena
Venus
```

This example uses cut to display the second to the seventh character of /etc/passwd.

```
[student@linux pipes]$ cut -c2-7 /etc/passwd | tail -4
igo:x:
faff:x
arry:x
ermion
```

19.8. paste

The paste command will two files line by line. By default, it merges the first line of the first file with the first line of the second file, and so on. The screenshot below shows the result of merging two files.

```
student@linux:~/pipes$ cat count.txt
one
two
three
four
five
student@linux:~/pipes$ cat country.txt
France,Paris,60
Italy,Rome,50
Belgium,Brussels,10
Iran,Teheran,70
Germany,Berlin,100
student@linux:~/pipes$ paste -d',' count.txt country.txt
one,France,Paris,60
two,Italy,Rome,50
three,Belgium,Brussels,10
four,Iran,Teheran,70
five,Germany,Berlin,100
```

Option -d specifies a character to separate the columns. The default value is a tab character.

19.9. join

The `join` command is used to join two files on a common field. The common field in both files should be in the same order.

```
student@debian:~/pipes$ cat country-code.txt
Belgium,be
France,fr
Germany,de
Iran,ir
Italy,it
student@debian:~/pipes$ cat country-sorted.txt
Belgium,Brussels,10
France,Paris,60
Germany,Berlin,100
Iran,Teheran,70
Italy,Rome,50
student@debian:~/pipes$ join -t, country-code.txt country-sorted.txt
Belgium,be,Brussels,10
France,fr,Paris,60
Germany,de,Berlin,100
Iran,ir,Teheran,70
Italy,it,Rome,50
```

Option `-t` specifies the delimiter character.

19.10. sort

The `sort` filter will default to an alphabetical sort.

```
student@linux:~/pipes$ cat music.txt
Queen
Brel
Led Zeppelin
Abba
student@linux:~/pipes$ sort music.txt
Abba
Brel
Led Zeppelin
Queen
```

But the `sort` filter has many options to tweak its usage. This example shows sorting different columns (column 1 or column 2).

```
[student@linux pipes]$ sort -k1 country.txt
Belgium,Brussels,10
France,Paris,60
Germany,Berlin,100
Iran,Teheran,70
Italy,Rome,50
[student@linux pipes]$ sort -k2 country.txt
Germany,Berlin,100
Belgium,Brussels,10
France,Paris,60
Italy,Rome,50
Iran,Teheran,70
```

The screenshot below shows the difference between an alphabetical sort and a numerical sort (both on the third column).

```
[student@linux pipes]$ sort -k3 country.txt
Belgium,Brussels,10
Germany,Berlin,100
Italy,Rome,50
France,Paris,60
Iran,Teheran,70
[student@linux pipes]$ sort -n -k3 country.txt
Belgium,Brussels,10
Italy,Rome,50
France,Paris,60
Iran,Teheran,70
Germany,Berlin,100
```

19.11. uniq

With uniq you can remove duplicates from a *sorted list*.

```
student@linux:~/pipes$ cat music.txt
Queen
Brel
Queen
Abba
student@linux:~/pipes$ sort music.txt
Abba
Brel
Queen
Queen
student@linux:~/pipes$ sort music.txt | uniq
Abba
Brel
Queen
```

uniq can also count occurrences with the -c option.

```
student@linux:~/pipes$ sort music.txt |uniq -c
        1 Abba
        1 Brel
        2 Queen
```

19.12. fmt

Reformats text files to a specified width, preserving paragraphs and ensuring words are not split.

```
student@debian:~/pipes$ cat lorem.txt
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
 ↪  tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
 ↪  veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
 ↪  commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
 ↪  velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
 ↪  occaecat cupidatat non proident, sunt in culpa qui officia deserunt
 ↪  mollit anim id est laborum.
student@debian:~/pipes$ fmt -w 35 lorem.txt
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut
enim ad minim veniam, quis nostrud
exercitation ullamco laboris
nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor
in reprehenderit in voluptate
velit esse cillum dolore eu fugiat
nulla pariatur. Excepteur sint
occaecat cupidatat non proident,
sunt in culpa qui officia deserunt
mollit anim id est laborum.
```

19.13. nl

Add line numbers to input text.

```
student@debian:~/pipes$ nl count.txt
     1  one
     2  two
     3  three
     4  four
     5  five
```

19.14. wc

Counting words, lines and characters is easy with `wc`.

```
[student@linux pipes]$ wc tennis.txt
    5  15 100 tennis.txt
[student@linux pipes]$ wc -l tennis.txt
5 tennis.txt
[student@linux pipes]$ wc -w tennis.txt
15 tennis.txt
[student@linux pipes]$ wc -c tennis.txt
100 tennis.txt
[student@linux pipes]$
```

19.15. column

Arrange the input in columns. Option `-t` will create a nicely formatted table. Option `-s` specifies the delimiter of the input text (default is whitespace).

```
[student@linux pipes]$  column -t -s: < /etc/passwd | head -4
root      x    0    0    Super User    /root       /bin/bash
bin       x    1    1    bin           /bin        /usr/sbin/nologin
daemon    x    2    2    daemon        /sbin       /usr/sbin/nologin
adm       x    3    4    adm           /var/adm    /usr/sbin/nologin
```

It also allows you to emit JSON with option `-J`. Option `-N` specifies the column/key names.

```
[vagrant@fedora pipes]$ head -4 /etc/passwd | column -J -N
 ↪  user,passwd,uid,gid,name,home,shell -s:
{
   "table": [
      {
         "user": "root",
         "passwd": "x",
         "uid": "0",
         "gid": "0",
         "name": "Super User",
         "home": "/root",
         "shell": "/bin/bash"
      },{
         "user": "bin",
         "passwd": "x",
         "uid": "1",
         "gid": "1",
         "name": "bin",
         "home": "/bin",
         "shell": "/usr/sbin/nologin"
      },{
         "user": "daemon",
         "passwd": "x",
         "uid": "2",
         "gid": "2",
         "name": "daemon",
         "home": "/sbin",
         "shell": "/usr/sbin/nologin"
      },{
         "user": "adm",
         "passwd": "x",
         "uid": "3",
         "gid": "4",
         "name": "adm",
         "home": "/var/adm",
         "shell": "/usr/sbin/nologin"
      }
   ]
}
```

19.16. comm

Comparing streams (or files) can be done with the `comm`. By default `comm` will output three columns. In this example, Abba, Cure and Queen are in both lists, Bowie and Sweet are only in the first file, Turner is only in the second.

```
student@linux:~/pipes$ cat > list1.txt
Abba
Bowie
Cure
Queen
Sweet
student@linux:~/pipes$ cat > list2.txt
Abba
Cure
Queen
Turner
```

```
student@linux:~/pipes$ comm list1.txt list2.txt
                Abba
Bowie
                Cure
                Queen
Sweet
        Turner
```

The output of `comm` can be easier to read when outputting only a single column. The digits point out which output columns should not be displayed.

```
student@linux:~/pipes$ comm -12 list1.txt list2.txt
Abba
Cure
Queen
student@linux:~/pipes$ comm -13 list1.txt list2.txt
Turner
student@linux:~/pipes$ comm -23 list1.txt list2.txt
Bowie
Sweet
```

19.17. grep

The `grep` filter is famous among Unix users. The most common use of `grep` is to filter lines of text containing (or not containing) a certain text pattern. That pattern can be a simple string or a regular expression.

```
[student@linux pipes]$ cat tennis.txt
Amelie Mauresmo,Fra
Kim Clijsters,BEL
Justine Henin,Bel
Serena Williams,usa
Venus Williams,USA
[student@linux pipes]$ grep Williams tennis.txt
Serena Williams,usa
Venus Williams,USA
```

One of the most useful options of grep is `grep -i` which filters in a case insensitive way.

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

Another very useful option is `grep -v` which outputs lines *not* matching the string.

```
[student@linux pipes]$ grep -v Fra tennis.txt
Kim Clijsters,BEL
Justine Henin,Bel
Serena Williams,usa
Venus Williams,USA
```

And of course, both options can be combined to filter all lines not containing a case insensitive string.

```
[student@linux pipes]$ grep -vi usa tennis.txt
Amelie Mauresmo,Fra
Kim Clijsters,BEL
Justine Henin,Bel
```

With `grep -A1` one line `after` the result is also displayed.

```
student@linux:~/pipes$ grep -A1 Henin tennis.txt
Justine Henin,Bel
Serena Williams,usa
```

With `grep -B1` one line `before` the result is also displayed.

```
student@linux:~/pipes$ grep -B1 Henin tennis.txt
Kim Clijsters,BEL
Justine Henin,Bel
```

With `grep -C1` (context) one line `before` and one `after` are also displayed. All three options (A,B, and C) can display any number of lines (using e.g. A2, B4 or C20).

```
student@linux:~/pipes$ grep -C1 Henin tennis.txt
Kim Clijsters,BEL
Justine Henin,Bel
Serena Williams,usa
```

19.18. tr

The filter `tr` (short for *translate*) is used to translate *characters*. It's a bit different from the other filters, as it works on single characters, instead of lines of text. Also, it's not possible to specify a file as input, only `stdin`.

You can translate characters with `tr`. The screenshot shows the translation of all occurrences of e to E.

```
[student@linux pipes]$ cat tennis.txt | tr 'e' 'E'
AmEliE MaurEsmo,Fra
Kim ClijstErs,BEL
JustinE HEnin,BEl
SErEna Williams,usa
VEnus Williams,USA
```

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

```
[student@linux pipes]$ cat tennis.txt | tr 'a-z' 'A-Z'
AMELIE MAURESMO,FRA
KIM CLIJSTERS,BEL
JUSTINE HENIN,BEL
SERENA WILLIAMS,USA
VENUS WILLIAMS,USA
```

Here we translate all newlines to spaces.

```
[student@linux pipes]$ cat count.txt
one
two
three
four
five
[student@linux pipes]$ cat count.txt | tr '\n' ' '
one two three four five [student@linux pipes]$
```

The `tr -s` filter can also be used to squeeze multiple occurrences of a character to one.

```
[student@linux pipes]$ cat spaces.txt
one    two        three
        four   five  six
[student@linux pipes]$ cat spaces.txt | tr -s ' '
one two three
    four five six
```

You can also use `tr` to 'encrypt' texts with `rot13`.

```
[student@linux pipes]$ cat count.txt | tr 'a-z' 'nopqrstuvwxyzabcdefghijklm'
bar
gjb
guerr
sbhe
svir
[student@linux pipes]$ cat count.txt | tr 'a-z' 'n-za-m'
bar
gjb
guerr
sbhe
svir
```

This last example uses `tr -d` to delete characters.

```
student@linux:~/pipes$ cat tennis.txt | tr -d e
Amli Maursmo,Fra
Kim Clijstrs,BEL
Justin Hnin,Bl
Srna Williams,usa
Vnus Williams,USA
```

19.19. sed

The stream editor `sed` can perform editing functions in the stream, using *regular expressions*. It is very often used for search and replace operations.

The command `s` (short for substitute) takes a regular expression and a replacement string as arguments in the form `s/regexp/replacement/` and replaces the first occurrence of the regular expression on each line of input with the replacement string.

```
student@linux:~/pipes$ sed 's/5/42/' <<< 'level5 level7'
level42 level7
student@linux:~/pipes$ sed 's/level/jump/' <<< 'level5 level7'
jump5 level7
```

Add `g` for global replacements (all occurrences of the string per line).

```
student@linux:~/pipes$ sed 's/level/jump/g' <<< 'level5 level7'
jump5 jump7
```

With `/regex/d` you can remove lines from a stream matching the specified regular expression.

```
student@linux:~/filters$ cat tennis.txt
Venus Williams,USA
Martina Hingis,SUI
Justine Henin,BE
Serena williams,USA
Kim Clijsters,BE
Yanina Wickmayer,BE
```

```
student@linux:~/filters$ cat tennis.txt | sed '/BE/d'
Venus Williams,USA
Martina Hingis,SUI
Serena williams,USA
```

There are many more options for `sed`, but they are beyond the scope of this chapter. Check e.g. this list of sed one-liners and do an Internet search for more examples.

19.20. awk

Maybe `awk` does not really belong in a list of filters "that do one specific thing", as it is a complete programming language. But it is often used in the same way as the other filters, so we give at least a few examples. Just like `sed`, `awk` is a very powerful tool, and we can only scratch the surface here. See the GNU Awk User's Guide for in-depth information, or find examples like this list of awk one-liners.

AWK was developed in the late 1970's. The name stands for the last names of its authors: Alfred Aho, Peter Weinberger, and Brian Kernighan. If the last name sounds familiar, it's because Brian Kernighan he is also co-authors of the C programming language and contributed to the development of Unix. Awk is a part of the POSIX standard and therefore is always available on Unix-like systems.

You can consider AWK as a programming language with a built-in for loop. For each line of input, it will perform the specified action (written out in a C-like syntax). The action is enclosed in curly braces `{}`. Awk also automatically splits each line into whitespace delimited fields, which can be accessed with `$1`, `$2`, etc. The whole line is `$0`.

The following example prints the first and third field of each line of the `passwd` file. Since the fields are separated by colons, we use `-F:` to specify the field separator. Compare to the `cut` example above!

```
student@linux:~/pipes$ awk -F: '{print $1, $3}' /etc/passwd | tail -4
Figo 510
Pfaff 511
Harry 516
Hermione 517
```

You can apply an action only to lines matching a regular expression. This example prints the first and third field of each line of the `passwd` file, but only for lines ending on the string `bash`.

```
student@linux:~/pipes$ awk -F: '/bash$/ {print $1, $3}' /etc/passwd
root 0
student 1000
paul 1001
serena 1002
```

Awk automatically interprets numbers as numbers, so you can do calculations with them. This example prints the first and third field of each line of the `passwd` file, but only for lines where the third field (UID) is greater than or equals to 1000 (i.e. "normal" users).

```
student@linux:~/pipes$ awk -F: '$3 >= 1000 {print $1, $3}' /etc/passwd
student 1000
paul 1001
serena 1002
```

Awk can also be used to perform calculations.

```
student@linux:~/pipes$ cat country.txt
France,Paris,60
Italy,Rome,50
Belgium,Brussels,10
Iran,Teheran,70
Germany,Berlin,100
student@linux:~/pipes$ awk -F, '{ sum += $3 } END { print sum/NR }'
 ↪  country.txt
58
```

A lot is going on here. For each line of input, the numerical value of column 3 is added to the variable `sum`. This variable is implicitly initialized to 0. The `END` block is executed after all lines have been processed. It prints the value of `sum` divided by the number of records (lines) processed, which is stored in the built-in variable `NR`. So, this one-liner calculates the average of the third column of the file `country.txt`.

19.21. pipe examples

19.21.1. who | wc

How many users are logged on to this system ?

```
[student@linux pipes]$ who
root     tty1         Jul 25 10:50
paul     pts/0        Jul 25 09:29 (laika)
Harry    pts/1        Jul 25 12:26 (barry)
paul     pts/2        Jul 25 12:26 (pasha)
[student@linux pipes]$ who | wc -l
4
```

19.21.2. who | cut | sort

Display a sorted list of logged on users.

```
[student@linux pipes]$ who | cut -d' ' -f1 | sort
Harry
paul
paul
root
```

Display a sorted list of logged on users, but every user only once .

```
[student@linux pipes]$ who | cut -d' ' -f1 | sort | uniq
Harry
paul
root
```

19.21.3. grep | cut

Display a list of all *user accounts* on this computer. Users accounts are explained in in the chapters about user management.

```
student@linux:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
paul:x:1000:1000:paul,,,:/home/paul:/bin/bash
serena:x:1001:1001::/home/serena:/bin/bash
student@linux:~$ grep bash /etc/passwd | cut -d: -f1
root
paul
serena
```

19.21.4. ip | awk

Display only IPv4 addresses of this computer.

```
student@debian:~/pipes$ ip -br a
lo       UNKNOWN     127.0.0.1/8 ::1/128
eth0     UP          10.0.2.15/24 fe80::a00:27ff:fee6:f5c9/64
eth1     UP          192.168.56.21/24 fe80::a00:27ff:fe0f:afa/64
student@debian:~/pipes$ ip -br a | awk '{print $3}'
127.0.0.1/8
10.0.2.15/24
192.168.56.21/24
```

19.22. practice: filters

1. Put a sorted list of all bash users in bashusers.txt.
2. Put a sorted list of all logged on users in onlineusers.txt.
3. Make a list of all filenames in `/etc` that contain the string `conf` in their filename.
4. Make a sorted list of all files in `/etc` that contain the case insensitive string `conf` in their filename.
5. Look at the output of `ip neigh` (ARP cache, listing other hosts on the same local network) and filter out the MAC addresses.
6. Write a line that removes all non-letters from a stream.
7. Write a line that receives a text file, and outputs all words on a separate line.
8. Write a spell checker on the command line, i.e. list all words in a file that do not occur in a dictionary. There will probably be a dictionary in `/usr/share/dict/`.

19.23. solution: filters

1. Put a sorted list of all bash users in bashusers.txt.

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

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

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

3. Make a list of all filenames in `/etc` that contain the string `conf` in their filename.

```
ls /etc | grep conf
```

4. Make a sorted list of all files in `/etc` that contain the case insensitive string `conf` in their filename.

```
ls /etc | grep -i conf | sort
```

5. Look at the output of `ip neigh` (ARP cache, listing other hosts on the same local network) and filter out the MAC addresses.

```
ip n | cut -d' ' -f5
```

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

```
student@linux:~$ cat text
This is, yes really! , a text with ?&* too many str$ange# characters ;-)
student@linux:~$ tr -d ',!$?.*&^%#@;()-' < text
This is yes really  a text with  too many strange characters
```

7. Write a line that receives a text file, and outputs all words on a separate line.

```
student@linux:~$ cat text2
it is very cold today without the sun

student@linux:~$ tr ' ' '\n' < text2
it
is
very
cold
today
without
the
sun
```

8. Write a spell checker on the command line, i.e. list all words in a file that do not occur in a dictionary. There will probably be a dictionary in `/usr/share/dict/`.

   We will use the Python 3 copyright notice as example input. The dictionary is `/usr/share/dict/words`. We will use `comm` to compare the two lists. To make the search case insensitive, we will convert both lists to lowercase.

```
student@linux:~/pipes$ tr 'A-Z' 'a-z' < /usr/share/dict/words | sort |
    uniq > lcase-dict.txt
```

   Next, we make a list of all words in the input file, removing all punctuation and digits.

```
student@linux:~/pipes$ tr -d '[:punct:][:digit:]' <
    /usr/share/doc/python3/copyright | tr '[A-Z] ' '[a-z]\n' | sort |
    uniq > words.txt
```

   Finally, we compare the two lists.

```
student@debian:~/pipes$ comm -23 words.txt lcase-dict.txt
andor
beopen
beopencom
bernd
brentrup
bsbunimuensterde
centrum
cnri
[ ... some lines omitted ... ]
tortious
virginias
zope
```

20. 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`.

20.1. 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 {} \;
```

20.2. 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.

20.3. 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
```

20.4. 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
```

20.5. 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 ~$
```

20.6. 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
```

20.7. 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
```

20.8. 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
```

20.9. 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
```

20.10. 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
```

20.11. 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 ?

20.12. 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

21. 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`.

21.1. 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!`

21.2. grep

21.2.1. 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.

21.2.2. 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 Valentina and `in` will match Valent`in`a but not Tania.

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

21.2.3. 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
```

21.2.4. 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:~$
```

21.2.5. 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
```

21.2.6. 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.

21.2.7. 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:~$
```

21.2.8. 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
```

21.2.9. 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
```

21.3. rename

21.3.1. 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, ...).
```

21.3.2. 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
```

21.3.3. 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.

21.3.4. 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).

21.3.5. 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$
```

21.3.6. 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.

21.4. sed

21.4.1. 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
```

21.4.2. 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
```

21.4.3. 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
```

21.4.4. 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
```

21.4.5. 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
```

21.4.6. 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`.

21.4.7. 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
```

21.4.8. 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
```

21.4.9. 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:~$
```

21.4.10. 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:~$
```

21.5. 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 VI.

Vi

22. Introduction to vi

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

The `vi` editor is installed on almost every Unix. Linux will very often install `vim` (`vi improved`) which is similar. Every system administrator should know `vi(m)`, because it is an easy tool to solve problems.

The `vi` editor is not intuitive, but once you get to know it, `vi` becomes a very powerful application. Most Linux distributions will include the `vimtutor` which is a 45 minute lesson in `vi(m)`.

22.1. command mode and insert mode

The vi editor starts in `command mode`. In command mode, you can type commands. Some commands will bring you into `insert mode`. In insert mode, you can type text. The `escape key` will return you to command mode.

Table 22.1.: getting to command mode

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

22.2. start typing (a A i I o O)

The difference between a A i I o and O is the location where you can start typing. a will append after the current character and A will append at the end of the line. i will insert before the current character and I will insert at the beginning of the line. o will put you in a new line after the current line and O will put you in a new line before the current line.

Table 22.2.: switch to insert mode

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

22.3. replace and delete a character (r x X)

When in command mode (it doesn't hurt to hit the escape key more than once) you can use the x key to delete the current character. The big X key (or shift x) will delete the character left of the cursor. Also when in command mode, you can use the r key to replace one single character. The r key will bring you in insert mode for just one key press, and will return you immediately to command mode.

Table 22.3.: replace and delete

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

22.4. undo, redo and repeat (u .)

When in command mode, you can undo your mistakes with u. Use `ctrl-r` to redo the undo.

You can do your mistakes twice with . (in other words, the . will repeat your last command).

Table 22.4.: undo and repeat

command	action
u	undo the last action
ctrl-r	redo the last undo
.	repeat the last action

22.5. cut, copy and paste a line (dd yy p P)

When in command mode, dd will cut the current line. yy will copy the current line. You can paste the last copied or cut line after (p) or before (P) the current line.

Table 22.5.: cut, copy and paste a line

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

22.6. cut, copy and paste lines (3dd 2yy)

When in command mode, before typing dd or yy, you can type a number to repeat the command a number of times. Thus, 5dd will cut 5 lines and 4yy will copy (yank) 4 lines. That last one will be noted by vi in the bottom left corner as "4 line yanked".

Table 22.6.: cut, copy and paste lines

command	action
3dd	cut three lines
4yy	copy four lines

22.7. start and end of a line (0 or ^ and $)

When in command mode, the 0 and the caret ^ will bring you to the start of the current line, whereas the $ will put the cursor at the end of the current line. You can add 0 and $ to the d command, d0 will delete every character between the current character and the start of the line. Likewise d$ will delete everything from the current character till the end of the line. Similarly y0 and y$ will yank till start and end of the current line.

Table 22.7.: start and end of line

command	action
0	jump to start of current line
^	jump to start of current line
$	jump to end of current line
d0	delete until start of line
d$	delete until end of line

22.8. join two lines (J) and more

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

Table 22.8.: join two lines

command	action
J	join two lines
yyp	duplicate a line
ddp	switch two lines

22.9. words (w b)

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

Table 22.9.: words

command	action
w	forward one word
b	back one word
3w	forward three words
dw	delete one word
yw	yank (copy) one word

command	action
5yb	yank five words back
7dw	delete seven words

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

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

Table 22.10.: save and exit vi

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

The last one is a bit special. With **`:w! vi`** will try to `chmod` the file to get write permission (this works when you are the owner) and will `chmod` it back when the write succeeds. This should always work when you are root (and the file system is writable).

22.11. Searching (/ ?)

When in command mode typing / will allow you to search in vi for strings (can be a regular expression). Typing /foo will do a forward search for the string foo and typing ?bar will do a backward search for bar.

Table 22.11.: searching

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

22.12. replace all (:1,$ s/foo/bar/g)

To replace all occurrences of the string foo with bar, first switch to ex mode with : . Then tell vi which lines to use, for example 1,$ will do the replace all from the first to the last line. You can write 1,5 to only process the first five lines. The s/foo/bar/g will replace all occurrences of foo with bar.

Table 22.12.: replace

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

22.13. reading files (:r :r !cmd)

When in command mode, :r foo will read the file named foo, :r !foo will execute the command foo. The result will be put at the current location. Thus :r !ls will put a listing of the current directory in your text file.

Table 22.13.: read files and input

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

22.14. text buffers

There are 36 buffers in vi to store text. You can use them with the `"` character.

Table 22.14.: text buffers

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

22.15. multiple files

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

Table 22.15.: multiple files

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

22.16. abbreviations

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

Table 22.16.: abbreviations

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

22.17. key mappings

Similarly to their abbreviations, you can use mappings with `:map` for command mode and `:map!` for insert mode.

This example shows how to set the F6 function key to toggle between `set number` and `set nonumber`. The <bar> separates the two commands, `set number!` toggles the state and `set number?` reports the current state.

```
:map <F6> :set number!<bar>set number?<CR>
```

22.18. setting options

Some options that you can set in vim.

```
:set number  ( also try :se nu )
:set nonumber
:syntax on
:syntax off
:set all  (list all options)
:set tabstop=8
:set tx   (CR/LF style endings)
:set notx
```

You can set these options (and much more) in `~/.vimrc` for vim or in `~/.exrc` for standard vi.

```
student@linux:~$ cat ~/.vimrc
set number
set tabstop=8
set textwidth=78
map <F6> :set number!<bar>set number?<CR>
student@linux:~$
```

22.19. practice: vi(m)

1. Start the vimtutor and do some or all of the exercises. You might need to run `aptitude install vim` on xubuntu.

2. What 3 key sequence in command mode will duplicate the current line.

3. What 3 key sequence in command mode will switch two lines' place (line five becomes line six and line six becomes line five).

4. What 2 key sequence in command mode will switch a character's place with the next one.

5. vi can understand macro's. A macro can be recorded with q followed by the name of the macro. So qa will record the macro named a. Pressing q again will end the recording. You can recall the macro with @ followed by the name of the macro. Try this example: i 1 'Escape Key' qa yyp 'Ctrl a' q 5@a (Ctrl a will increase the number with one).

6. Copy /etc/passwd to your ~/passwd. Open the last one in vi and press Ctrl v. Use the arrow keys to select a Visual Block, you can copy this with y or delete it with d. Try pasting it.

7. What does dwwP do when you are at the beginning of a word in a sentence ?

22.20. solution: vi(m)

1. Start the vimtutor and do some or all of the exercises. You might need to run `aptitude install vim` on xubuntu.

```
vimtutor
```

2. What 3 key sequence in command mode will duplicate the current line.

```
yyp
```

3. What 3 key sequence in command mode will switch two lines' place (line five becomes line six and line six becomes line five).

```
ddp
```

4. What 2 key sequence in command mode will switch a character's place with the next one.

```
xp
```

5. vi can understand macro's. A macro can be recorded with q followed by the name of the macro. So qa will record the macro named a. Pressing q again will end the recording. You can recall the macro with @ followed by the name of the macro. Try this example: i 1 'Escape Key' qa yyp 'Ctrl a' q 5@a (Ctrl a will increase the number with one).

6. Copy /etc/passwd to your ~/passwd. Open the last one in vi and press Ctrl v. Use the arrow keys to select a Visual Block, you can copy this with y or delete it with d. Try pasting it.

```
cp /etc/passwd ~
vi passwd
(press Ctrl-V)
```

7. What does `dwwP` do when you are at the beginning of a word in a sentence ?

`dwwP` can switch the current word with the next word.

Part VII.

Scripting

23. introduction to scripting

(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/)

The goal of this chapter is to give you all the information in order to read, write and understand small, long and complex shell scripts.

You should have basic understanding on how the shell works, shell variables, globbing, I/O redirection and filters to understand all content in this chapter.

23.1. introduction

When you open a terminal and type a command, you are using a *shell*, an interactive environment that interprets your commands, executes them, and shows you the output the command generates. Most Linux distributions have Bash (the “Bourne Again Shell”) as the default, but there are others as well: the original “Bourne shell” (`sh`), the “Debian Amquist Shell” (`dash`, a modern implementation of `sh`), the “Korn shell” (`ksh`), the “C shell” (`csh`), and the “Z shell” (`zsh`), to name a few.

A sequence of commands can be saved in a file and executed as a single command. This is called a *script*. Shell scripts are used to automate tasks, and are an essential tool for system administrators and developers. Subsequently, this means that system administrators or SysOps also need solid knowledge of *scripting* to understand how their servers and their applications are started, updated, upgraded, patched, maintained, configured and removed, and also to understand how a user environment is built.

Shells have also support for programming constructs (like loops, functions, variables, etc.) so that you can write more complex scripts. This makes a scripting language basically as powerful as a programming language. Scripting languages are often interpreted, rather than compiled.

If you copy a script to one of the `bin` directories (e.g. `/usr/local/bin`), you can execute it from the command line just like any other command. In fact, many UNIX/Linux commands are essentially `scripts`. You can check this for yourself by executing the `file` command on the executables in the `/bin` directory. For example:

```
student@linux:~$ file /usr/bin/* | awk '{ print($2, $3, $4) }' \
  | sort | uniq -c | sort -nr
    466 ELF 64-bit LSB
    168 symbolic link to
     74 POSIX shell script,
     71 Perl script text
     14 Python script, ASCII
     10 setuid ELF 64-bit
      7 setgid ELF 64-bit
      6 Bourne-Again shell script,
      2 Python script, Unicode
      1 Python script, ISO-8859
```

We find POSIX (Bourne), Bash, Perl and Python scripts, as well as ELF binaries (compiled programs). This shows that a significant portion of the commands in a typical Linux system are actually scripts.

Bash scripting is a valuable skill for any Linux user, but these days, its applications are no longer limited to Linux. Bash is also present on macOS (albeit an older version), and with the advent of Windows Subsystem for Linux (WSL), Bash is now available for Windows users as well. Moreover, Git Bash, a Bash shell for Windows, is also available.

23.2. hello world

Just like in every programming course, we start with a simple `hello_world` script. The following script will output `Hello World`.

```
echo Hello World
```

After creating this simple script in `nano`, `vi`, or with `echo`, you'll have to `chmod +x hello_world` to make it executable. And unless you add the scripts directory to your path, you'll have to type the path to the script for the shell to be able to find it.

```
student@linux:~$ echo echo Hello World > hello_world
student@linux:~$ chmod +x hello_world
student@linux:~$ ./hello_world
Hello World
student@linux:~$
```

23.3. she-bang

Let's expand our example a little further by putting `#!/bin/bash` on the first line of the script. The `#!` is called a `she-bang` (sometimes called `sha-bang`), where the `she-bang` is the first two characters of the script.

Open the file with `nano hello_world` or `vi hello_world` and add the following line at the top of the file.

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

You can never be sure which (interactive) shell a user is running. A script that works flawlessly in `bash` might not work in `ksh`, `csh`, or `dash`. To instruct a shell to run your script with a specific interpreter, you should start your script with a `she-bang` followed by the absolute path to the executable of the interpreter.

This script will run in a bash shell.

```
#!/bin/bash
echo -n hello
echo A bash subshell $(echo -n hello)
```

This script will be interpreted by Python:

```
#!/usr/bin/env python3
print("Hello World!")
```

The following script will run in a Korn shell (unless `/bin/ksh` is a hard link to `/bin/bash`). The `/etc/shells` file contains a list of shells available on your system. Check it to see which ones are available to you

```
#!/bin/ksh
echo -n hello
echo a Korn subshell $(echo -n hello)
```

If you're not sure in which `bin` directory the shell executable is located,you can use `env`. The command `env` is normally used to print environment variables, but in the context of a script, it is used to launch the correct interpreter.

```
#!/usr/bin/env bash
echo -n hello
echo A bash subshell $(echo -n hello)
```

This is particularly useful for macOS users: out-of-the-box, a macOS system has a very old version of `bash` in `/bin/bash`. If you want to use a more recent version, you can install it with Homebrew, that will put it in `/usr/local/bin/bash`. If you use `#!/usr/bin/env bash` in your scripts, the newer version will be used.

23.4. comments

When writing Bash scripts, it is always a good practice to make your code clean and easily understandable. Organizing your code in blocks, indenting, giving variables and functions descriptive names are several ways to do this. Another way to improve the readability of your code is by using comments. A comment is a human-readable explanation or annotation that is written in the shell script.

Let's expand our example a little further by adding comment lines.

```
#!/usr/bin/env bash
#
# hello_world.sh -- My first script
#
echo Hello World

# this is old way of calling for subshell with backtick ``
echo A bash subshell `echo -n hello`

# this is more modern way of calling for subshell with dollar and brackets
 ↪ $()
echo A bash subshell $(echo -n hello)

#NOTICE: backtick might not work in future versions of bash shell
```

23.5. extension

A general convention is to give files an extension that indicates the file type. On a Linux system, this is not strictly necessary. Remember that you can always use the `file` command to determine the type of a file by scanning its contents. The system will not care if you call your script `hello_world.sh` or `hello_world`. However, it is a good practice to use an extension, as it makes it easier to identify the type of file.

We recommend to always give your scripts the `.sh` extension, but to remove the extension when you install it in a `bin` directory as a command.

23.6. shell variables

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

```
#!/bin/bash
# hello-user.sh -- example of a shell variable in a script
echo "Hello ${USER}"
```

In Bash, you can access the value of a variable by prefixing the variable name with the $ sign. The braces are not mandatory in this case, but they are a good practice to avoid ambiguity. In some cases they are required, so it's best to be consistent in your coding style.

The variable `${USER}` is a shell variable that is defined by the system when you log in.

```
student@linux:~$ chmod +x hello-user.sh
student@linux:~$ ./hello-user.sh
Hello student
```

23.7. variable assignment

Assigning a variable is done by using the = operator. The variable name must start with a letter or an underscore, and can contain only letters, digits, or underscores. Remark that spaces are not allowed around the = sign!

```
#!/bin/bash
# hello-var.sh -- example of variable assignment
user="Tux"

echo "Hello ${user}"
```

Because variable names are case-sensitive, this variable `${user}` is different from `${USER}` in the previous example!

Tip: naming convention. You can use any name for a variable, but it is a good practice to use all uppercase letters for environment variables (e.g. `${USER}`) and constants and all lowercase letters for local variables (e.g. `${user}`). This is also recommended by the Google Shell Style Guide. If a variable consists of multiple words, use underscores to separate them (e.g. `${current_user}`).

Running the script:

```
student@linux:~$ chmod +x hello-var.sh
student@linux:~$ ./hello-var.sh
Hello Tux
```

Scripts can contain variables, but since scripts are run in their own subshell, the variables do not survive the end of the script.

```
student@linux:~$ echo $user

student@linux:~$ ./hello-var.sh
Hello Tux
student@linux:~$ echo $user

student@linux:~$
```

23.8. unbound variables

Remove the line `user="Tux"` from the script, or comment out the line and run it again. What do you expect to happen if the variable user is not assigned, but we try to use it in the script?

```
student@linux:~$ ./hello-var.sh
Hello
```

Bash will not complain if you use a variable that is not assigned, but it will simply replace the variable with an empty string. This can lead to unexpected results and is a common cause of bugs that can be hard to find. However, you can change the behavior of the shell by starting your scripts with the command `set -o nounset` (or shorter: `set -u`). This will cause the script to exit with an error if you try to use an unassigned variable.

Add the line to the script, right below the comment lines and try again!

```
#!/bin/bash
# hello-var.sh -- example of variable assignment

set -o nounset

echo "Hello ${user}"
```

Running the script:

```
student@linux:~$ ./hello-var.sh
./hello-var.sh: line 6: user: unbound variable
```

This is what you want to see. The script exits with an error, and you can see the line number where the error occurred and which variable is unbound. Start all your scripts with `set -o nounset` to prevent this kind of error!

23.9. sourcing a script

Luckily, you can force a script to run in the same shell; this is called `sourcing` a script.

```
student@linux:~$ source hello-var.sh
Hello Tux
student@linux:~$ echo $name
Tux
```

Instead of `source`, you can use the `.` (dot) command.

```
student@linux:~$ . hello-var.sh
Hello Tux
student@linux:~$ echo $name
Tux
```

23.10. quoting

Go back to `hello-user.sh` and replace the double quotes with single quotes:

```
#!/bin/bash
# hello-user.sh -- example of a shell variable in a script
echo 'Hello ${USER}'
```

Run the script again:

```
student@linux:~$ ./hello-user.sh
Hello ${USER}
```

What happened? By using single quotes, we turned off the shell's variable expansion. The shell will not replace `${USER}` with the value of the `USER` variable. This is why you should use double quotes when you want to use a variable.

Using quotes is important. Most of the times, when you reference the value of a variable, you should enclose it in double quotes. To illustrate this, write the following script:

```
#!/bin/bash
# create-file.sh -- example of using quotes
file="my file.txt"
touch $file
```

What we expect is that the script will create a file called `my file.txt`. However, when we run the script:

```
student@linux:~$ ./create-file.sh
student@linux:~$ ls -l
total 4
-rwxr-xr-x 1 student student     88 Mar  6 16:20 create-file.sh
-rw-r--r-- 1 student student      0 Mar  6 16:20 file.txt
-rw-r--r-- 1 student student      0 Mar  6 16:20 my
```

So actually two files were created, one named `my` and the other `file.txt`. The reason has to do with the way Bash interprets a command and how it substitutes variables. The line

```
touch $file
```

is expanded to

```
touch my file.txt
```

without the quotes. The `touch` command now sees two arguments, `my` and `file.txt`, and creates two files. To fix this, you should always use double quotes:

```
#!/bin/bash
# create-file.sh -- example of using quotes
file="my file.txt"
touch "${file}"
```

Now the expansion of the variable is done within the quotes, and the `touch` command sees only one argument.

```
student@linux:~$ ./create-file.sh
student@linux:~$ ls -l
total 4
-rwxr-xr-x 1 student student     92 Mar  6 16:20 create-file.sh
-rw-r--r-- 1 student student      0 Mar  6 16:20 'my file.txt'
```

23.11. backticks and command substitution

In Bash, backquotes (`` ` ``), also called *backticks*, also have a specific use. However, they are considered deprecated and should be avoided. The reason is that they are hard to read, especially when combined with single or double quotes. Instead, you should use the `$()` syntax, which is easier to read and nest.

This syntax is called *command substitution*. The shell will execute the command inside the parentheses and replace the command with the output of the command.

```
[vagrant@el scripts]$ current_date=$(date)
[vagrant@el scripts]$ echo $current_date
Sun Oct 27 02:40:44 PM UTC 2024
```

In this example, the `date` command is executed and the output is stored in the variable `current_date`.

23.12. troubleshooting a script

In this section, we discuss a few techniques that can help you to troubleshoot a script. Some general guidelines are:

- **Start with a simple script** and execute it as often as possible.
- Make **small changes** and test them immediately.
- Have your script **print out debugging information**. The output can help identify where the script is failing.
- Open your text editor and a terminal **side by side**. It's easier to interpret how the script works if you see the output and the code at the same time.

23.12.1. bash -n

You can **check the syntax** of a shell script by using the `bash` command with the `-n` option. This option causes the shell to parse the script and check for syntax errors, but it does not execute the script.

Take a look at the following script (that doesn't do anything useful):

```
#! /bin/bash

file_name='my file.txt'

if[ $# -ne 1 ]; then
    echo "One argument expected, got $#"
fi

touch $file_name
```

It has several problematic mistakes. The `if` command and square bracket `[` are not separated by a space, and the variable `$file_name` is not enclosed in double quotes (which will cause word splitting).

When we check the syntax of the script, we get the following output:

```
[vagrant@el scripts]$ bash -n syntax.sh
syntax.sh: line 5: syntax error near unexpected token `then'
syntax.sh: line 5: `if[ $# -ne 1 ]; then'
```

Remark that the first problem is reported (albeit with an unhelpful error message), but the other is not. In the following sections, we will give additional tips to also catch these kinds of errors.

23.12.2. shellcheck

ShellCheck is a static analyzer for shell scripts. It shows common errors and mistakes in your scripts that can't be caught by a syntax check with `bash -n`. You can use it online on the tool's website, install it as a command-line tool, and it is available as a plugin for many text editors (including Vim, VS Code, etc.). If we try it on the `syntax.sh` script, we get the following output:

```
[vagrant@el scripts]$ shellcheck syntax.sh

In syntax.sh line 5:
if[ $# -ne 1 ]; then
  ^-- SC1069 (error): You need a space before the [.


In syntax.sh line 9:
touch $file_name
      ^--------^ SC2086 (info): Double quote to prevent globbing and word
 ↪  splitting.

Did you mean:
touch "$file_name"

For more information:
  https://www.shellcheck.net/wiki/SC1069 -- You need a space before the [.
  https://www.shellcheck.net/wiki/SC2086 -- Double quote to prevent globbing
 ↪  ...
```

The error message for the first problem is much more helpful than the one given by `bash -n`. The second problem is also reported, and a suggestion is given to fix it. The links on the last lines of output point to a Wiki with more information about the errors and tips on how to write better code.

Using `shellcheck` in your workflow immediately helps you to improve your scripting skills, so you should always use it!

23.12.3. show expanded commands

Another way to run a script in a separate shell is by typing `bash` with the name of the script as a parameter. Expanding this to `bash -x` allows you to see the commands that the shell is executing (after shell expansion).

Try this with the `create-file.sh` script introduced earlier. The incorrect version without the quotes:

```
$ bash -x create-file.sh
+ file='my file.txt'
+ touch my file.txt
```

Notice the absence of the commented (#) line, and the replacement of the variable in the argument `touch`.

After the fix, you get:

```
$ bash -x create-file.sh
+ file='my file.txt'
+ touch 'my file.txt'
```

Do you notice the difference?

In longer scripts, this setting produces a lot of output, which may be hard to read. You can limit the output to a specific problematic part of your script by using `set -x` and `set +x` to turn the debugging on and off, respectively.

```
#!/bin/bash
# create-file.sh -- example of using quotes
file="my file.txt"

set -x
touch "${file}"
set +x
```

23.12.4. bash's "strict mode"

Apart from the `nounset` shell option, there are two other options that are very useful for debugging scripts: `set -o errexit` (or `set -e`) and `set -o pipefail`. The first option causes the script to exit with an error if any command fails. The second option gives better error messages when a command in a pipeline fails.

Start all your scripts with the following lines to prevent errors and to make debugging easier:

```
#!/bin/bash --
set -o nounset
set -o errexit
set -o pipefail
```

This is called "strict mode" by some. You can write this shorter in one line as `set -euo pipefail`, but this is less readable.

23.13. prevent setuid root spoofing

Some user may try to perform `setuid` based script `root spoofing`. This is a rare but possible attack. To improve script security and to avoid interpreter spoofing, you need to add -- after the `#!/bin/bash`, which disables further option processing so the shell will not accept any options.

```
#!/usr/bin/env bash -
or
#!/usr/bin/env bash --
```

Any arguments after the -- are treated as filenames and arguments. An argument of - is equivalent to --.

23.14. practice: introduction to scripting

1. Write a Python "Hello World" script, give it a shebang and make it executable. Execute it like you would a shell script and verify that this works.
2. What would happen if you remove the shebang and try to execute the script again?
3. Create a Bash script `greeting.sh` that says hello to the user (make use of the shell variable with the current user's login name), prints the current date and time, and prints a quote, e.g.:

```
student@linux:~$ ./greeting.sh
Hello student, today is:
Wed Mar  6 09:04:19 PM UTC 2024
Quote of the day:
 ______________________________________
/ Having nothing, nothing can he lose. \
|                                      |
\ -- William Shakespeare, "Henry VI"   /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Ensure that you apply the shell settings to make your script easier to debug.

4. Copy the script to `/usr/local/bin` without the extension and verify that you can run it from any directory as a command.

5. Take another look at the script `hello-var.sh` where we printed a variable that was not assigned:

```bash
#!/bin/bash
# hello-var.sh -- example of variable assignment
# user="Tux" # Remark: this line is commented out

echo "Hello ${user}"
```

   What happens if you assign the value `Tux` to the variable `user` on the interactive shell and then run the script? What do we have to do to make sure the variable is available in the script?

6. What if we change the value of the variable `user` in the script? Will this change affect the value of the variable in the interactive shell after the script is finished?

23.15. solution: introduction to scripting

1. Write a Python Hello World script, give it a shebang and make it executable.

```python
#!/usr/bin/python3
print("Hello, World!")
```

```
$ chmod +x hello.py
$ ./hello.py
Hello, World!
```

2. What would happen if you remove the shebang and try to execute the script again?

   The script will be executed by the default interpreter, in this case, the Bash shell, which will not understand the Python syntax.

```
$ ./hello.py
./hello.py: line 1: syntax error near unexpected token `"Hello world!"'
./hello.py: line 1: `print("Hello world!")'
```

3. Create a Bash script `greeting.sh` that says hello to the user (make use of the shell variable with the current user's login name), prints the current date and time, and prints a quote. Ensure that you apply the shell settings to make your script easier to debug.

```bash
#! /bin/bash --

set -o nounset
set -o errexit
set -o pipefail

echo "Hello ${USER}, today is:"
date
echo "Quote of the day:"
fortune | cowsay
```

4. Copy the script to `/usr/local/bin` without the extension and verify that you can run it from any directory as a command.

```
student@linux:~$ sudo cp greeting.sh /usr/local/bin/greeting
student@linux:~$ greeting
Hello student, today is:
Wed Mar  6 09:17:00 PM UTC 2024
Quote of the day:
 ______________________________________
/ You plan things that you do not even \
| attempt because of your extreme      |
\ caution.                             /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
student@linux:~$ cd /tmp
student@linux:/tmp$ greeting
Hello student, today is:
Wed Mar  6 09:17:08 PM UTC 2024
Quote of the day:
 _________________________________
< You will be successful in love. >
 ---------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

5. Take another look at the script `hello-var.sh` where we printed a variable that was not assigned. What happens if you assign the value Tux to the variable `user` on the interactive shell and then run the script? What do we have to do to make sure the variable is available in the script?

```
student@linux:~$ ./hello-var.sh
Hello
student@linux:~$ user=Tux
student@linux:~$ ./hello-var.sh
Hello
student@linux:~$ export user
student@linux:~$ ./hello-var.sh
Hello Tux
```

6. What if we change the value of the variable `user` in the script? Will this change affect the value of the variable in the interactive shell after the script is finished?

We change the script to:

```
#!/bin/bash
# hello-var.sh -- example of variable assignment
user="Linus"

echo "Hello ${user}"
```

And execute it:

```
student@linux:~$ export user=Tux
student@linux:~$ echo $user
Tux
student@linux:~$ ./hello-var.sh
Hello Linus
student@linux:~$ echo $user
Tux
```

The change in the script does not affect the value of the variable in the interactive shell after the script is finished!

24. scripting loops

(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/)

24.1. test

The `test` command can evaluate **logical expressions** and indicate through their *exit status* whether the expression was *true* or *false*. In Bash, boolean values *do not exist* as a data type and are represented by the exit status of commands. *True* is represented by the exit status `0` (denoting that the command finished successfully), and *false* is represented by any other exit status (1-255, denoting any failure).

Let's start by testing whether 10 is greater than 55, and then show the exit status.

```
[student@linux ~]$ test 10 -gt 55 ; echo $?
1
```

The test command returns 1 if the test fails. And as you see in the next screenshot, test returns 0 when a test succeeds.

```
[student@linux ~]$ test 56 -gt 55 ; echo $?
0
```

If you prefer true and false, then write the test like this.

```
[student@linux ~]$ test 56 -gt 55 && echo true || echo false
true
[student@linux ~]$ test 6 -gt 55 && echo true || echo false
false
```

24.2. square brackets [ ]

The test command can also be written as square brackets. In this case, the final argument must be a closing square bracket].

The screenshot below is identical to the one above.

```
[student@linux ~]$ [ 56 -gt 55 ] && echo true || echo false
true
[student@linux ~]$ [ 6 -gt 55 ] && echo true || echo false
false
```

Remark that the square bracket is a **command** that takes arguments like any other command. Consequently, you **must** put a space between the square bracket and the arguments. You can check this fact by looking at the contents of the `/bin` directory.

```
[vagrant@el ~]$ ls /bin | head -3
[
addr2line
alias
```

The first command in `/bin` is the square bracket!

Remark that there also exists a test written as `[[ ]]`. This is a **Bash built-in** and is more powerful than the square bracket. The double square bracket is a **keyword** and not a command. However, this notation is specific to recent versions of Bash and is not POSIX compliant. Using it makes your script less portable. We will not discuss this notation here.

24.3. more tests

Below are some example tests. Take a look at the man page of `test(1)` and `bash(1)` (under *CONDITIONAL EXPRESSIONS*) to see more options for tests.

Test	Description
`[ -d foo ]`	Does the *directory* foo exist?
`[ -e bar ]`	Does the file (or directory) bar *exist*?
`[ -f foo ]`	Is foo a *regular file*?
`[ -r bar ]`	Is bar a *readable* file?
`[ -w bar ]`	Is bar a *writeable* file?
`[ foo -nt bar ]`	Is file foo newer than file bar?
`[ '/etc' = "${PWD}" ]`	Is the string `/etc` *equal to* the variable `$PWD`?
`[ "${1}" ≠ 'secret' ]`	Is the first parameter *not equal to* `secret`?
`[ 55 -lt "${bar}" ]`	Is 55 *less than* the value of `${bar}`?
`[ "${foo}" -ge '1000' ]`	Is the value of `${foo}` greater or equal to 1000?

Numerical values must be *integers*. Floating point numbers can not be interpreted by Bash.

Tests can be combined with logical AND and OR.

```
student@linux:~$ [ 66 -gt 55 -a 66 -lt 500 ]; echo $?
0
student@linux:~$ [ 66 -gt 55 -a 660 -lt 500 ]; echo $?
1
student@linux:~$ [ 66 -gt 55 -o 660 -lt 500 ]; echo $?
0
```

However, the `-a` and `-o` options are deprecated and *not recommended*. Instead, use the `&&` and `||` operators.

```
student@linux:~$ [ 66 -gt 55 ] && [ 66 -lt 500 ]; echo $?
0
student@linux:~$ [ 66 -gt 55 ] && [ 660 -lt 500 ]; echo $?
1
student@linux:~$ [ 66 -gt 55 ] || [ 660 -lt 500 ]; echo $?
0
```

24.4. if then else

The `if then else` construction is about choice. If a certain condition is met, then execute something, else execute something else. The example below tests whether a file exists, and if the file exists then a proper message is echoed.

```
#!/bin/bash
file="isit.txt"

if [ -f "${file}" ]
then
    echo "${file} exists!"
else
    echo "${file} not found!"
fi
```

If we name the above script `choice.sh`, then it executes like this:

```
[student@linux scripts]$ ./choice.sh
isit.txt not found!
[student@linux scripts]$ touch isit.txt
[student@linux scripts]$ ./choice.sh
isit.txt exists!
[student@linux scripts]$
```

24.5. if then elif

You can nest a new `if` inside an `else` with `elif`. This is a simple example.

```
#!/bin/bash
count=42
if [ "${count}" -eq '42' ]
then
    echo "42 is correct."
elif [ "${count}" -gt '42' ]
then
    echo "Too much."
else
    echo "Not enough."
fi
```

24.6. for loop

The example below shows the syntax of a typical `for` loop in bash.

```
for i in 1 2 4
do
    echo "${i}"
done
```

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

```
#!/bin/bash
for counter in $(seq 1 20)
do
    echo "counting from 1 to 20, now at ${counter}"
    sleep 1
done
```

The same example as above can be written without the embedded shell using the Bash `{from..to}` shorthand.

```
#!/bin/bash
for counter in {1..20}
do
    echo "counting from 1 to 20, now at ${counter}"
    sleep 1
done
```

This `for loop` uses file globbing (from the shell expansion). Putting the instruction on the command line has identical functionality.

```
[student@linux ~]$ ls
count.sh  go.sh
[student@linux ~]$ for file in *.sh ; do cp "${file}" "${file.backup}" ; done
[student@linux ~]$ ls
count.sh  count.sh.backup  go.sh  go.sh.backup
```

The for loop you know from C-like programming languages like Java can also be used in Bash. However, it is much less common.

```
#!/bin/bash
for (( i=0; i<10; i++ ))
do
    echo "counting from 0 to 9, now at ${i}"
done
```

24.7. while loop

Below a simple example of a `while loop`.

```
i=100;
while [ "${i}" -ge '0' ]
do
    echo "Counting down from 100 to 0, now at $i;"
    (( i-- ))
done
```

Endless loops can be made with `while true` or `while :` , where the colon `:` is the equivalent of *no operation* in the *Korn* and *Bash* shells.

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

24.8. until loop

Below a simple example of an `until` loop.

```
i=100
until [ "${i}" -le '0' ]
do
    echo "Counting down from 100 to 1, now at ${i}"
    (( i-- ))
done
```

24.9. practice: scripting tests and loops

1. Write a script that uses a `for` loop to count from 3 to 7.
2. Write a script that uses a `for` loop to count from 1 to 17000.
3. Write a script that uses a `while` loop to count from 3 to 7.
4. Write a script that uses an `until` loop to count down from 8 to 4.
5. Write a script that uses a for loop to count the number of files ending in `.txt` in the current directory and displays a message "There are N files ending in .txt".
6. Improve the script with conditional statements so the displayed message is also correct when there are zero files or one file ending in `.txt`.

24.10. solution: scripting tests and loops

1. Write a script that uses a `for` loop to count from 3 to 7.

```
#!/bin/bash

for i in 3 4 5 6 7
do
   echo "Counting from 3 to 7, now at ${i}"
done
```

   You can also use brace expansion, e.g. `for i in {3..7}`.

2. Write a script that uses a `for` loop to count from 1 to 17000.

```
#!/bin/bash

for i in $(seq 1 17000)
do
    echo "Counting from 1 to 17000, now at ${i}"
done
```

3. Write a script that uses a `while` loop to count from 3 to 7.

```
#!/bin/bash

i=3
while [ "${i}" -le '7' ]
do
   echo "Counting from 3 to 7, now at ${i}"
   i=(( i+1 ))   # or (( i++ ))
done
```

4. Write a script that uses an `until` loop to count down from 8 to 4.

```
#!/bin/bash

i=8
until [ "${i}" -lt '4' ]
do
 echo "Counting down from 8 to 4, now at ${i}"
 (( i-- ))
done
```

5. Write a script that uses a for loop to count the number of files ending in `.txt` in the current directory and displays a message “There are N files ending in .txt”.

```bash
#!/bin/bash

i=0
for file in *.txt
do
    (( i++ ))
done
echo "There are ${i} files ending in .txt"
```

6. Improve the script with conditional statements so the displayed message is also correct when there are zero files or one file ending in `.txt`.

```bash
#! /bin/bash

if ! ls ./*.txt > /dev/null 2>&1; then
    echo "There are no files ending in .txt"
    exit 0
fi

i=0
for file in *.txt
do
    (( i++ ))
done

if [ "${i}" -eq '1' ]; then
    echo "There is 1 file ending in .txt"
else
    echo "There are ${i} files ending in .txt"
fi
```

25. scripting parameters

(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/)

25.1. script parameters

On the CLI, you often pass on options and arguments to a command to alter its behaviour. Bash *shell scripts* also can have options and arguments, called *positional parameters*. These parameters are stored in variables with names `${1}`, `${2}`, `${3}`, and so on.

```
#! /bin/bash --
echo "The first argument is ${1}"
echo "The second argument is ${2}"
echo "The third argument is ${3}"
```

If you save this script in a file called `pars.sh`, and make it executable, you can run it with parameters:

```
[student@linux scripts]$ ./pars.sh one two three
The first argument is one
The second argument is two
The third argument is three
```

Pay attention! In many code examples you encounter on the Internet, you'll see the positional parameters referenced as $1, $2, $3, etc, without the braces. This is also valid in Bash. However, if you want to reference the tenth positional parameter and you write `$10`, Bash will interpret this as the value of the first positional parameter followed by a zero. To avoid this, you should always use braces around the number, like `${10}`. For example:

```
#! /bin/bash --

# Confusing use of positional parameters, this will not expand as expected
echo "The first argument is $1"
echo "The tenth argument is $10"
```

If you run this script (let's call it `tenparams.sh`):

```
[student@linux scripts]$ ./tenparams.sh one two three four five six seven
  ↪ eight nine ten
The first argument is one
The tenth argument is one0
```

So, if you write a $ followed by a number, only the first digit will be interpreted as the positional parameter! We recommend to always using braces to avoid this confusion.

```
#! /bin/bash --

# Confusing use of positional parameters, this will not expand as expected
echo "The first argument is ${1}"
echo "The tenth argument is ${10}"
```

When you run a script, other special pre-defined variables are available. The man page of `bash(1)` has a full list, but we list a few here.

Variable	Description
`${0}`	The name of the script
`$#`	The number of parameters
`$*`	All the parameters (as one long string)
`$@`	All the parameters (as a list)
`$?`	The return code (0-255) of the last command
`$$`	The process ID of the script

An example script that uses these variables:

```
#! /bin/bash --

cat << _EOF_
The script name is: ${0}
Number of arguments: $#
The first argument is ${1}
The second argument is ${2}
The third argument is ${3}
PID of the script: $$
Last return code: $?
All the arguments (list): $@
All the arguments (string): $*
_EOF_
```

The output would look like this (if you save the script in a file called `special-vars.sh`):

```
[student@linux scripts]$ ./special-vars.sh one two three
The script name is: ./special-vars.sh
Number of arguments: 3
The first argument is one
The second argument is two
The third argument is three
PID of the script: 5612
Last return code: 0
All the arguments (list): one two three
All the arguments (string): one two three
```

If you pass on less parameters than the script expects, the missing parameters will be interpreted as empty strings. If you start the script with `set -o nounset` (or `set -u` for short), the script will exit with an error if you try to reference a positional parameter that was not passed on. When parsing parameters, always check the number of arguments first to avoid this.

25.2. shift through parameters

The `shift` command will drop the first positional parameter, and move all the others one position to the left, i.e. `${1}` will dissapear, `${2}` will become `${1}`, `${3}` will become `${2}`, and so on.

Using a `while` loop in combination with `shift` statement, you can parse all parameters one by one. This is a sample script (called `shift.sh`) that does this:

```
#! /bin/bash --

if [ "$#" -eq "0" ]
then
    echo "You have to give at least one parameter."
    exit 1
fi

while (( $# ))
do
    echo "arg: ${1}"
    shift
done
```

The `while` loop can also be written as `while [ "$#" -gt "0" ]`.

Below is some sample output of the script above.

```
[student@linux scripts]$ ./shift.sh one
arg: one
[student@linux scripts]$ ./shift.sh one two three 1201 "33 42"
arg: one
arg: two
arg: three
arg: 1201
arg: 33 42
[student@linux scripts]$ ./shift.sh
You have to give at least one parameter.
```

25.3. for loop through parameters

You can also use a `for` loop to iterate over the positional parameters. This is a sample script (called `for.sh`) that does this:

```
#! /bin/bash --

if [ "$#" -eq "0" ]
then
    echo "You have to give at least one parameter."
    exit 1
fi

for arg in "${@}"
do
    echo "arg: ${arg}"
done
```

This script behaves in the same way as the `shift.sh` script. However, after the loop, all positional parameters are still available.

25.4. runtime input

You can ask the user for input with the `read` command in a script.

```
#!/bin/bash
read -p 'Enter your name ' -r name
echo "Hello, ${name}!"
```

Use option -p to display a prompt, and -r to prevent backslashes from being interpreted as escape characters.

You can also use read to read lines from a file and then iterate on them.

```
#! /bin/bash --
while read -r line
do
    echo "line: ${line}"
done < inputfile.txt
```

Example output:

```
[student@linux scripts]$ cat inputfile.txt
one
two
three
[student@linux scripts]$ ./readlines.sh
line: one
line: two
line: three
```

25.5. sourcing a config file

The source (as seen in the shell chapters), or the shortcut . can be used to source a configuration file. With source, the specified file is executed *in the current shell*, i.e. without creating a subshell. Consequently, variables set in the configuration file are available in the script that sources the file.

Below a sample configuration file for an application.

```
[student@linux scripts]$ cat myApp.conf
# The config file of myApp

# Enter the path here
myAppPath=/var/myApp

# Enter the number of quines here
quines=5
```

And here an application (my-app.sh) that uses this file.

```
#! /bin/bash --
#
# Welcome to the myApp application
#

# Source the configuration file
. ./myApp.conf

echo "There are ${quines} quines"
```

The running application can use the values inside the sourced configuration file.

```
[student@linux scripts]$ ./my-app.sh
There are 5 quines
```

25.6. get script options with getopts

The `getopts` function allows you to parse options given to a command. The following script allows for any combination of the options a, f and z.

```
#! /bin/bash --

while getopts ":afz" option;
do
    case "${option}" in
        a)
            echo 'received -a'
            ;;
        f)
            echo 'received -f'
            ;;
        z)
            echo 'received -z'
            ;;
        *)
            echo "invalid option -${OPTARG}"
            ;;
    esac
done
```

This is sample output from the script above. First we use correct options, then we enter twice an invalid option.

```
student@linux$ ./options.ksh
student@linux$ ./options.ksh -af
received -a
received -f
student@linux$ ./options.ksh -zfg
received -z
received -f
invalid option -g
student@linux$ ./options.ksh -a -b -z
received -a
invalid option -b
received -z
```

You can also check for options that need an argument, as this example script(`argoptions.sh`) shows.

```
#! /bin/bash --

while getopts ":af:z" option
do
    case $option in
        a)
            echo 'received -a'
            ;;
        f)
            echo "received -f with ${OPTARG}"
            ;;
        z)
            echo 'received -z'
            ;;
        :)
```

```
            echo "option -${OPTARG} needs an argument"
            ;;
        *)
            echo "invalid option -${OPTARG}"
            ;;
    esac
done
```

This is sample output from the script above.

```
student@linux$ ./argoptions.sh -a -f hello -z
received -a
received -f with hello
received -z
student@linux$ ./argoptions.sh -zaf 42
received -z
received -a
received -f with 42
student@linux$ ./argoptions.sh -zf
received -z
option -f needs an argument
```

Additionally, there's a utility command called `getopt` (part of the GNU project) that can be used to parse complex cases with mixed use of long options (like `--verbose`), combinations of multiple short options (like `-abc` for `-a -b -c`), and options with arguments (like `-f file`). The `getopt` command will rewrite the command line options in a "canonical" form that is easier to parse with a `while` loop. However, `getopt` is not portable: BSD and macOS don't have GNU `getopt`, but they have their own version of `getopt` that is less powerful. Using `getopt` is out of the scope of this chapter.

25.7. get shell options with shopt

You can toggle the values of variables controlling optional shell behaviour with the `shopt` built-in shell command. The example below first verifies whether the cdspell option is set; it is not. The next shopt command sets the value, and the third shopt command verifies that the option really is set. You can now use minor spelling mistakes in the `cd` command. The man page of `bash(1)` has a complete list of options.

```
student@linux:~$ shopt -q cdspell ; echo $?
1
student@linux:~$ shopt -s cdspell
student@linux:~$ shopt -q cdspell ; echo $?
0
student@linux:~$ cd /Etc
/etc
```

25.8. practice: parameters and options

1. Write a script that receives four parameters, and outputs them in reverse order.
2. Write a script that receives two parameters (two filenames) and outputs whether those files exist.
3. Write a script that takes a filename as parameter, or asks for a filename if none was given. Verify the existence of the file, then verify that you own the file, and whether it is writable. If not, then make it writable.

4. Make a configuration file for the previous script. Put a logging switch in the config file, logging means writing detailed output of everything the script does to a log file in `/tmp`.

25.9. solution: parameters and options

1. Write a script that receives four parameters, and outputs them in reverse order.

```
#!/bin/bash
echo "${4} ${3} ${2} ${1}"
```

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

```
#!/bin/bash

if [ "$#" -ne '2' ]
then
    echo "This script needs two parameters, got $#" >&2
    echo "Usage: ${0} <file1> <file2>" >&2
    exit 1
fi

if [ -f "${1}" ]
then
    echo "${1} exists!"
else
    echo "${1} not found!"
fi

if [ -f "${2}" ]
then
    echo "${2} exists!"
else
    echo "${2} not found!"
fi
```

3. Write a script (e.g. named `chkw.sh`) that takes a filename as parameter, or asks for a filename if none was given. Verify the existence of the file, then verify that you own the file, and whether it is writable. If not, then make it writable.

```
#! /bin/bash

# Check if a filename was given as a parameter
if [ "$#" -eq '0' ]; then
    read -r -p "Enter a filename: " filename
else
    filename="${1}"
fi

# Check if the file exists
if [ ! -f "${filename}" ]; then
    echo "File does not exist, or is not a file"
    exit 1
fi

# Check if the file is owned by the user
if [ ! -O "${filename}" ]; then
    echo "You do not own this file"
    exit 1
fi

# Check if the file is writable
if [ ! -w "${filename}" ]; then
    echo "File is not writable"
    chmod u+w "${filename}"
    echo "File is now writable"
else
    echo "File is writable"
fi
```

Interaction with the script:

```
[student@linux scripts]$ touch test{1..3}.txt
[student@linux scripts]$ ls -l
total 4
-rwxr--r--. 1 student student 970 25 okt 12:39 chkw.sh
-rw-------. 1 student student   0 25 okt 12:41 test1.txt
-rw-------. 1 student student   0 25 okt 12:41 test2.txt
-rw-------. 1 student student   0 25 okt 12:41 test3.txt
[student@linux scripts]$ chmod -w test1.txt
[student@linux scripts]$ sudo chown root:root test2.txt
[student@linux scripts]$ ls -l
total 4
-rwxr--r--. 1 student student 970 25 okt 12:39 chkw.sh
-r--------. 1 student student   0 25 okt 12:41 test1.txt
-rw-------. 1 root    root      0 25 okt 12:41 test2.txt
-rw-------. 1 student student   0 25 okt 12:41 test3.txt
[student@linux scripts]$ ./chkw.sh
Enter a filename: test1.txt
File is not writable
File is now writable
[student@linux scripts]$ ./chkw.sh test2.txt
You do not own this file
[student@linux scripts]$ ./chkw.sh test3.txt
File is writable
[student@linux scripts]$ ls -l
total 4
-rwxr--r--. 1 student student 970 25 okt 12:39 chkw.sh
-rw-------. 1 student student   0 25 okt 12:41 test1.txt
-rw-------. 1 root    root      0 25 okt 12:41 test2.txt
-rw-------. 1 student student   0 25 okt 12:41 test3.txt
```

4. Make a configuration file for the previous script. Put a logging switch in the config file, logging means writing detailed output of everything the script does to a log file in /tmp.

```
#! /bin/bash

. .chkwrc

if [ "${logging}" = 'on' ]; then
    log="tee -a ${logfile}"
    date -Is >> "${logfile}"
    echo "Filename: ${1}" >> "${logfile}"
else
    log='cat'
fi

# Check if a filename was given as a parameter
```

```
if [ "$#" -eq '0' ]; then
    read -r -p "Enter a filename: " filename
else
    filename="${1}"
fi

# Check if the file exists
if [ ! -f "${filename}" ]; then
    echo "File does not exist, or is not a file" | ${log}
    exit 1
fi

# Check if the file is owned by the user
if [ ! -O "${filename}" ]; then
    echo "You do not own this file" | ${log}
    exit 1
fi

# Check if the file is writable
if [ ! -w "${filename}" ]; then
    echo "File is not writable" | ${log}
    chmod u+w "${filename}"
    echo "File is now writable" | ${log}
else
    echo "File is writable" | ${log}
fi
```

Configuration file (`.chkwrc`):

```
logging=on
logfile=/tmp/chkw.log
```

Interaction with the script:

```
[student@linux scripts] $ ls -l
total 4
-rwxr--r--. 1 student student 1133 26 okt 13:26 chkw.sh
-r--------. 1 student student    0 25 okt 12:41 test1.txt
-rw-------. 1 root    root       0 25 okt 12:41 test2.txt
-r--r-----. 1 student student    0 25 okt 12:41 test3.txt
[student@linux scripts] $ ./chkw.sh test1.txt
File is not writable
File is now writable
[student@linux scripts] $ ./chkw.sh test2.txt
You do not own this file
[student@linux scripts] $ ./chkw.sh test3.txt
File is not writable
File is now writable
[student@linux scripts] $ cat /tmp/chkw.log
2024-10-26T13:27:26+02:00
File is not writable
File is now writable
2024-10-26T13:27:50+02:00
Filename: test2.txt
You do not own this file
2024-10-26T13:27:54+02:00
Filename: test3.txt
File is not writable
File is now writable
```

26. more scripting

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

26.1. eval

`eval` reads arguments as input to the shell (the resulting commands are executed). This allows using the value of a variable as a variable.

```
student@linux:~/test42$ answer=42
student@linux:~/test42$ word=answer
student@linux:~/test42$ eval x=\$$word ; echo $x
42
```

Both in `bash` and `Korn` the arguments can be quoted.

```
kahlan@solexp11$ answer=42
kahlan@solexp11$ word=answer
kahlan@solexp11$ eval "y=\$$word" ; echo $y
42
```

Sometimes the `eval` is needed to have correct parsing of arguments. Consider this example where the `date` command receives one parameter `1 week ago`.

```
student@linux~$ date --date="1 week ago"
Thu Mar  8 21:36:25 CET 2012
```

When we set this command in a variable, then executing that variable fails unless we use `eval`.

```
student@linux~$ lastweek='date --date="1 week ago"'
student@linux~$ $lastweek
date: extra operand `ago"'
Try `date --help' for more information.
student@linux~$ eval $lastweek
Thu Mar  8 21:36:39 CET 2012
```

26.2. (( ))

The `(( ))` allows for evaluation of numerical expressions.

```
student@linux:~/test42$ (( 42 > 33 )) && echo true || echo false
true
student@linux:~/test42$ (( 42 > 1201 )) && echo true || echo false
false
student@linux:~/test42$ var42=42
student@linux:~/test42$ (( 42 == var42 )) && echo true || echo false
true
student@linux:~/test42$ (( 42 == $var42 )) && echo true || echo false
true
student@linux:~/test42$ var42=33
student@linux:~/test42$ (( 42 == var42 )) && echo true || echo false
false
```

26.3. let

The `let` built-in shell function instructs the shell to perform an evaluation of arithmetic expressions. It will return 0 unless the last arithmetic expression evaluates to 0.

```
[student@linux ~]$ let x="3 + 4" ; echo $x
7
[student@linux ~]$ let x="10 + 100/10" ; echo $x
20
[student@linux ~]$ let x="10-2+100/10" ; echo $x
18
[student@linux ~]$ let x="10*2+100/10" ; echo $x
30
```

The `shell` can also convert between different bases.

```
[student@linux ~]$ let x="0xFF" ; echo $x
255
[student@linux ~]$ let x="0xC0" ; echo $x
192
[student@linux ~]$ let x="0xA8" ; echo $x
168
[student@linux ~]$ let x="8#70" ; echo $x
56
[student@linux ~]$ let x="8#77" ; echo $x
63
[student@linux ~]$ let x="16#c0" ; echo $x
192
```

There is a difference between assigning a variable directly, or using `let` to evaluate the arithmetic expressions (even if it is just assigning a value).

```
kahlan@solexp11$ dec=15 ; oct=017 ; hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 017 0x0f
kahlan@solexp11$ let dec=15 ; let oct=017 ; let hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 15 15
```

26.4. case

You can sometimes simplify nested if statements with a case construct.

```
[student@linux ~]$ ./help
What animal did you see ? lion
You better start running fast!
[student@linux ~]$ ./help
What animal did you see ? dog
Don't worry, give it a cookie.
[student@linux ~]$ cat help
#!/bin/bash
#
# Wild Animals Helpdesk Advice
#
echo -n "What animal did you see ? "
read animal
case $animal in
        "lion" | "tiger")
                echo "You better start running fast!"
        ;;
        "cat")
                echo "Let that mouse go..."
        ;;
        "dog")
                echo "Don't worry, give it a cookie."
        ;;
        "chicken" | "goose" | "duck" )
                echo "Eggs for breakfast!"
        ;;
        "liger")
                echo "Approach and say 'Ah you big fluffy kitty...'."
        ;;
        "babelfish")
                echo "Did it fall out your ear ?"
        ;;
        *)
                echo "You discovered an unknown animal, name it!"
        ;;
esac
[student@linux ~]$
```

26.5. shell functions

Shell functions can be used to group commands in a logical way.

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

function greetings {
echo Hello World!
echo and hello to $USER to!
}
```

```
echo We will now call a function
greetings
echo The end
```

This is sample output from this script with a `function`.

```
kahlan@solexp11$ ./funcs.ksh
We will now call a function
Hello World!
and hello to kahlan to!
The end
```

A shell function can also receive parameters.

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

function plus {
let result="$1 + $2"
echo  $1 + $2 = $result
}

plus 3 10
plus 20 13
plus 20 22
```

This script produces the following output.

```
kahlan@solexp11$ ./addfunc.ksh
3 + 10 = 13
20 + 13 = 33
20 + 22 = 42
```

26.6. practice : more scripting

1. Write a script that asks for two numbers, and outputs the sum and product (as shown here).

```
Enter a number: 5
Enter another number: 2

Sum:       5 + 2 = 7
Product:   5 x 2 = 10
```

2. Improve the previous script to test that the numbers are between 1 and 100, exit with an error if necessary.

3. Improve the previous script to congratulate the user if the sum equals the product.

4. Write a script with a case insensitive case statement, using the shopt nocasematch option. The nocasematch option is reset to the value it had before the scripts started.

5. If time permits (or if you are waiting for other students to finish this practice), take a look at Linux system scripts in /etc/init.d and /etc/rc.d and try to understand them. Where does execution of a script start in /etc/init.d/samba ? There are also some hidden scripts in ~, we will discuss them later.

26.7. solution : more scripting

1. Write a script that asks for two numbers, and outputs the sum and product (as shown here).

```
Enter a number: 5
Enter another number: 2

Sum:       5 + 2 = 7
Product:   5 x 2 = 10
```

```
#!/bin/bash

echo -n "Enter a number : "
read n1

echo -n "Enter another number : "
read n2

let sum="$n1+$n2"
let pro="$n1*$n2"

echo -e "Sum\t: $n1 + $n2 = $sum"
echo -e "Product\t: $n1 * $n2 = $pro"
```

2. Improve the previous script to test that the numbers are between 1 and 100, exit with an error if necessary.

```
echo -n "Enter a number between 1 and 100 : "
read n1

if [ $n1 -lt 1 -o $n1 -gt 100 ]
then
       echo Wrong number...
       exit 1
fi
```

3. Improve the previous script to congratulate the user if the sum equals the product.

```
if [ $sum -eq $pro ]
then echo Congratulations $sum == $pro
fi
```

4. Write a script with a case insensitive case statement, using the shopt nocasematch option. The nocasematch option is reset to the value it had before the scripts started.

```
#!/bin/bash
#
# Wild Animals Case Insensitive Helpdesk Advice
#

if shopt -q nocasematch; then
  nocase=yes;
else
  nocase=no;
  shopt -s nocasematch;
fi

echo -n "What animal did you see ? "
read animal

case $animal in
        "lion" | "tiger")
                echo "You better start running fast!"
        ;;
        "cat")
                echo "Let that mouse go... "
        ;;
        "dog")
                echo "Don't worry, give it a cookie."
        ;;
        "chicken" | "goose" | "duck" )
                echo "Eggs for breakfast!"
        ;;
        "liger")
                echo "Approach and say 'Ah you big fluffy kitty.'"
        ;;
        "babelfish")
                echo "Did it fall out your ear ?"
        ;;
        *)
                echo "You discovered an unknown animal, name it!"
        ;;
esac

if [ nocase = yes ] ; then
        shopt -s nocasematch;
else
        shopt -u nocasematch;
fi
```

5. If time permits (or if you are waiting for other students to finish this practice), take a look at Linux system scripts in /etc/init.d and /etc/rc.d and try to understand them. Where does execution of a script start in /etc/init.d/samba ? There are also some hidden scripts in ~, we will discuss them later.

Part VIII.

Local user management

27. introduction to users

(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 little chapter will teach you how to identify your user account on a Linux 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.

Finally, you will learn how to run a command as another user with `sudo`.

27.1. whoami

The `whoami` command tells you your username.

```
student@debian:~$ whoami
student
```

27.2. who

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

```
student@debian:~$ who
yanina   tty1         2024-10-15 18:59
student  pts/0        2024-10-15 18:55 (192.168.56.1)
vagrant  pts/1        2024-10-15 18:56 (10.0.2.2)
serena   pts/2        2024-10-15 18:57 (192.168.56.11)
venus    pts/3        2024-10-15 18:57 (192.168.56.15)
```

In this example, the `who` command shows that the user `yanina` is logged in on the physical machine (no IP address is shown), the other students are logged in via SSH. The IP addresses are shown in the output.

In the second column, `tty` is short for *teletype*. This term refers to a teleprinter, a device that was in the past used to interact with a computer. The `pts` stands for *pseudo terminal slave* and refers to a virtual terminal that is used to interact with the system over a network connection.

27.3. who am i

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

```
student@debian:~$ who am i
student  pts/0        2024-10-15 18:55 (192.168.56.1)
```

In fact, any two words would work here with the same result. The following is also common:

```
student@debian:~$ who mom loves
student  pts/0        2024-10-15 18:55 (192.168.56.1)
```

27.4. w

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

```
student@debian:~$ w
 19:13:30 up 18 min,  5 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
yanina   tty1     -                18:59   14:34   0.03s  0.01s -bash
student  pts/0    192.168.56.1     18:55    1.00s  0.06s  0.01s w
vagrant  pts/1    10.0.2.2         18:56    7.00s  0.02s   ?    pager
serena   pts/2    192.168.56.1     18:57   16:29   0.02s  0.02s -zsh
venus    pts/3    192.168.56.1     18:57   42.00s  0.02s   ?    nano README.md
```

27.5. id

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

```
student@debian:~$ id
uid=1001(student) gid=1001(student) groups=1001(student),27(sudo),100(users)
```

On Enterprise Linux you will also get `SELinux` context information with this command.

```
[student@el ~]$ id
uid=1001(student) gid=1001(student) groups=1001(student),10(wheel)
 ↪  context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

27.6. su to another user

The `su` command (*substitute user*) allows a user to run a shell as another user. You need to know the password of the user you want to become.

```
student@debian:~$ su venus
Password:
venus@debian:/home/student$ pwd
/home/student
```

In this example, the user `student` becomes the user `venus`. The prompt changes to reflect the new user. The `pwd` command shows that the current directory is still the home directory of the original user. In fact, the only thing that changed is the current user id, the rest of the environment is still the same.

Used like this, the `su` command is actually not very useful. Use `su -` instead (see below).

27.7. su - $username

To become another user and also get the target user's environment, as you would when you log in as that user, issue the `su -` command followed by the target username.

```
venus@debian:/home/student$ su - venus
Password:
venus@debian:~$ pwd
/home/venus
```

27.8. su -

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

```
student@debian:~$ su -
Password:
root@debian:~# pwd
/root
```

Remark that this assumes that the root user has a password set. On modern Linux distributions, more often than not, the root user does not have a password and you will not be able to use su to become root. Instead, use sudo (see below).

27.9. 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). Nowadays, this is the preferred way to run commands with superuser privileges instead of logging in as the root user.

The screenshot below shows the usage of sudo. User student received the right to run useradd with the credentials of root. This allows student to create new users on the system without becoming root and without knowing the *root password*.

First the command fails:

```
student@debian:~$ /sbin/useradd -m -s /bin/bash valentina
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
```

But with sudo it works. The first time a user executes sudo, they have to enter their own password to confirm the action. The sudo command will remember the password for a short time (usually 15 minutes) so that they don't have to enter the password for every command.

```
student@debian:~$ sudo useradd -m -s /bin/bash valentina
student@debian:~$ getent passwd valentina
valentina:x:1006:1006::/home/valentina:/bin/bash
```

For more information about the commands used in this example, see the chapter about user management.

27.10. visudo

The /etc/sudoers file can be edited with the visudo command. This command will check the syntax of the file before saving it.

Check the man page of visudo(8) before playing with the /etc/sudoers file. Editing the sudoers is out of scope for this fundamentals book.

The default configuration of the sudoers file on *Enterprise Linux* is to give all users in the wheel group the right to use sudo. On *Debian*-based systems, users in the group sudo get these rights. In both cases, users have to enter their own password on first use. VMs created with Vagrant are set up in such a way that the default user vagrant can use sudo without entering a password.

27.11. sudo su -

On most modern Linux systems, the root user does not have a password set. This means that it is not possible to login as root from the login screen, or via SSH. In order to get a root prompt, one of the users with sudo rights can type sudo su - and become root without having to enter the root password.

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

27.12. sudo logging

Using sudo without authorization will result in a severe warning:

```
paul@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.
paul@linux:~$
```

On systemd based distributions, use journalctl (with superuser privileges) to see the logs:

```
student@debian:~$ sudo journalctl -et sudo
[ ... some output omitted ... ]
Oct 15 19:21:30 debian sudo[1669]:    paul : user NOT in sudoers ; TTY=pts/0
 ↪ ; PWD=/home/paul ; USER=root ; COMMAND=/usr/bin/su -
```

On *Enterprise Linux*, there is a separate log file for sudo, /var/log/secure:

```
[vagrant@el ~]$ sudo grep 'NOT in sudoers' /var/log/secure
Oct 16 07:00:19 el sudo[7458]:  paul : user NOT in sudoers ; TTY=pts/1 ;
 ↪ PWD=/home/paul ; USER=root ; COMMAND=/bin/su -
```

On older *Debian* systems, the log file is /var/log/auth.log, with similar content. If this file does not exist, this is an indication that you're on a newer system and that journalct should be used.

27.13. 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.

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 chapter about user management.)

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

27.14. 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. ry to create a new user account (when using your normal user account). This should fail. (Details on adding user accounts are explained in the chapter about user management.)

```
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.
laura@linux:~$
```

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.

28. user management

(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 will teach you how to create, modify and remove user accounts.

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

28.1. 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`, `passwd`, `gpasswd` 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 a text editor like `vi` or `nano`. This is strongly discouraged, though, since a small mistake in the configuration file format may make your system unusable.

28.2. /etc/passwd

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

```
student@linux:~$ tail /etc/passwd
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
messagebus:x:100:107::/nonexistent:/usr/sbin/nologin
sshd:x:101:65534::/run/sshd:/usr/sbin/nologin
_rpc:x:102:65534::/run/rpcbind:/usr/sbin/nologin
statd:x:103:65534::/var/lib/nfs:/usr/sbin/nologin
vagrant:x:1000:1000:vagrant,,,:/home/vagrant:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
student:x:1001:1001::/home/student:/bin/bash
tcpdump:x:104:109::/nonexistent:/usr/sbin/nologin
polkitd:x:994:994:polkit:/nonexistent:/usr/sbin/nologin
```

This file contains seven columns separated by a colon. The columns contain the username, an x, the user id (a number that uniquely identifies a user), the primary group id, a description, the name of the user's home directory, and the login shell.

More information can be found in man page `passwd(5)`:

```
student@linux:~$ man 5 passwd
```

Searching for information about a user in the passwd database can be done with the `getent` command. The example below shows the information for the user `student`.

```
student@linux:~$ getent passwd student
student:x:1001:1001::/home/student:/bin/bash
```

Since the `passwd` database is a plain text file, `grep` works as well.

```
student@linux:~$ grep student /etc/passwd
student:x:1001:1001::/home/student:/bin/bash
```

It may be counterintuitive, but the `passwd` file does *not* contain the field it's named after, viz. the user's password. Originally, it was kept in the second field. Since the `passwd` file is world readable, realisation dawned that this was a bad idea, even if the password is stored in an encrypted (hashed) format. The password is now stored in the `shadow` file.

28.3. /etc/shadow

The `shadow` file contains additional information about users, specifically the encrypted password and password-related settings. The file is only readable by the root user.

```
student@debian:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 944 Oct 15 14:38 /etc/shadow
```

On Enterprise Linux, even the owner's permissions are turned off!

```
[student@el ~]$ ls -l /etc/shadow
----------. 1 root root 1234 Oct 15 10:16 /etc/shadow
```

However, root ignores file permissions and still can read (and edit) the file.

Finding information about a user from the shadow file can also be done with `getent`, however, you need superuser privileges.

```
student@debian:~$ getent shadow student
student@debian:~$ sudo !!
sudo getent shadow student
student:$y$j9T$k/zvYGDp1caN0p50aa9QI.$svec3ZhaLcxykbHKh6SMb4VnhYiJKgdN2ZhDGBLApa6:20007:0:99999:7:::
```

Without the `sudo` password, the command does not return any output.

The second field is the hashed password, with the other fields you can configure password expiration and more. See the `shadow(5)` man page for details.

28.4. 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 user id 0 (regardless of the name of the account).

```
student@linux:~$ getent passwd root
root:x:0:0:root:/root:/bin/bash
```

28.5. 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 login shell (`-s`), and setting the comment field that usually contains the user's real name (`-c`).

After that, we test whether the user was created correctly.

```
student@debian:~$ sudo useradd -m -s /bin/bash -c "Yanina Wickmayer" yanina
student@debian:~$ getent passwd yanina
yanina:x:1002:1002:Yanina Wickmayer:/home/yanina:/bin/bash
```

The user named yanina received userid 1002 and `primary group` id 1002 (a newly created group with name `yanina`).

Remark that on **Debian**-based systems, the `useradd` command does **not** automatically create a home directory and sets the login shell to `/bin/sh`. Consequently, the `-m` and `-s` options are necessary for creating a user that can log in. On **Enterprise Linux**, `useradd` does create a home directory and `/bin/bash` is the default login shell, so `useradd <user>` is sufficient.

At this time, the user is not yet able to log in.

```
student@debian:~$ getent shadow yanina
yanina:!:20011:0:99999:7:::
```

The password field contains a `!`, which means that the account is locked. The password must be set with the `passwd` command.

```
student@debian:~$ sudo passwd yanina
New password:
Retype new password:
passwd: password updated successfully
student@debian:~$ sudo getent shadow yanina
yanina:$y$j9T$mUV.AmwvCHj8RlVknMJxi0$zkYFDtn4oHWhGqd8kTlw5sWr8/xnykHwGBVIzBGLRg6:20011:0:99999:7:::
```

The password field now contains a hashed password, so we can test whether the user can log in.

```
student@debian:~$ su - yanina
Password:
yanina@debian:~$ pwd
/home/yanina
```

28.5.1. /etc/default/useradd

Both *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`.

```
student@debian:~$ /sbin/useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
LOG_INIT=yes
```

28.6. usermod

You can modify the properties of a user like the comment field, login shell, password expiration, etc. with the `usermod` command.

The `usermod` command can also be used to change group membership of users. This is discussed in the chapter about groups.

28.6.1. changing the comment field

This example uses `usermod` to change the description of a new user tux.

```
student@debian:~$ sudo useradd -m -s /bin/bash tux
student@debian:~$ getent passwd tux
tux:x:1003:1003::/home/tux:/bin/bash
student@debian:~$ sudo usermod -c "Tuxedo T. Penguin" tux
student@debian:~$ getent passwd tux
tux:x:1003:1003:Tuxedo T. Penguin:/home/tux:/bin/bash
```

28.6.2. locking an account

The command can also be used with the `-L` option to temporarily lock the account of a user.

```
student@debian:~$ sudo passwd tux
New password:
Retype new password:
passwd: password updated successfully
student@debian:~$ su - tux
Password:
tux@debian:~$
logout
student@debian:~$ sudo usermod -L tux
student@debian:~$ su - tux
Password:
su: Authentication failure
```

To re-enable the account, use the `-U` option.

```
student@debian:~$ sudo usermod -U tux
student@debian:~$ su - tux
Password:
tux@debian:~$
logout
```

28.6.3. 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.

```
student@debian:~$ getent passwd annelies
annelies:x:1006:1006:sword fighter:/home/annelies:/bin/bash
student@debian:~$ getent passwd laura
laura:x:1007:1007:art dealer:/home/laura:/bin/ksh
```

You can use the usermod command to change the shell for a user.

```
student@debian:~$ sudo usermod -s /bin/ksh annelies
student@debian:~$ getent passwd annelies
annelies:x:1006:1006:sword fighter:/home/annelies:/bin/ksh
```

28.7. chsh

Users can change their own login shell with the `chsh` command.

In the example below, user `yanina` obtains a list of available shells and then changes their login shell to the Z shell (`/bin/zsh`). First install `zsh` before trying this yourself.

```
yanina@debian:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/usr/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
/bin/zsh
/usr/bin/zsh
/usr/bin/zsh
yanina@debian:~$ getent passwd yanina
yanina:x:1002:1002:Yanina Wickmayer:/home/yanina:/bin/bash
yanina@debian:~$ chsh -s /usr/bin/zsh
Password:
yanina@debian:~$ exit
logout
student@debian:~$ su - yanina
Password:
yanina@debian ~ % echo $SHELL
/usr/bin/zsh
yanina@debian ~ % getent passwd yanina
yanina:x:1002:1002:Yanina Wickmayer:/home/yanina:/usr/bin/zsh
```

On Enterprise Linux, `chsh` has an option `-l` option that lists the available shells. This option is not available on Debian.

```
[student@el ~]$ chsh -l
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
/usr/bin/zsh
/bin/zsh
```

28.8. userdel

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

```
student@debian:~$ sudo userdel -r yanina
```

28.9. managing home directories

The easiest way to create a home directory is to supply the `-m` option with `useradd`. If you forgot the option on a Debian system, you could delete the user and start again, or create the directory yourself. This also requires setting the owner and the permissions on the directory with `chmod` and `chown` (both commands are discussed in detail in another chapter).

```
student@debian:~$ sudo useradd -s /bin/bash laura
student@debian:~$ getent passwd laura
laura:x:1004:1004::/home/laura:/bin/bash
student@debian:~$ sudo mkdir /home/laura
student@debian:~$ sudo chown laura:laura /home/laura
student@debian:~$ sudo chmod -R 700 /home/laura
student@debian:~$ ls -ld /home/laura/
drwx------ 2 laura laura 4096 Jun 24 15:17 /home/laura/
```

28.9.1. /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.

```
student@debian:~$ ls -la /etc/skel
total 20
drwxr-xr-x  2 root root 4096 Jan 31  2024 .
drwxr-xr-x 81 root root 4096 Oct 15 14:07 ..
-rw-r--r--  1 root root  220 Apr 23  2023 .bash_logout
-rw-r--r--  1 root root 3526 Apr 23  2023 .bashrc
-rw-r--r--  1 root root  807 Apr 23  2023 .profile
```

28.9.2. deleting home directories

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

```
student@debian:~$ sudo useradd -m -s /bin/bash wim
student@debian:~$ getent passwd wim
wim:x:1005:1005::/home/wim:/bin/bash
student@debian:~$ sudo userdel -r wim
student@debian:~$ ls -ld /home/wim/
ls: cannot access '/home/wim/': No such file or directory
```

28.10. adduser, deluser (debian/ubuntu)

On Debian/Ubuntu, an additional command to manage users is available: `adduser` and `deluser`. The `adduser` command is a more user-friendly frontend to `useradd` and does all that is necessary to create a new user that can immediately log in to the system. The disadvantage is that it's an interactive command, so it's not suited to use in a script to automate the installation of a machine.

```
student@debian:~$ sudo adduser kim
Adding user `kim' ...
Adding new group `kim' (1003) ...
Adding new user `kim' (1003) with group `kim (1003)' ...
Creating home directory `/home/kim' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for kim
Enter the new value, or press ENTER for the default
        Full Name []: Kim Clijsters
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
Adding new user `kim' to supplemental / extra groups `users' ...
Adding user `kim' to group `users' ...
student@debian:~$ su - kim
Password:
kim@debian:~$ pwd
/home/kim
```

The `deluser` command is a frontend to `userdel` and `groupdel` and removes a user and the associated group. The home directory is not removed by default.

```
student@debian:~$ sudo deluser kim
Removing crontab ...
Removing user `kim' ...
Done.
student@debian:~$ ls /home/
kim  student  yanina
student@debian:~$ sudo rm -rf /home/kim/
```

28.11. 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 second user named `venus`, including home directory, Bash as login 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 the `date` command as their default logon shell, `/tmp` as their home directory and an empty string as password.
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/zsh`. Verify before and after you make this change.

28.12. 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.

```
student@debian:~$ sudo useradd -m -c 'Serena Williams' serena
```

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

```
student@debian:~$ sudo useradd -m -s /bin/bash -c 'Venus Williams' venus
```

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

```
student@debian:~$ getent passwd serena
serena:x:1002:1002:Serena Williams:/home/serena:/bin/sh
student@debian:~$ sudo getent shadow serena
serena:!:20011:0:99999:7:::
student@debian:~$ getent passwd venus
venus:x:1003:1003:Venus Williams:/home/venus:/bin/bash
student@debian:~$ sudo getent shadow venus
venus:!:20011:0:99999:7:::
```

At this time, their password isn't set yet, so the shadow file will show `!` as the password hash, which also denotes that the account is locked.

```
student@debian:~$ sudo passwd serena
New password:
Retype new password:
passwd: password updated successfully
student@debian:~$ sudo getent shadow serena
serena:$y$j9T$7VErSS/8GYyeALTc7nC0Y.$PeNsJlxzG3tfZ9yEk1rKgDRc4KJVZvHiWj
    wfdIeKSi0:20011:0:99999:7:::
```

Setting the password for `venus` is equivalent.

4. Verify that their home directory was created.

```
student@debian:~$ ls -l /home
total 16
drwxr-xr-x 2 serena  serena  4096 Oct 15 14:38 serena
drwxr-xr-x 2 student student 4096 Oct 15 15:04 student
drwxr-xr-x 2 venus   venus   4096 Oct 15 14:38 venus
```

5. Create a user named `einstime` with the `date` command as their default logon shell, `/tmp` as their home directory and an empty string as password.

```
student@debian:~$ sudo useradd -s $(which date) -d /tmp -p '' einstime
student@debian:~$ getent passwd einstime
einstime:x:1004:1004::/tmp:/bin/date
student@debian:~$ sudo getent shadow einstime
einstime::20011:0:99999:7:::
```

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?

```
student@debian:~$ su - einstime
Tue Oct 15 04:33:59 PM UTC 2024
```

You get to see the current time. This trick can also be useful when you want to restrict a user to a specific application only. 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.

```
student@debian:~$ sudo nano /etc/skel/welcome.txt
student@debian:~$ cat /etc/skel/welcome.txt
Welcome to Debian 12! Have fun while learning!
```

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

```
student@debian:~$ sudo useradd -m -s /bin/bash testuser
student@debian:~$ sudo su - testuser
testuser@debian:~$ ls -l
total 4
-rw-r--r-- 1 testuser testuser 47 Oct 15 16:36 welcome.txt
testuser@debian:~$ cat welcome.txt
Welcome to Debian 12! Have fun while learning!
testuser@debian:~$ ^D
logout
student@debian:~$ sudo userdel -r testuser
```

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

```
student@debian:~$ getent passwd serena
serena:x:1002:1002:Serena Williams:/home/serena:/bin/sh
student@debian:~$ sudo usermod -s /bin/zsh serena
student@debian:~$ getent passwd serena
serena:x:1002:1002:Serena Williams:/home/serena:/bin/zsh
student@debian:~$ su - serena
Password:
serena@debian ~ % echo $SHELL
/bin/zsh
```

29. 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.

29.1. 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
```

29.2. 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`.

29.3. 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 ~]#
```

29.4. 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!

29.5. 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.
```

29.6. /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:~#
```

29.7. 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:~#
```

29.8. 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.

29.9. 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)
```

29.10. 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.

29.11. 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
```

30. User profiles

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

Logged on users have a number of preset (and customized) aliases, variables, and functions, but where do they come from ? The `shell` uses a number of startup files that are executed (or rather `sourced`) whenever the shell is invoked. What follows is an overview of startup scripts.

30.1. system profile

Both the `bash` and the `ksh` shell will verify the existence of `/etc/profile` and `source` it if it exists.

When reading this script, you will notice (both on Debian and on Red Hat Enterprise Linux) that it builds the PATH environment variable (among others). The script might also change the PS1 variable, set the HOSTNAME and execute even more scripts like `/etc/inputrc`

This screenshot uses grep to show PATH manipulation in `/etc/profile` on Debian.

```
root@linux:~# grep PATH /etc/profile
  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
export PATH
root@linux:~#
```

This screenshot uses grep to show PATH manipulation in `/etc/profile` on RHEL7/CentOS7.

```
[root@linux ~]# grep PATH /etc/profile
    case ":${PATH}:" in
                PATH=$PATH:$1
                PATH=$1:$PATH
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL
[root@linux ~]#
```

The `root user` can use this script to set aliases, functions, and variables for every user on the system.

30.2. ~/.bash_profile

When this file exists in the home directory, then `bash` will source it. On Debian Linux 5/6/7 this file does not exist by default.

RHEL7/CentOS7 uses a small `~/.bash_profile` where it checks for the existence of `~/.bashrc` and then sources it. It also adds $HOME/bin to the $PATH variable.

```
[root@linux ~]# cat /home/paul/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
[root@linux ~]#
```

30.3. ~/.bash_login

When `.bash_profile` does not exist, then `bash` will check for `~/.bash_login` and source it.

Neither Debian nor Red Hat have this file by default.

30.4. ~/.profile

When neither `~/.bash_profile` and `~/.bash_login` exist, then bash will verify the existence of `~/.profile` and execute it. This file does not exist by default on Red Hat.

On Debian this script can execute `~/.bashrc` and will add $HOME/bin to the $PATH variable.

```
root@linux:~# tail -11 /home/paul/.profile
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

RHEL/CentOS does not have this file by default.

30.5. ~/.bashrc

The `~/.bashrc` script is often sourced by other scripts. Let us take a look at what it does by default.

Red Hat uses a very simple `~/.bashrc`, checking for `/etc/bashrc` and sourcing it. It also leaves room for custom aliases and functions.

```
[root@linux ~]# cat /home/paul/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-
paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

On Debian this script is quite a bit longer and configures $PS1, some history variables and a number af active and inactive aliases.

```
root@linux:~# wc -l /home/paul/.bashrc
110 /home/paul/.bashrc
```

30.6. ~/.bash_logout

When exiting `bash`, it can execute `~/.bash_logout`.

Debian use this opportunity to clear the console screen.

```
serena@linux:~$ cat .bash_logout
# ~/.bash_logout: executed by bash(1) when login shell exits.

# when leaving the console clear the screen to increase privacy

if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
```

Red Hat Enterprise Linux 5 will simple call the `/usr/bin/clear` command in this script.

```
[serena@linux ~]$ cat .bash_logout
# ~/.bash_logout

/usr/bin/clear
```

Red Hat Enterprise Linux 6 and 7 create this file, but leave it empty (except for a comment).

```
student@linux:~$ cat .bash_logout
# ~/.bash_logout
```

30.7. Debian overview

Below is a table overview of when Debian is running any of these bash startup scripts.

Table 30.1.: Debian User Environment

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

30.8. RHEL5 overview

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

Table 30.2.: Red Hat User Environment

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

30.9. practice: user profiles

1. Make a list of all the profile files on your system.

2. Read the contents of each of these, often they `source` extra scripts.

3. Put a unique variable, alias and function in each of those files.

4. Try several different ways to obtain a shell (su, su -, ssh, tmux, gnome-terminal, Ctrl-alt-F1, ...) and verify which of your custom variables, aliases and function are present in your environment.

5. Do you also know the order in which they are executed?

6. When an application depends on a setting in $HOME/.profile, does it matter whether $HOME/.bash_profile exists or not ?

30.10. solution: user profiles

1. Make a list of all the profile files on your system.

```
ls -a ~ ; ls -l /etc/pro* /etc/bash*
```

2. Read the contents of each of these, often they `source` extra scripts.

3. Put a unique variable, alias and function in each of those files.

4. Try several different ways to obtain a shell (su, su -, ssh, tmux, gnome-terminal, Ctrl-alt-F1, ...) and verify which of your custom variables, aliases and function are present in your environment.

5. Do you also know the order in which they are executed?

```
same name aliases, functions and variables will overwrite each other
```

6. When an application depends on a setting in $HOME/.profile, does it matter whether $HOME/.bash_profile exists or not ?

```
Yes it does matter. (man bash /INVOCATION)
```

31. 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`.

31.1. 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
```

31.2. 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.

31.3. 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 ~]$
```

31.4. 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.

31.5. 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
```

31.6. groupdel

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

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

31.7. 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
```

31.8. 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#
```

31.9. 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.

31.10. 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.

31.11. 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 IX.

File security

32. 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*.

32.1. file ownership

32.1.1. 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.

32.1.2. 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#
```

32.1.3. 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
```

32.2. 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
```

32.3. permissions

32.3.1. 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`)

32.3.2. 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.

32.3.3. 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.

32.3.4. 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!)

32.3.5. 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
```

32.3.6. 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
```

32.3.7. 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/
```

32.3.8. 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
```

32.4. 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.

32.5. 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 permitted
```

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'? y
[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/
```

33. advanced file permissions

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

33.1. sticky bit on directory

You can set the `sticky bit` on a directory to prevent users from removing files that they do not own as a user owner. The sticky bit is displayed at the same location as the x permission for others. The sticky bit is represented by a `t` (meaning x is also there) or a `T` (when there is no x for others).

```
root@linux:~# mkdir /project55
root@linux:~# ls -ld /project55
drwxr-xr-x  2 root root 4096 Feb  7 17:38 /project55
root@linux:~# chmod +t /project55/
root@linux:~# ls -ld /project55
drwxr-xr-t  2 root root 4096 Feb  7 17:38 /project55
root@linux:~#
```

The `sticky bit` can also be set with octal permissions, it is binary 1 in the first of four triplets.

```
root@linux:~# chmod 1775 /project55/
root@linux:~# ls -ld /project55
drwxrwxr-t  2 root root 4096 Feb  7 17:38 /project55
root@linux:~#
```

You will typically find the `sticky bit` on the `/tmp` directory.

```
root@linux:~# ls -ld /tmp
drwxrwxrwt 6 root root 4096 2009-06-04 19:02 /tmp
```

33.2. setgid bit on directory

`setgid` can be used on directories to make sure that all files inside the directory are owned by the group owner of the directory. The `setgid` bit is displayed at the same location as the x permission for group owner. The `setgid` bit is represented by an `s` (meaning x is also there) or a `S` (when there is no x for the group owner). As this example shows, even though `root` does not belong to the group proj55, the files created by root in /project55 will belong to proj55 since the `setgid` is set.

```
root@linux:~# groupadd proj55
root@linux:~# chown root:proj55 /project55/
root@linux:~# chmod 2775 /project55/
root@linux:~# touch /project55/fromroot.txt
root@linux:~# ls -ld /project55/
drwxrwsr-x  2 root proj55 4096 Feb  7 17:45 /project55/
root@linux:~# ls -l /project55/
total 4
-rw-r--r--  1 root proj55 0 Feb  7 17:45 fromroot.txt
root@linux:~#
```

You can use the `find` command to find all `setgid` directories.

```
student@linux:~$ find / -type d -perm -2000 2> /dev/null
/var/log/mysql
/var/log/news
/var/local
...
```

33.3. setgid and setuid on regular files

These two permissions cause an executable file to be executed with the permissions of the `file owner` instead of the `executing owner`. This means that if any user executes a program that belongs to the `root user`, and the `setuid` bit is set on that program, then the program runs as `root`. This can be dangerous, but sometimes this is good for security.

Take the example of passwords; they are stored in `/etc/shadow` which is only readable by `root`. (The `root` user never needs permissions anyway.)

```
root@linux:~# ls -l /etc/shadow
-r--------  1 root root 1260 Jan 21 07:49 /etc/shadow
```

Changing your password requires an update of this file, so how can normal non-root users do this? Let's take a look at the permissions on the `/usr/bin/passwd`.

```
root@linux:~# ls -l /usr/bin/passwd
-r-s--x--x  1 root root 21200 Jun 17  2005 /usr/bin/passwd
```

When running the `passwd` program, you are executing it with `root` credentials.

You can use the `find` command to find all `setuid` programs.

```
student@linux:~$ find /usr/bin -type f -perm -04000
/usr/bin/arping
/usr/bin/kgrantpty
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/sudo
/usr/bin/fping6
/usr/bin/passwd
/usr/bin/gpasswd
...
```

In most cases, setting the `setuid` bit on executables is sufficient. Setting the `setgid` bit will result in these programs to run with the credentials of their group owner.

33.4. setuid on sudo

The `sudo` binary has the `setuid` bit set, so any user can run it with the effective userid of root.

```
student@linux:~$ ls -l $(which sudo)
---s--x--x. 1 root root 123832 Oct  7  2013 /usr/bin/sudo
student@linux:~$
```

33.5. practice: sticky, setuid and setgid bits

1a. Set up a directory, owned by the group sports.

1b. Members of the sports group should be able to create files in this directory.

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

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

1e. Test that this works!

2. Verify the permissions on `/usr/bin/passwd`. Remove the `setuid`, then try changing your password as a normal user. Reset the permissions back and try again.

3. If time permits (or if you are waiting for other students to finish this practice), read about file attributes in the man page of `chattr` and `lsattr`. Try setting the `i` attribute on a file and test that it works.

33.6. solution: sticky, setuid and setgid bits

1a. Set up a directory, owned by the group sports.

```
groupadd sports

mkdir /home/sports

chown root:sports /home/sports
```

1b. Members of the sports group should be able to create files in this directory.

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

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

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

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

```
chmod +t /home/sports
```

1e. Test that this works!

Log in with different users (group members and others and root), create files and watch the permissions. Try changing and deleting files...

2. Verify the permissions on `/usr/bin/passwd`. Remove the `setuid`, then try changing your password as a normal user. Reset the permissions back and try again.

```
root@linux:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
root@linux:~# chmod 755 /usr/bin/passwd
root@linux:~# ls -l /usr/bin/passwd
-rwxr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
```

A normal user cannot change password now.

```
root@linux:~# chmod 4755 /usr/bin/passwd
root@linux:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
```

3. If time permits (or if you are waiting for other students to finish this practice), read about file attributes in the man page of `chattr` and `lsattr`. Try setting the `i` attribute on a file and test that it works.

```
student@linux:~$ sudo su -
[sudo] password for paul:
root@linux:~# mkdir attr
root@linux:~# cd attr/
root@linux:~/attr# touch file42
root@linux:~/attr# lsattr
------------------ ./file42
root@linux:~/attr# chattr +i file42
root@linux:~/attr# lsattr
----i------------- ./file42
root@linux:~/attr# rm -rf file42
rm: cannot remove `file42': Operation not permitted
root@linux:~/attr# chattr -i file42
root@linux:~/attr# rm -rf file42
root@linux:~/attr#
```

34. access control lists

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

Standard Unix permissions might not be enough for some organisations. This chapter introduces `access control lists` or `acl's` to further protect files and directories.

34.1. acl in /etc/fstab

File systems that support `access control lists`, or `acls`, have to be mounted with the `acl` option listed in `/etc/fstab`. In the example below, you can see that the root file system has `acl` support, whereas /home/data does not.

```
root@linux:~# tail -4 /etc/fstab
/dev/sda1        /              ext3     acl,relatime     0  1
/dev/sdb2        /home/data     auto     noacl,defaults   0  0
pasha:/home/r    /home/pasha    nfs      defaults         0  0
wolf:/srv/data   /home/wolf     nfs      defaults         0  0
```

34.2. getfacl

Reading `acls` can be done with `/usr/bin/getfacl`. This screenshot shows how to read the `acl` of `file33` with `getfacl`.

```
student@linux:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
mask::rwx
other::r--
```

34.3. setfacl

Writing or changing `acls` can be done with `/usr/bin/setfacl`. These screenshots show how to change the `acl` of `file33` with `setfacl`.

First we add user `sandra` with octal permission 7 to the `acl`.

```
student@linux:~/test$ setfacl -m u:sandra:7 file33
```

Then we add the group tennis with octal permission 6 to the `acl` of the same file.

```
student@linux:~/test$ setfacl -m g:tennis:6 file33
```

The result is visible with `getfacl`.

```
student@linux:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
user:sandra:rwx
group::r--
group:tennis:rw-
mask::rwx
other::r--
```

34.4. remove an acl entry

The `-x` option of the `setfacl` command will remove an `acl` entry from the targeted file.

```
student@linux:~/test$ setfacl -m u:sandra:7 file33
student@linux:~/test$ getfacl file33 | grep sandra
user:sandra:rwx
student@linux:~/test$ setfacl -x sandra file33
student@linux:~/test$ getfacl file33 | grep sandra
```

Note that omitting the `u` or `g` when defining the `acl` for an account will default it to a user account.

34.5. remove the complete acl

The `-b` option of the `setfacl` command will remove the `acl` from the targeted file.

```
student@linux:~/test$ setfacl -b file33
student@linux:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
other::r--
```

34.6. the acl mask

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

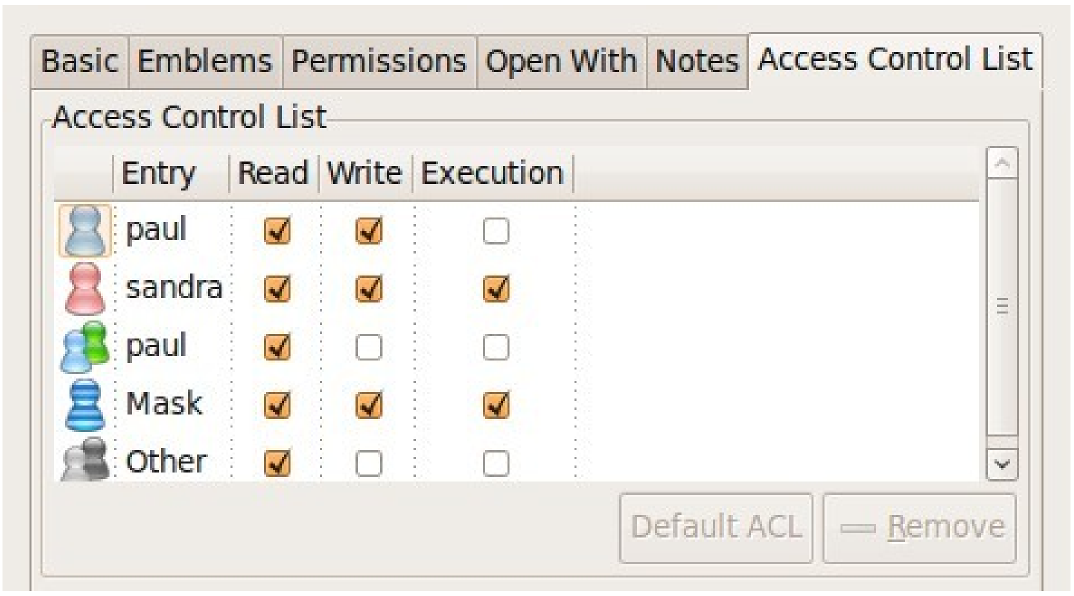
You can prevent the calculation by using the `--no-mask` switch.

```
student@linux:~/test$ setfacl --no-mask -m u:sandra:7 file33
student@linux:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
```

```
user::rw-
user:sandra:rwx          #effective:rw-
group::r--
mask::rw-
other::r--
```

34.7. eiciel

Desktop users might want to use `eiciel` to manage `acls` with a graphical tool.



	Entry	Read	Write	Execution
	paul	✓	✓	☐
	sandra	✓	✓	✓
	paul	✓	☐	☐
	Mask	✓	✓	✓
	Other	✓	☐	☐

You will need to install `eiciel` and `nautilus-actions` to have an extra tab in `nautilus` to manage `acls`.

```
student@linux:~$ sudo aptitude install eiciel nautilus-actions
```

35. 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.

35.1. inodes

35.1.1. 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/
```

35.1.2. 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.

35.1.3. 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).

35.1.4. 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.

35.2. about directories

35.2.1. 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$
```

35.2.2. . 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.

35.3. hard links

35.3.1. 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.

35.3.2. 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
```

35.4. 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).

35.5. 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
```

35.6. 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).

35.7. 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
```

A. certifications

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

A.1. Certification

A.1.1. LPI: Linux Professional Institute

A.1.1.1. LPIC Level 1

This is the junior level certification. You need to pass exams 101 and 102 to achieve `LPIC 1 certification`. To pass level one, you will need Linux command line, user management, backup and restore, installation, networking, and basic system administration skills.

A.1.1.2. LPIC Level 2

This is the advanced level certification. You need to be LPIC 1 certified and pass exams 201 and 202 to achieve `LPIC 2 certification`. To pass level two, you will need to be able to administer medium sized Linux networks, including Samba, mail, news, proxy, firewall, web, and ftp servers.

A.1.1.3. LPIC Level 3

This is the senior level certification. It contains one core exam (301) which tests advanced skills mainly about ldap. To achieve this level you also need LPIC Level 2 and pass a specialty exam (302 or 303). Exam 302 mainly focuses on Samba, and 303 on advanced security. More info on http://www.lpi.org.

A.1.1.4. LPI DevOps Tools Engineer

certification exam focuses on the practical skills required to work successfully in a DevOps environment -- focusing on the skills needed to use the most prominent DevOps tools. The result is a certification that covers the intersection between development and operations, making it relevant for all IT professionals working in the field of DevOps.

A.1.1.5. Ubuntu

When you are LPIC Level 1 certified, you can take a LPI Ubuntu exam (199) and become Ubuntu certified.

A.1.2. Red Hat

The big difference with most other certifications is that there are no multiple choice questions for `RHCSA`. Red Hat Certified System Administrator and Red Hat Certified Engineer have to take a live exam consisting of two parts. First, they have to troubleshoot and maintain an existing but broken setup (scoring at least 80 percent), and second they have to install and configure a machine (scoring at least 70 percent).

A.1.3. MySQL

There are two tracks for MySQL certification; Certified MySQL 5.6 Developer (CMDEV) and Certified MySQL 5.6 DBA (CMDBA). The `CMDEV` is focused towards database application developers, and the `CMDBA` towards database administrators. Both tracks require two exams each. The MySQL cluster DBA certification requires CMDBA certification and passing the CMCDBA exam.

A.1.4. Suse SLA/SCE

To become a `Suse Certified Linux Professional`, you have to take a live practicum. This is a VNC session to a set of real SLES servers. You have to perform several tasks and are free to choose your method (commandline or YaST or ...). No multiple choice involved.

A.1.5. Other certifications

There are many other lesser known certifications like EC council's Certified Ethical Hacker, CompTIA's Linux+, and Sair's Linux GNU.

B. keyboard settings

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Serge Van Ginderachter, https://github.com/srgvg/)

B.1. about keyboard layout

Many people (like US-Americans) prefer the default US-qwerty keyboard layout. So when you are not from the USA and want a local keyboard layout on your system, then the best practice is to select this keyboard at installation time. Then the keyboard layout will always be correct. Also, whenever you use ssh to remotely manage a Linux system, your local keyboard layout will be used, independent of the server keyboard configuration. So you will not find much information on changing keyboard layout on the fly on linux, because not many people need it. Below are some tips to help you.

B.2. X Keyboard Layout

This is the relevant portion in /etc/X11/xorg.conf, first for Belgian azerty, then for US-qwerty.

```
[student@linux ~]$ grep -i xkb /etc/X11/xorg.conf
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "be"

[student@linux ~]$ grep -i xkb /etc/X11/xorg.conf
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "us"
```

When in Gnome or KDE or any other graphical environment, look in the graphical menu in preferences, there will be a keyboard section to choose your layout. Use the graphical menu instead of editing xorg.conf.

B.3. shell keyboard layout

When in bash, take a look in the /etc/sysconfig/keyboard file. Below a sample US-qwerty configuration, followed by a Belgian azerty configuration.

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


[student@linux ~]$ cat /etc/sysconfig/keyboard
KEYBOARDTYPE="pc"
KEYTABLE="be-latin1"
```

B. keyboard settings

The keymaps themselves can be found in /usr/share/keymaps or /lib/kbd/keymaps.

```
[student@linux ~]$ ls -l /lib/kbd/keymaps/
total 52
drwxr-xr-x 2 root root 4096 Apr  1 00:14 amiga
drwxr-xr-x 2 root root 4096 Apr  1 00:14 atari
drwxr-xr-x 8 root root 4096 Apr  1 00:14 i386
drwxr-xr-x 2 root root 4096 Apr  1 00:14 include
drwxr-xr-x 4 root root 4096 Apr  1 00:14 mac
lrwxrwxrwx 1 root root    3 Apr  1 00:14 ppc -> mac
drwxr-xr-x 2 root root 4096 Apr  1 00:14 sun
```

C. hardware

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

C.1. buses

C.1.1. about buses

Hardware components communicate with the `Central Processing Unit` or `cpu` over a `bus`. The most common buses today are `usb`, `pci`, `agp`, `pci-express` and `pcmcia` aka `pc-card`. These are all `Plag and Play` buses.

Older x86 computers often had `isa` buses, which can be configured using `jumpers` or `dip switches`.

C.1.2. /proc/bus

To list the buses recognised by the Linux kernel on your computer, look at the contents of the `/proc/bus/` directory (screenshot from Ubuntu 7.04 and RHEL4u4 below).

```
root@linux:~# ls /proc/bus/
input  pccard  pci  usb
```

```
[root@linux ~]# ls /proc/bus/
input  pci  usb
```

Can you guess which of these two screenshots was taken on a laptop ?

C.1.3. /usr/sbin/lsusb

To list all the usb devices connected to your system, you could read the contents of `/proc/bus/usb/devices` (if it exists) or you could use the more readable output of `lsusb`, which is executed here on a SPARC system with Ubuntu.

```
root@shaka:~# lsusb
Bus 001 Device 002: ID 0430:0100 Sun Microsystems, Inc. 3-button Mouse
Bus 001 Device 003: ID 0430:0005 Sun Microsystems, Inc. Type 6 Keyboard
Bus 001 Device 001: ID 04b0:0136 Nikon Corp. Coolpix 7900 (storage)
root@shaka:~#
```

C.1.4. /var/lib/usbutils/usb.ids

The `/var/lib/usbutils/usb.ids` file contains a gzipped list of all known usb devices.

```
student@linux:~$ zmore /var/lib/usbutils/usb.ids | head
------> /var/lib/usbutils/usb.ids <------
#
#   List of USB ID's
#
#   Maintained by Vojtech Pavlik <vojtech@suse.cz>
#   If you have any new entries, send them to the maintainer.
#   The latest version can be obtained from
#       http://www.linux-usb.org/usb.ids
#
# $Id: usb.ids,v 1.225 2006/07/13 04:18:02 dbrownell Exp $
```

C.1.5. /usr/sbin/lspci

To get a list of all pci devices connected, you could take a look at `/proc/bus/pci` or run `lspci` (partial output below).

```
student@linux:~$ lspci
 ...
00:06.0 FireWire (IEEE 1394): Texas Instruments TSB43AB22/A IEEE-139...
00:08.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-816...
00:09.0 Multimedia controller: Philips Semiconductors SAA7133/SAA713...
00:0a.0 Network controller: RaLink RT2500 802.11g Cardbus/mini-PCI
00:0f.0 RAID bus controller: VIA Technologies, Inc. VIA VT6420 SATA ...
00:0f.1 IDE interface: VIA Technologies, Inc. VT82C586A/B/VT82C686/A...
00:10.0 USB Controller: VIA Technologies, Inc. VT82xxxxx UHCI USB 1....
00:10.1 USB Controller: VIA Technologies, Inc. VT82xxxxx UHCI USB 1....
 ...
```

C.2. interrupts

C.2.1. about interrupts

An `interrupt request` or `IRQ` is a request from a device to the CPU. A device raises an interrupt when it requires the attention of the CPU (could be because the device has data ready to be read by the CPU).

Since the introduction of pci, irq's can be shared among devices.

Interrupt 0 is always reserved for the timer, interrupt 1 for the keyboard. IRQ 2 is used as a channel for IRQ's 8 to 15, and thus is the same as IRQ 9.

C.2.2. /proc/interrupts

You can see a listing of interrupts on your system in `/proc/interrupts`.

```
student@linux:~$ cat /proc/interrupts
      CPU0     CPU1
0:  1320048     555  IO-APIC-edge      timer
1:    10224       7  IO-APIC-edge      i8042
7:        0       0  IO-APIC-edge      parport0
8:        2       1  IO-APIC-edge      rtc
10:     3062      21  IO-APIC-fasteoi   acpi
12:      131       2  IO-APIC-edge      i8042
15:    47073       0  IO-APIC-edge      ide1
18:        0       1  IO-APIC-fasteoi   yenta
19:    31056       1  IO-APIC-fasteoi   libata, ohci1394
20:    19042       1  IO-APIC-fasteoi   eth0
21:    44052       1  IO-APIC-fasteoi   uhci_hcd:usb1, uhci_hcd:usb2,...
22:   188352       1  IO-APIC-fasteoi   ra0
23:   632444       1  IO-APIC-fasteoi   nvidia
24:     1585       1  IO-APIC-fasteoi   VIA82XX-MODEM, VIA8237
```

C.2.3. dmesg

You can also use `dmesg` to find irq's allocated at boot time.

```
student@linux:~$ dmesg | grep "irq 1[45]"
[ 28.930069] ata3: PATA max UDMA/133 cmd 0x1f0 ctl 0x3f6 bmdma 0x2090 irq 14
[ 28.930071] ata4: PATA max UDMA/133 cmd 0x170 ctl 0x376 bmdma 0x2098 irq 15
```

C.3. io ports

C.3.1. about io ports

Communication in the other direction, from CPU to device, happens through `IO ports`. The CPU writes data or control codes to the IO port of the device. But this is not only a one way communication, the CPU can also use a device's IO port to read status information about the device. Unlike interrupts, ports cannot be shared!

C.3.2. /proc/ioports

You can see a listing of your system's IO ports via `/proc/ioports`.

```
[root@linux ~]# cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
```

```
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
02f8-02ff : serial
...
```

C.4. dma

C.4.1. about dma

A device that needs a lot of data, interrupts and ports can pose a heavy load on the cpu. With `dma` or `Direct Memory Access` a device can gain (temporary) access to a specific range of the `ram` memory.

C.4.2. /proc/dma

Looking at `/proc/dma` might not give you the information that you want, since it only contains currently assigned `dma` channels for `isa` devices.

```
root@linux:~# cat /proc/dma
1: parport0
4: cascade
```

`pci` devices that are using dma are not listed in `/proc/dma`, in this case `dmesg` can be useful. The screenshot below shows that during boot the parallel port received dma channel 1, and the Infrared port received dma channel 3.

```
root@linux:~# dmesg | egrep -C 1 'dma 1|dma 3'
[   20.576000] parport: PnPBIOS parport detected.
[   20.580000] parport0: PC-style at 0x378 (0x778), irq 7, dma 1 ...
[   20.764000] irda_init()
--
[   21.204000] pnp: Device 00:0b activated.
[   21.204000] nsc_ircc_pnp_probe() : From PnP, found firbase 0x2F8 ...
[   21.204000] nsc-ircc, chip->init
```

D. GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

D.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.

D.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

D.3. VERBATIM COPYING

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

D.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.

D.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.

D.6. COMBINING DOCUMENTS

D.7. COLLECTIONS OF DOCUMENTS

D.8. AGGREGATION WITH INDEPENDENT WORKS

D.9. TRANSLATION

D.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.

D.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.

D.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.