Linux for Data Scientists

Paul Cobbaut Thomas Parmentier Andy Van Maele
Bert Van Vreckem

November 23, 2024

Contents

Feel free to contact the author(s):

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

This book is used as the syllabus for the course "Linux for Data Scientists" for the Bachelor of Applied Computer Science at the HOGENT, Belgium. The contents are based on the Linux Training book series by Paul Cobbaut, with updates and additions written by the HOGENT Linux team.

This book is aimed at students specialising in the Data Engineering track that already have some basic knowledge of Linux. Where the sibling "Linux" course for students in Operations/System Administration focuses on Linux as a server operating system, this course rather discusses how Linux can be used as a platform for task and workflow automation.

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

Part I.

First Linux VM

1. 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.

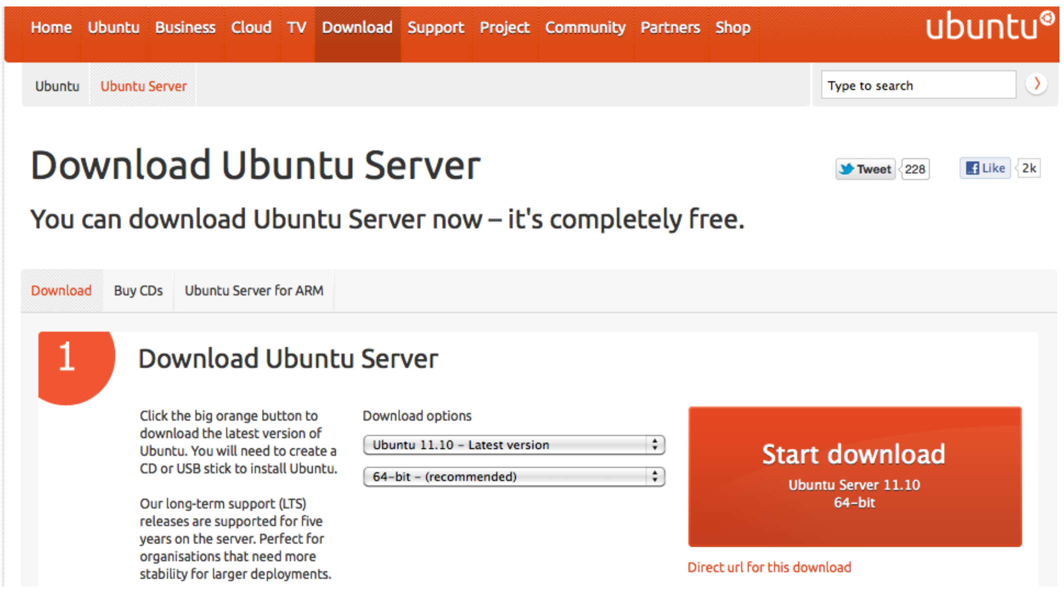
If you do not have access to a Linux computer at the moment, and if you are unable or unsure about installing Linux on your computer, then this chapter proposes a third option: installing Linux in a virtual machine.

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.

1.1. 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.


Home Ubuntu Business Cloud TV Download Support Project Community Partners Shop
ubuntu
Ubuntu Ubuntu Server
Type to search
Tweet 228
Like 2k
Download Buy CDs Ubuntu Server for ARM
1
Download options
Ubuntu 11.10 – Latest version
64-bit – (recommended)
Start download
Ubuntu Server 11.10
64-bit
Direct url for this download

Download Ubuntu Server

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

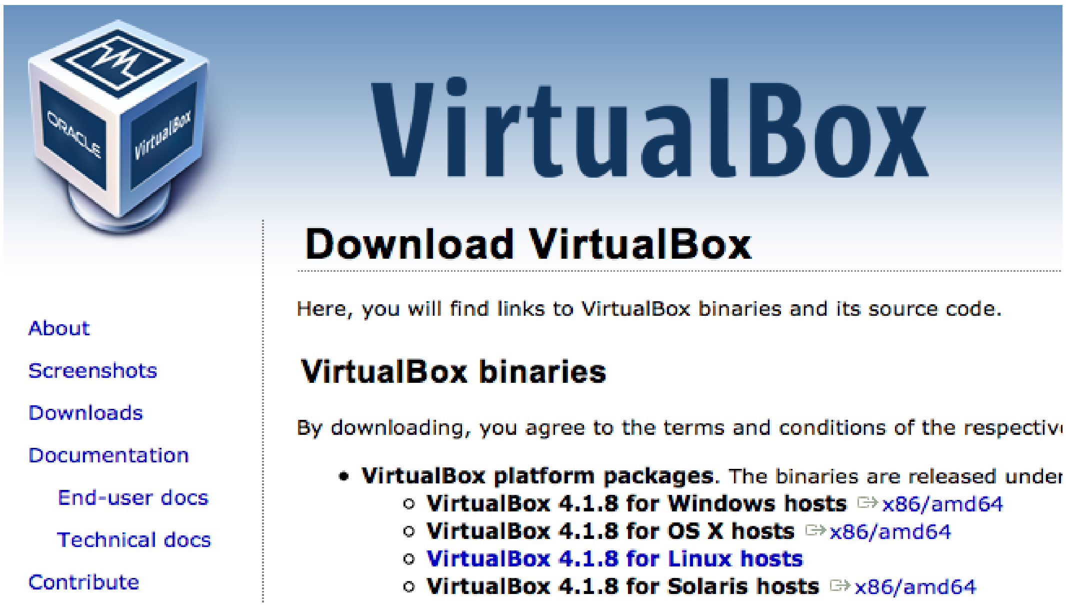
Download Ubuntu Server

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.

1.2. 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

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

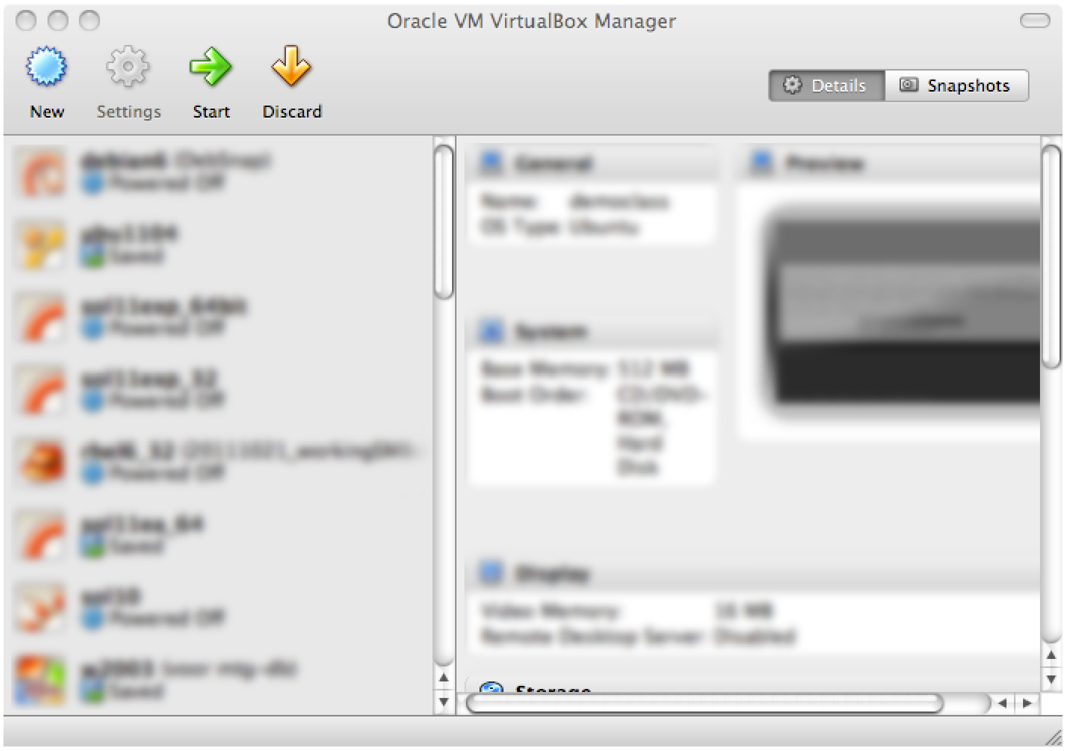
VirtualBox binaries

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

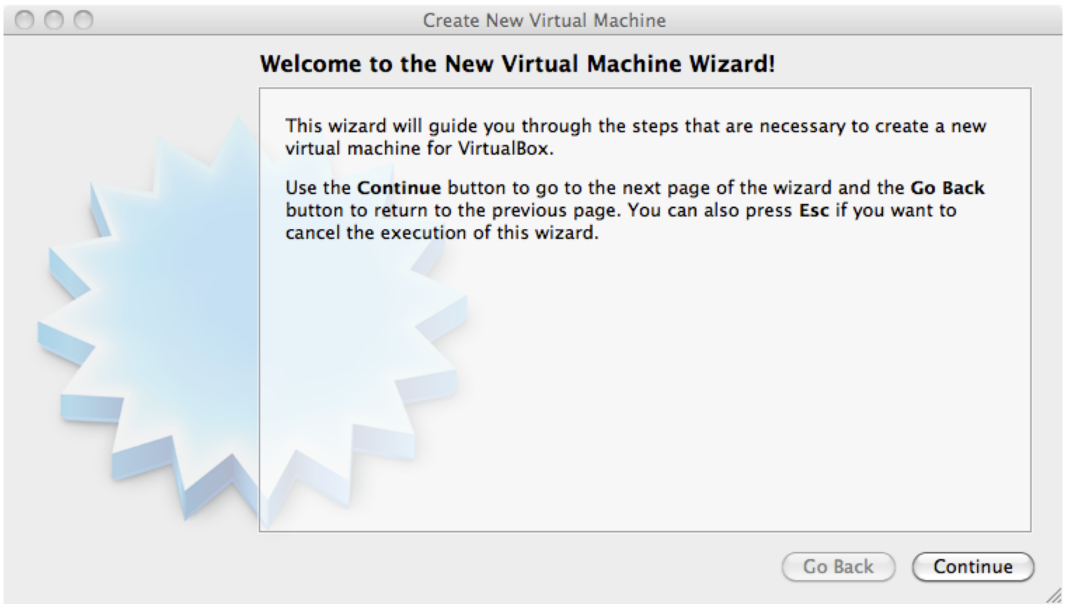
- VirtualBox platform packages. The binaries are released under
  - VirtualBox 4.1.8 for Windows hosts ⇨x86/amd64
  - VirtualBox 4.1.8 for OS X hosts ⇨x86/amd64
  - VirtualBox 4.1.8 for Linux hosts
  - VirtualBox 4.1.8 for Solaris hosts ⇨x86/amd64

1.3. 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.

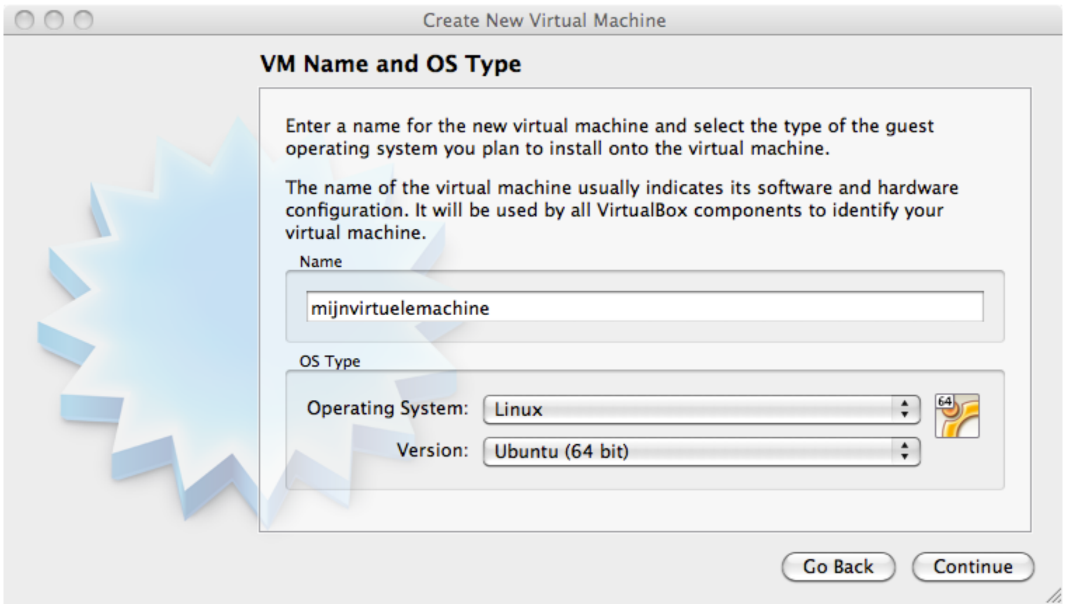


Welcome to the New Virtual Machine Wizard!

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

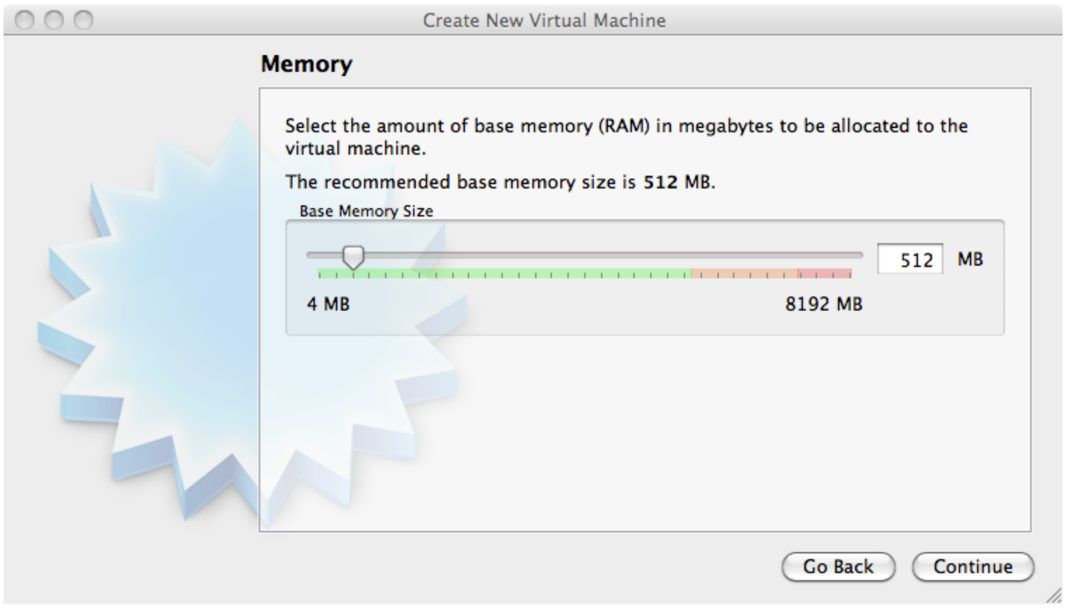
mijnvirtuelemachine

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).



Create New Virtual Machine

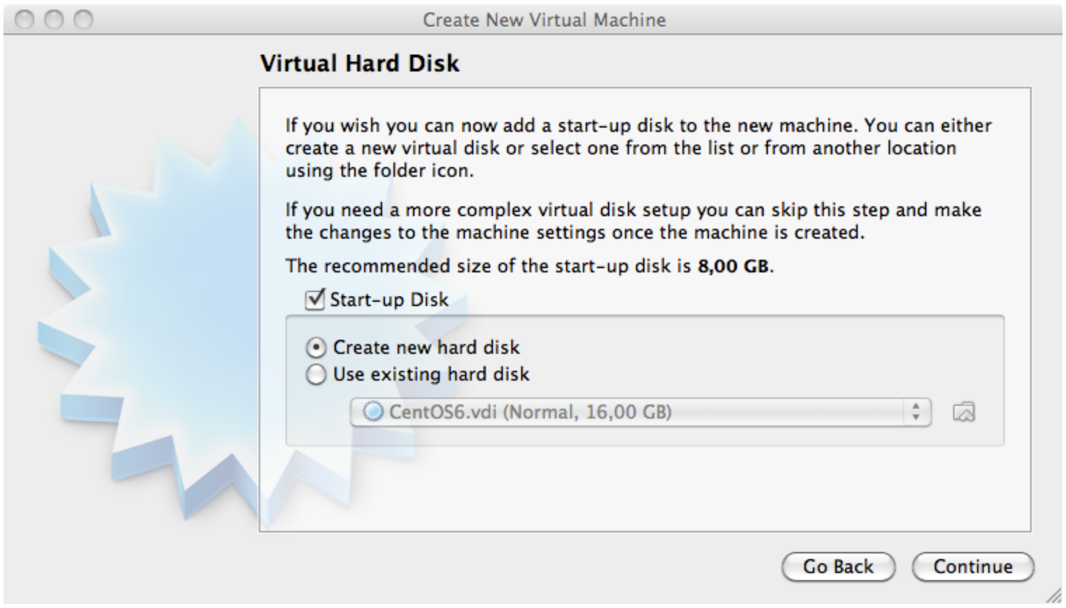
Memory

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).



Create New Virtual Machine

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.

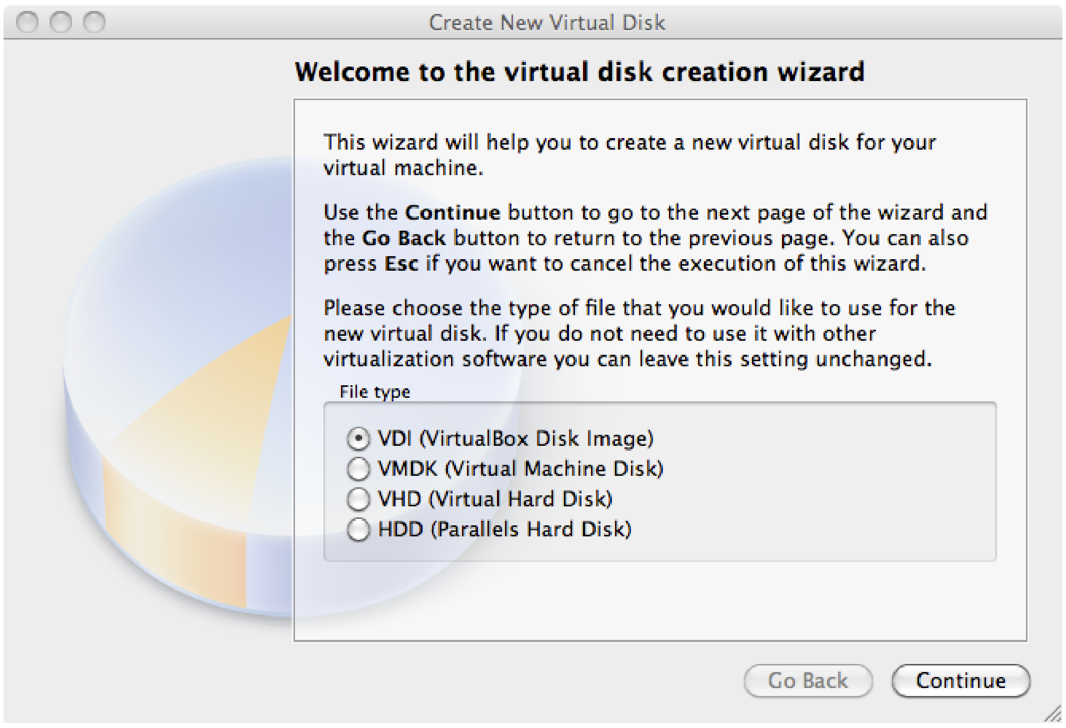
The recommended size of the start-up disk is **8,00 GB**.

☑ 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.



Welcome to the virtual disk creation wizard

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

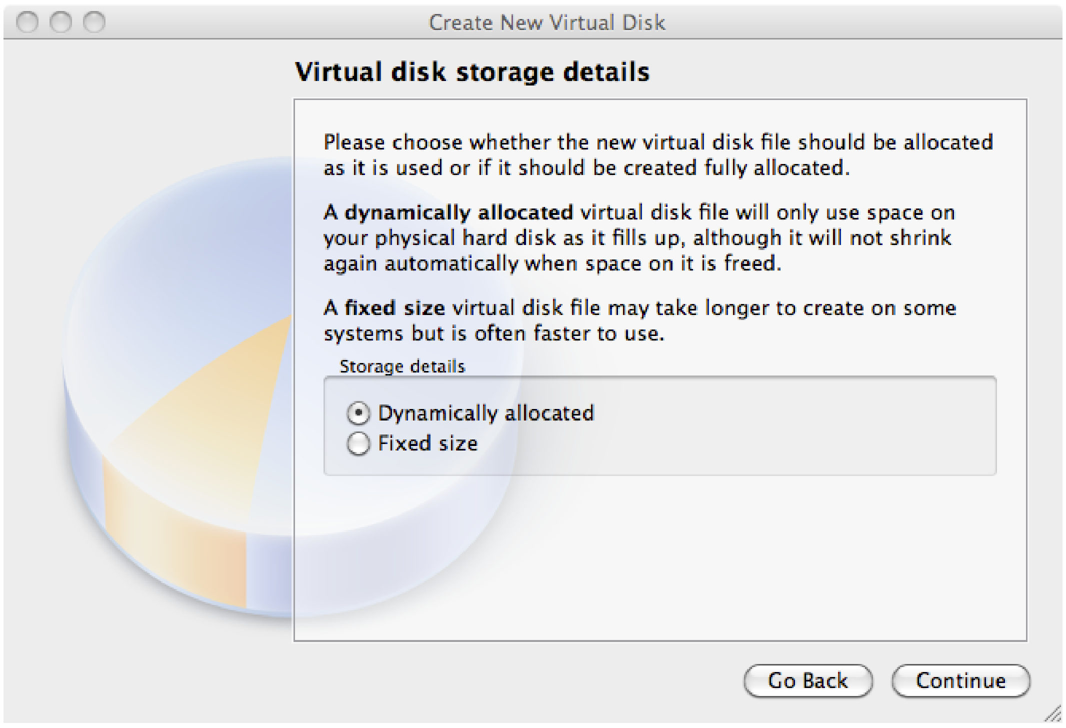
Use the **Continue** button to go to the next page of the wizard and the **Go Back** button to return to the previous page. You can also press **Esc** if you want to cancel the execution of this wizard.

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).



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.

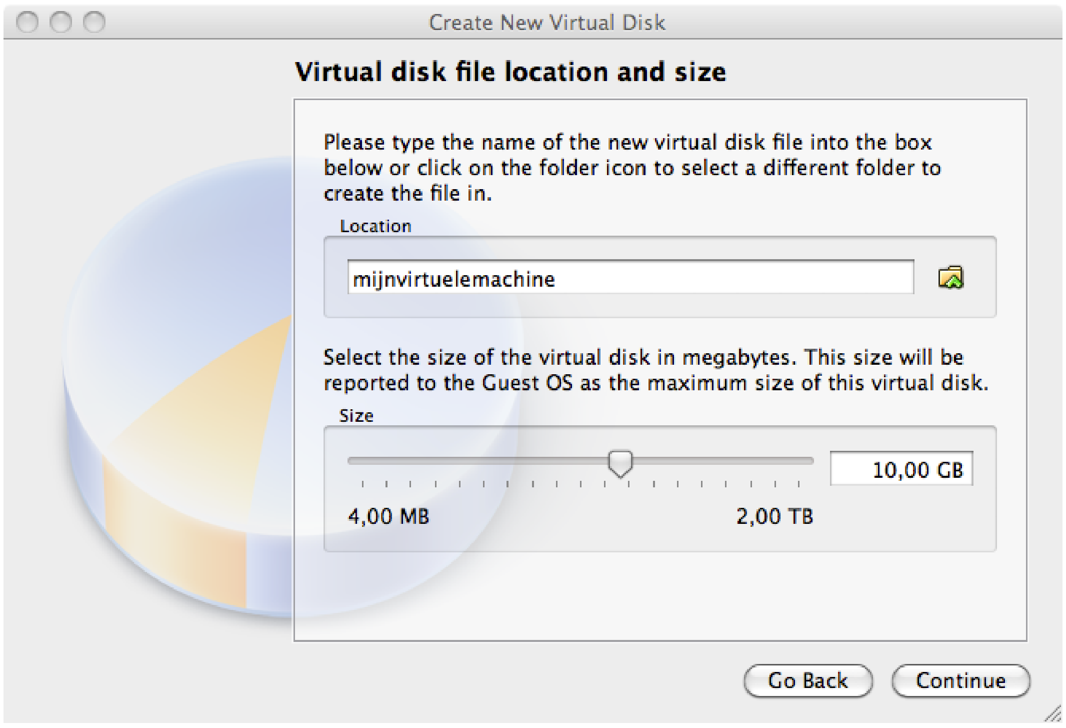
A **dynamically allocated** virtual disk file will only use space on your physical hard disk as it fills up, although it will not shrink again automatically when space on it is freed.

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

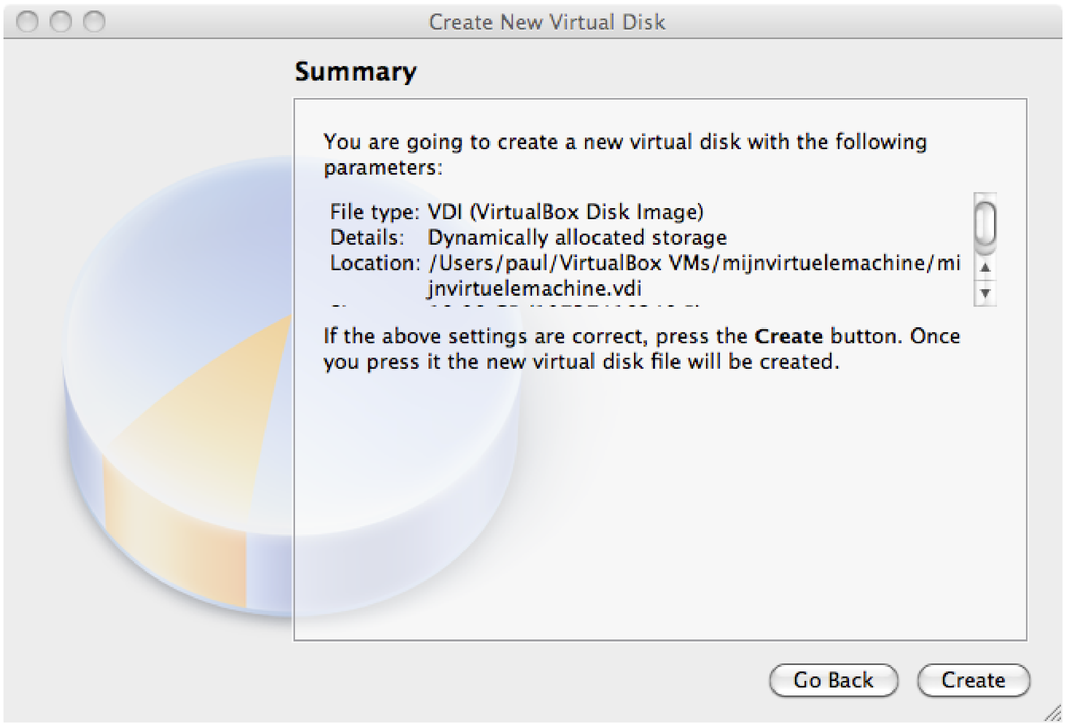
Please type the name of the new virtual disk file into the box below or click on the folder icon to select a different folder to create the file in.

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.



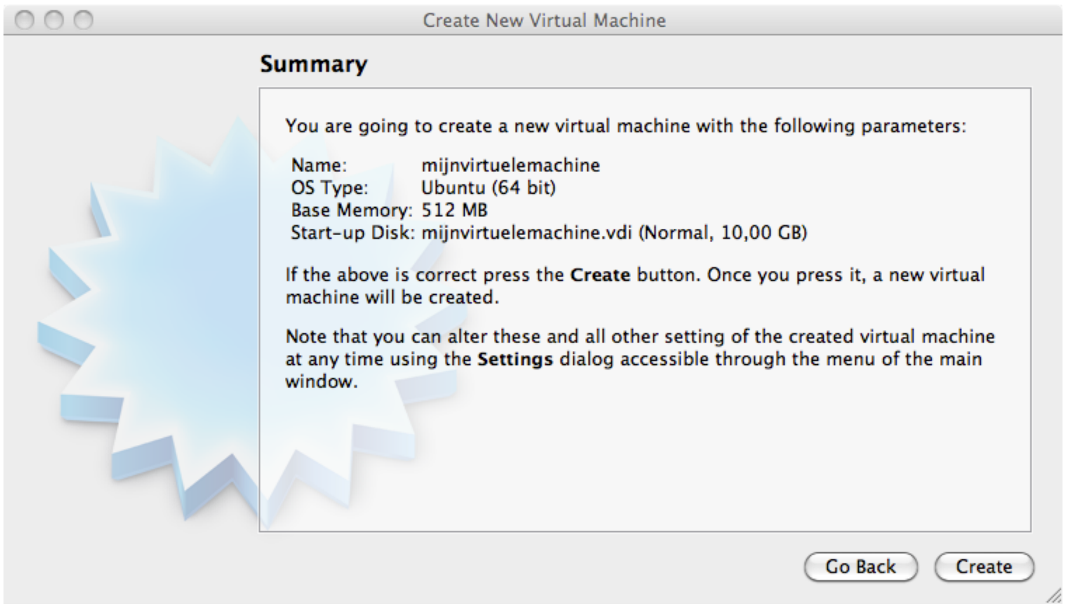
Summary

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/mijnvirtuelemachine.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.


Create New Virtual Machine

Summary

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

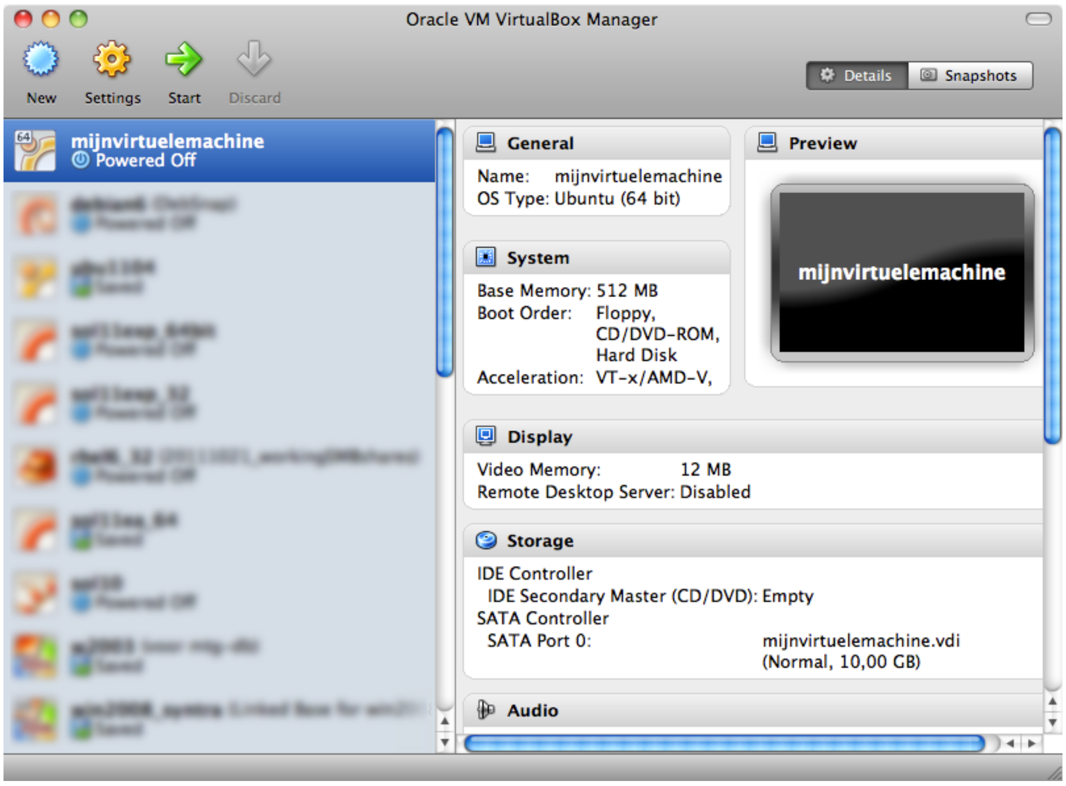
Name:	mijnvirtuelemachine
OS Type:	Ubuntu (64 bit)
Base Memory:	512 MB
Start-up Disk:	mijnvirtuelemachine.vdi (Normal, 10,00 GB)

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.

1.4. attach the CD image

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


Oracle VM VirtualBox Manager

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,

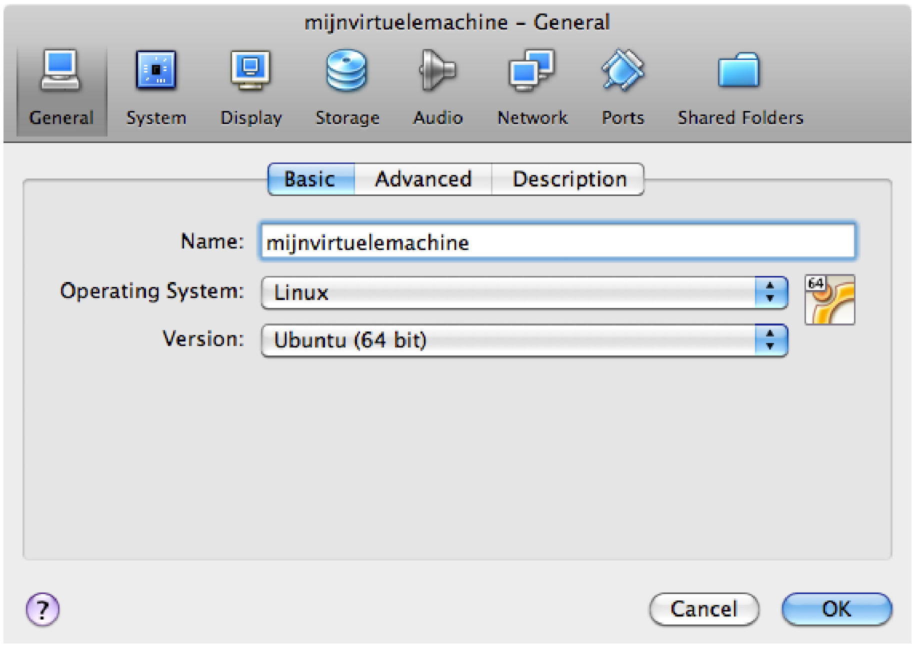
Preview
mijnvirtuelemachine

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)

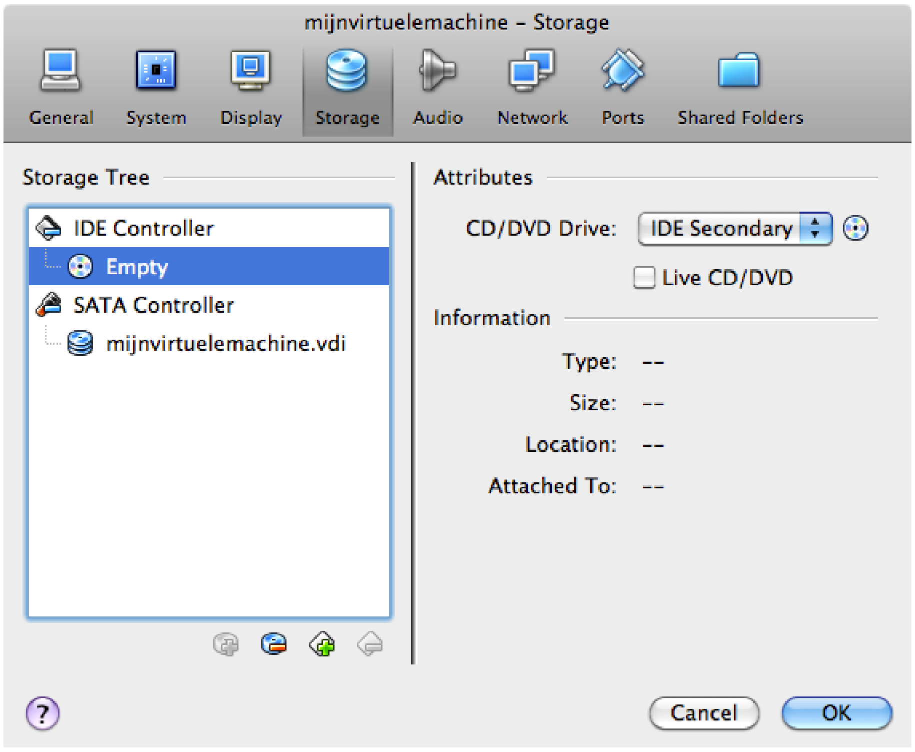
Audio

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



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

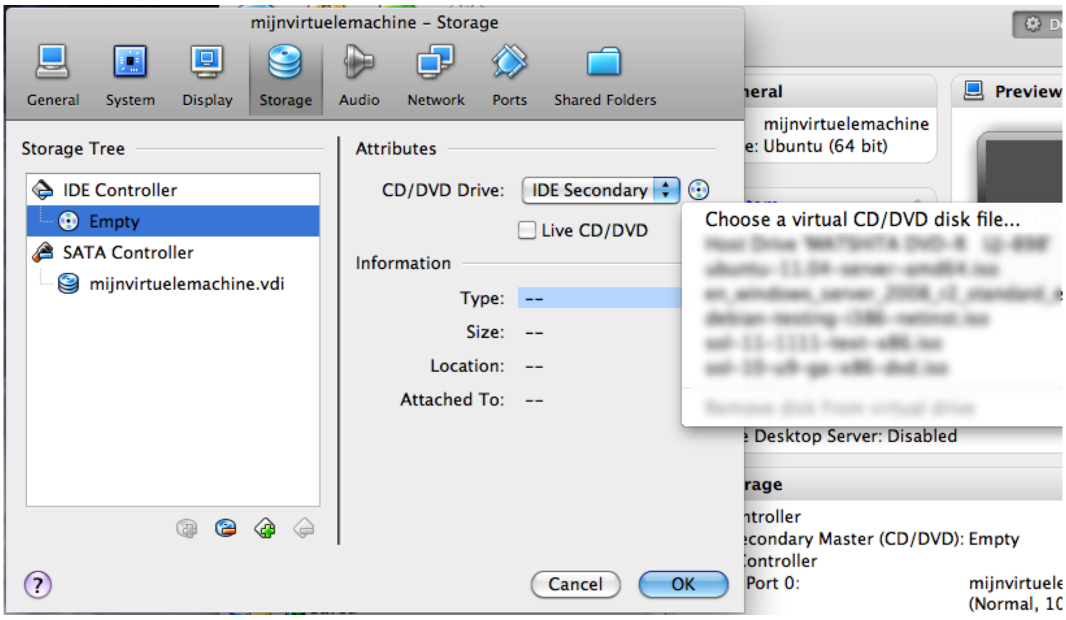
Remember the .ISO file you downloaded? Connect this .ISO file to this virtual machine by clicking on the CD icon next to `Empty`.



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.


mijnvirtuelemachine – Storage

Storage Tree
- IDE Controller
  - Empty
- SATA Controller
  - mijnvirtuelemachine.vdi

Attributes

CD/DVD Drive: IDE Secondary

Live CD/DVD

Information

Type: --

Size: --

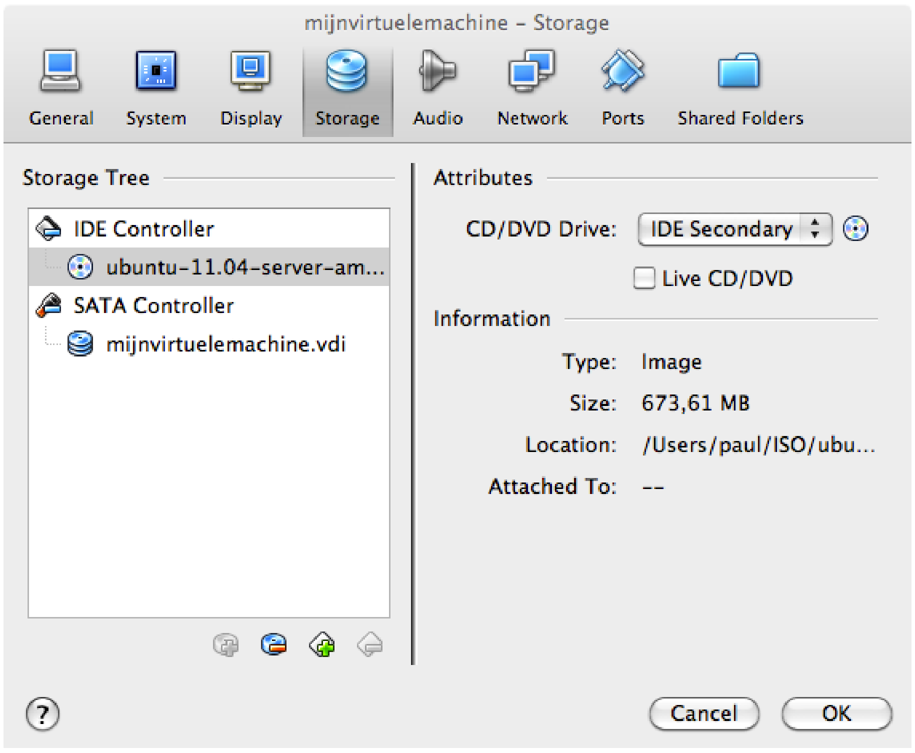
Location: --

Attached To: --

Choose a virtual CD/DVD disk file...

Desktop Server: Disabled

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).


mijnvirtuelemachine – Storage

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

Attributes

CD/DVD Drive: IDE Secondary

Live CD/DVD

Information

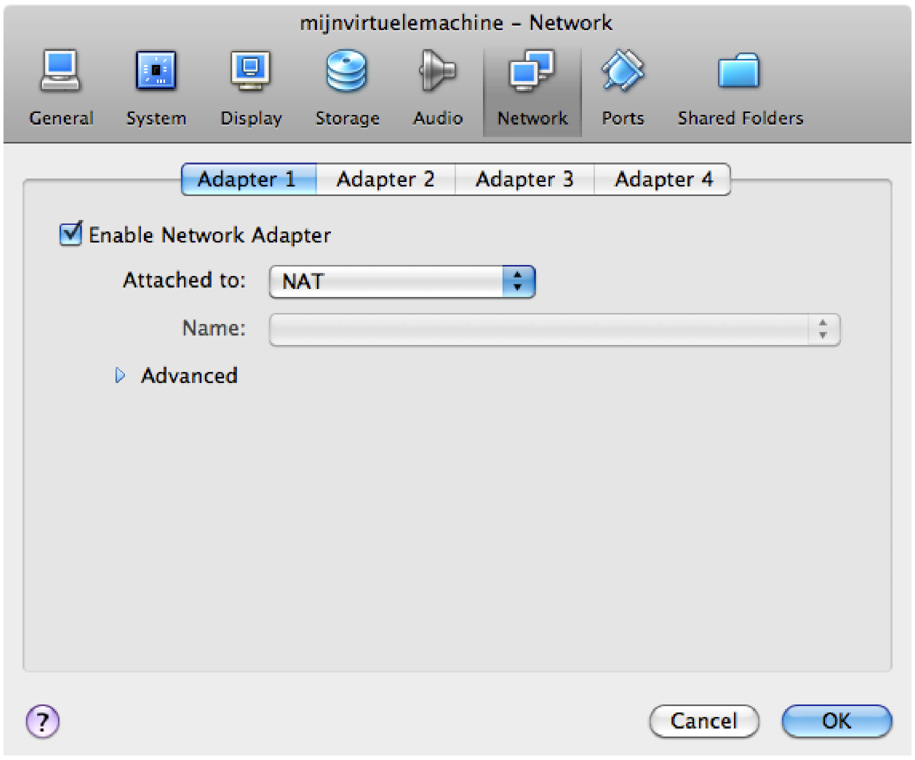
Type: Image

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:

1.5. install Linux

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

Part II.

Software management; curl

2. package 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/)

Most Linux distributions have a **package management** system with online **repositories** containing thousands of packages. This makes it very easy to install, update and remove applications, operating system components, documentation and much more.

In this module, we introduce some basic terminology and concepts. Details about managing packages on the major distributions, specifically the Debian and Enterprise Linux families, are covered in separate modules. In this module, we also discuss a two additional types of package management tools that are not distribution-specific.

- Some programming languages have their own package managers (e.g. `pip` for Python).
- In the 2010s, containerization became popular and started to be used as a solution for package management independent of the Linux distribution, e.g. `snap` and `flatpak`.

Finally, we show how you can install software outside of the package management system, starting from the source code.

2.1. package terminology

2.1.1. repository

A lot of software and documentation for your Linux distribution is available as **packages** in one or more centrally distributed **repositories**. The packages in such a repository are tested and very easy to install (or remove) with a graphical or command line installer.

2.1.2. .deb packages

Debian, Ubuntu, Mint and all derivatives of Debian and Ubuntu use `.deb` packages. To manage software on these systems, you can use `apt` or `apt-get`, both these tools are a front end for `dpkg`.

2.1.3. .rpm packages

Red Hat, Fedora, CentOS, OpenSUSE, Mandriva, Red Flag and others use `.rpm` packages. The tools to manage software packages on these systems are `dnf` and `rpm`.

2.1.4. dependency

Some packages need other packages to function. Tools like `apt-get`, `apt` and `dnf` will install all **dependencies** you need. When using `dpkg` or `rpm`, or when building from **source**, you will need to install dependencies yourself.

2.1.5. open source

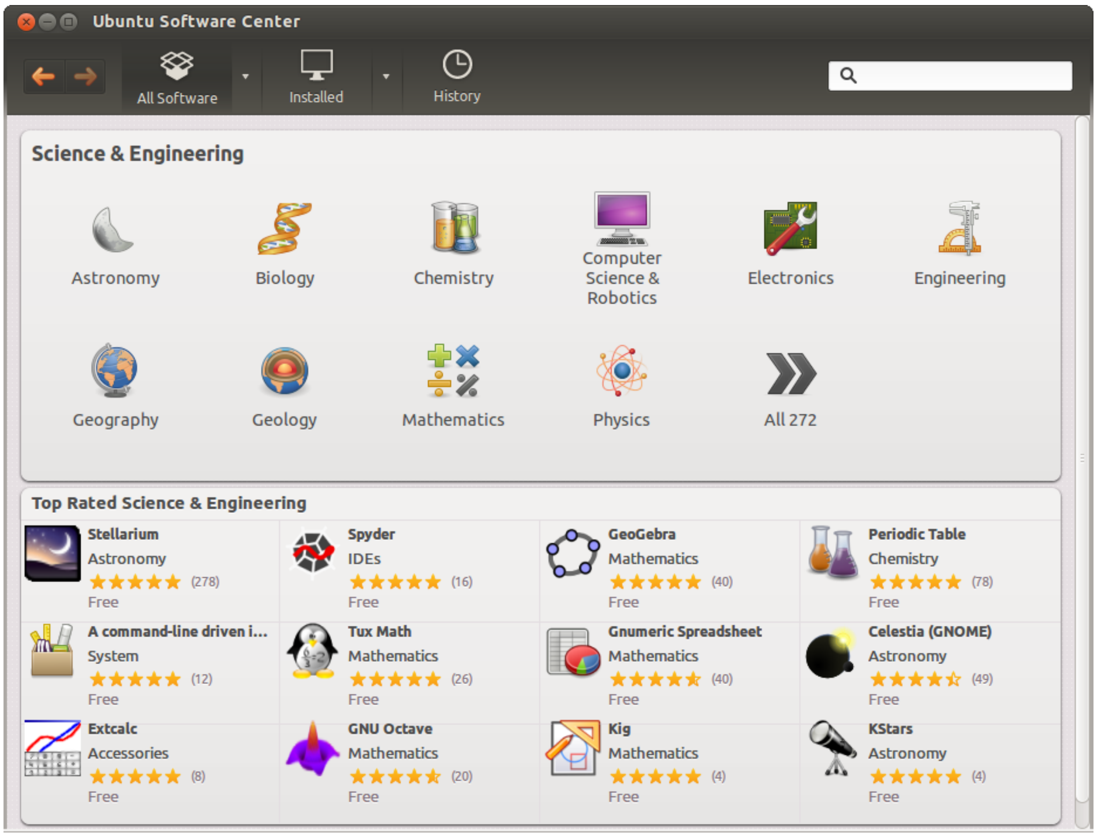
These repositories contain a lot of independent **open source software**. Often the source code is customized to integrate better with your distribution. Most distributions also offer this modified source code as a **package** in one or more **source repositories**.

You are free to go to the project website itself (samba.org, apache.org, github.com ...) and download the *vanilla* (= without the custom distribution changes) source code.

2.1.6. GUI software management

End users have several graphical applications available via the desktop (look for *add/remove software* or something similar).

Below a screenshot of Ubuntu Software Center running on Ubuntu 12.04. Graphical tools are not discussed in this book.


Ubuntu Software Center
All Software
Installed
History
Science & Engineering
Astronomy
Biology
Chemistry
Computer Science & Robotics
Electronics
Engineering
Geography
Geology
Mathematics
Physics
All 272
Top Rated Science & Engineering
Stellarium Astronomy (278) Free
Spyder IDEs (16) Free
GeoGebra Mathematics (40) Free
Periodic Table Chemistry (78) Free
A command-line driven i... System (12) Free
Tux Math Mathematics (26) Free
Gnumeric Spreadsheet Mathematics (40) Free
Celestia (GNOME) Astronomy (49) Free
Extcalc Accessories (8) Free
GNU Octave Mathematics (20) Free
Kig Mathematics (4) Free
KStars Astronomy (4) Free

2.2. pip, the Python package manager

Some programming languages, a.o. Python, have their own package management system that allows you to install applications and/or libraries. In the case of Python, the package manager is called `pip`. It is used to install Python packages from the Python Package Index (PyPI). In fact, there are multiple package managers for Python (a.o. easy_install, conda, etc.), but `pip` is the most widely used.

As a system administrator, or as an end user, this sometimes puts you in a difficult position. Some widely known and used Python libraries can be installed both through your distribution's package manager, and through pip. Which one to choose is not always clear. In general, it is best to use the distribution's package manager, as it will integrate the package into the system and will be updated when the system is updated. However, some packages are not available in the distribution's repositories, or the version you get with `pip` is more recent. In that case, you can use `pip` to install the package.

Another thing to note is that `pip` can be used as a normal user, or as root, and in each case it will install the package in a different location. When you install a package as a normal user, it will be installed in your home directory, and will only be available to you. When you install a package as root, it will be installed system-wide, and will be available to all users. However, if you install a package as root, you will get a warning message:

```
WARNING: Running pip as the 'root' user can result in broken permissions and
 ↪  conflicting behaviour with the system package manager. It is recommended
 ↪  to use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

A virtual environment is a way to create an isolated environment for a Python project, where you can install packages without affecting the system's Python installation. This is especially useful when you are developing Python applications, and you want to make sure that the libraries you use are the same as the ones used in production. Using and managing virtual environments is beyond the scope of this course, but you can find more information in the Python documentation.

As general guidelines, we suggest the following:

- If the library or application is available in the distribution's repositories, use the distribution's package manager to install it.
- Do not install Python libraries or applications system-wide as root using `pip`.
- Normal users may use `pip` to install Python libraries or applications in their home directory.
- Use virtual environments per project to manage Python libraries and applications.

2.2.1. installing pip

By default, `pip` is probably not installed on your system. You can install it using your distribution's package manager.

For example, on Red Hat-based systems, you can install it using `dnf`:

```
student@el ~$ sudo dnf install python3-pip
```

On Debian-based systems, you can install it using `apt`:

```
student@debian:~$ sudo apt install python3-pip
```

However, some Linux distributions like Linux Mint 22 enforce the guidelines mentioned above. When you try to install a Python package using `pip`, you will get an error message:

```
student@linuxmint:~$ sudo apt install python3-pip
student@linuxmint:~$ pip install pandas
error: externally-managed-environment

× This environment is externally managed
⍰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    If you wish to install a non-Debian packaged Python application,
    it may be easiest to use pipx install xyz, which will manage a
    virtual environment for you. Make sure you have pipx installed.

    See /usr/share/doc/python3.12/README.venv for more information.

note: If you believe this is a mistake, please contact your Python
 ↪  installation or OS distribution provider. You can override this, at the
 ↪  risk of breaking your Python installation or OS, by passing
 ↪  --break-system-packages.
hint: See PEP 668 for the detailed specification.
```

In this case, you can't use `pip` at all to install Python packages, not even as a user. So, you should use the distribution's package manager to install Python packages system-wide and virtual environments to install Python packages on a project-by-project basis.

2.2.2. listing packages

You can list the packages installed with `pip` using the `list` command:

```
student@linux:~$ pip list
Package         Version
--------------- --------
dbus-python     1.2.18
distro          1.5.0
gpg             1.15.1
libcomps        0.1.18
nftables        0.1
pip             21.2.3
PyGObject       3.40.1
python-dateutil 2.8.1
PyYAML          5.4.1
rpm             4.16.1.3
selinux         3.5
sepolicy        3.5
setools         4.4.3
setuptools      53.0.0
six             1.15.0
systemd-python  234
```

2.2.3. searching for packages

Searching for packages can **NOT** be done on the command line. To search for packages, you can use the Python Package Index website instead. If you try `pip search`, you will get an error message:

```
student@linux:~$ pip search ansible
ERROR: XMLRPC request failed [code: -32500]
RuntimeError: PyPI no longer supports 'pip search' (or XML-RPC search).
 ↪  Please use https://pypi.org/search (via a browser) instead. See
 ↪  https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-methods
 ↪  for more information.
```

2.2.4. installing packages

You can install a package using the `install` command:

```
student@linux:~$ pip install ansible
```

Just like apt and dnf, pip will install the package and its dependencies.

2.2.5. removing packages

Uninstalling a package is done with the uninstall command:

```
student@linux:~$ pip uninstall ansible
```

Unfortunately, dependencies are not removed when you uninstall a package with pip.

2.2.6. using a virtual environment

A virtual environment allows you to install Python packages in an isolated environment, so they don't interfere with the system's Python installation. You can create a virtual environment using the venv module:

```
student@linuxmint:~$ sudo apt install python3-venv
[ ... output omitted ... ]
student@linuxmint:~$ mkdir pyproject
student@linuxmint:~$ cd pyproject
student@linuxmint:~/pyproject$ python3 -m venv .venv
```

This will create a new directory called .venv in the current directory, which contains scripts and tools to set up the virtual environment. You can choose the name of the directory, but .venv is a common convention. By adding the dot at the beginning of the directory name, it is hidden from the output of the ls command.

```
student@linuxmint:~/pyproject$ ls .venv/
bin  include  lib  lib64  pyvenv.cfg
student@linuxmint:~/pyproject$ ls .venv/bin/
activate  activate.csh  activate.fish  Activate.ps1  f2py  numpy-config  pip
↪  pip3  pip3.12  python  python3  python3.12
```

You can activate the virtual environment using the source command:

```
student@linuxmint:~/pyproject$ source .venv/bin/activate
(.venv) student@linuxmint:~/pyproject$
```

The source command will execute the script *in the current environment* (i.e. without creating a subshell). When the virtual environment is activated, the prompt will change to indicate that you are now working in the virtual environment. You can now use pip to install Python packages, and they will be installed in the virtual environment, not in the system.

```
(.venv) student@linuxmint:~/pyproject$ pip install pandas
Collecting pandas
  Downloading pandas-2.2.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_
↪  x86_64.whl.metadata (89
↪  kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.9/89.9 kB 2.1 MB/s eta
↪  0:00:00
[ ... some output omitted ... ]
Installing collected packages: pytz, tzdata, six, numpy, python-dateutil,
↪  pandas
Successfully installed numpy-2.1.2 pandas-2.2.3 python-dateutil-2.9.0.post0
↪  pytz-2024.2 six-1.16.0 tzdata-2024.2
```

Packages will be installed in the directory `.venv/lib/python<version>/site-packages`.

It is good practice to explicitly list the packages you have installed in the virtual environment in a `requirements.txt` file. This file can be used to recreate the virtual environment on another system. You can create the file using the `freeze` command, or edit it manually.

```
(.venv) student@linuxmint:~/pyproject$ pip freeze > requirements.txt
(.venv) student@linuxmint:~/pyproject$ cat requirements.txt
numpy==2.1.2
pandas==2.2.3
python-dateutil==2.9.0.post0
pytz==2024.2
six==1.16.0
tzdata==2024.2
```

When you are done working in the virtual environment, you can deactivate it using the `deactivate` command:

```
(.venv) student@linuxmint:~/pyproject$ deactivate
student@linuxmint:~/pyproject$
```

If you keep the code in this directory in a Git repository, it's best to add the `.venv` directory to the `.gitignore` file, so it is not included in the repository.

2.3. container-based package managers

With the release of Docker, container-based virtualization has become very popular as a method of distributing and deploying applications on servers. One of the advantages of containers is that they offer a sandbox environment for applications, meaning the application and its dependencies are isolated from the rest of the system. This makes it possible to run applications with different dependencies on the same server, without the risk of conflicts. Containers are also very lightweight, they don't impose much overhead on the host system.

Now, there is no reason why containers can't be used to deploy applications on desktop systems as well. In fact, there are several container-based package managers that allow you to install and run applications in containers on your desktop. The advantage is that third party software vendors can distribute their applications independent of the Linux distribution, so they don't need to maintain different packages for (each family of) distribution(s). The disadvantage is that each application comes with their own dependencies, so you lose the advantage of sharing libraries between applications. Also, since the application is running in a container, it may not integrate well with the rest of the system, or may have only limited permissions to access files or other resources on your computer.

As with many Linux-based technologies, there are multiple tools to choose from. The most popular ones are Flatpak and Snap.

2.3.1. flatpak

Flatpak is a container-based package manager developed by an independent community of contributors, volunteers and supporting organizations. It is available for most Linux distributions and is supported by a large number of third party software vendors. Red Hat was one of the first to endorse Flatpak, and many others followed. Fedora Silverblue is a variant of Fedora that uses Flatpak as its primary package manager. Linux Mint also has Flatpak support enabled by default: in the Software Manager, some applications like Bitwarden, Slack, VS Code, etc. are available as Flatpaks.

If you want to use a container based package manager, Flatpak is probably the best choice for any Linux distribution other than Ubuntu.

In the following example, we'll install the open source password manager Bitwarden with Flatpak on a Linux Mint system. Remark that you don't need to be root to install Flatpak applications!

```
student@mint:~$ flatpak search Bitwarden
Name       Description                            Application ID
 ↪  Version  Branch Remotes
Bitwarden  A secure and free password manager for com.bitwarden.desktop
 ↪  2024.2.0 stable flathub
Goldwarden A Bitwarden compatible desktop client  com.quexten.Goldwarden
 ↪  0.2.13   stable flathub
student@mint:~$ flatpak install Bitwarden
Looking for matches…
Found ref ‘app/com.bitwarden.desktop/x86_64/stable’ in remote ‘flathub’
 ↪  (system).
Use this ref? [Y/n]: y
Required runtime for com.bitwarden.desktop/x86_64/stable
 ↪  (runtime/org.freedesktop.Platform/x86_64/23.08) found in remote flathub
Do you want to install it? [Y/n]: y

com.bitwarden.desktop permissions:
    ipc                    network                       wayland       x11
 ↪  dri       file access [1]
    dbus access [2]        system dbus access [3]

    [1] xdg-download
    [2] com.canonical.AppMenu.Registrar, org.freedesktop.Notifications,
 ↪  org.freedesktop.secrets, org.kde.StatusNotifierWatcher
    [3] org.freedesktop.login1


        ID                                     Branch      Op Remote  Download
 1. [✓] com.bitwarden.desktop.Locale           stable      i  flathub 300.7 kB
 ↪  / 9.8 MB
 2. [✓] org.freedesktop.Platform.GL.default  23.08       i  flathub 162.0 MB
 ↪  / 162.3 MB
 3. [✓] org.freedesktop.Platform.GL.default  23.08-extra i  flathub  17.9 MB
 ↪  / 162.3 MB
 4. [✓] org.freedesktop.Platform.Locale      23.08       i  flathub  17.9 kB
 ↪  / 359.9 MB
 5. [✓] org.freedesktop.Platform             23.08       i  flathub 171.6 MB
 ↪  / 225.6 MB
 6. [✓] com.bitwarden.desktop                stable      i  flathub 132.5 MB
 ↪  / 133.4 MB

Installation complete.
```

To remove a Flatpak application, you can use the `uninstall` command:

```
student@mint:~$ flatpak uninstall Bitwarden
Found installed ref ‘app/com.bitwarden.desktop/x86_64/stable’ (system). Is
 ↪  this correct? [Y/n]: y


        ID                                     Branch         Op
 1. [-] com.bitwarden.desktop                stable         r
 2. [-] com.bitwarden.desktop.Locale         stable         r
```

```

Uninstall complete.
```

2.3.2. snap

Snap was developed by Canonical and is installed by default on Ubuntu. It is also available for other distributions (like the official Ubuntu derivatives, Solus and Zorin OS), but it is not as widely supported as Flatpak. Snap was also designed to work for cloud applications and Internet of Things devices.

In the following example, we'll install Grafana on an Ubuntu Server system.

```
student@ubuntu:~$ snap search grafana
Name          Version Publisher  Notes Summary
grafana       6.7.4   canonical✓ -     feature rich metrics dashboard and
  ↪ graph editor
grafana-agent 0.35.4  0×12b      -     Telemetry Agent
[...]
student@ubuntu:~$ sudo snap install grafana
grafana 6.7.4 from Canonical✓ installed
```

To uninstall a Snap application, you can use the `remove` command:

```
student@ubuntu:~$ sudo snap remove grafana
grafana removed
```

2.4. downloading software outside the repository

These days, the case where you need software that is not available as a binary package has become exceedingly rare. However, *if* you want to install some experimental tool that hasn't been packaged yet, or you want to test the very latest experimental version of an application, you may have to download the source code and compile it yourself. Usually, the source code is available on the project's website or on a code hosting platform like GitHub, GitLab or Bitbucket. You then either download the source code as a `tgz`, `.tar.gz`, `.tar.bz2`, `tar.xz` file (also called a *tarball*) or you can clone the repository using `git`.

In the example below, we assume that you have downloaded the source code of an application written in C or C++, as is common for many Linux applications. Remark that in order to be able to compile the source code, you need to have the C compiler `gcc` and the build tool `make` installed on your system. You can install these using your distribution's package manager. Also, many applications depend on other libraries, which also have to be installed as source.

2.4.1. example: compiling zork

As an example, we will download the source code for Zork, an ancient text based adventure game, and compile it on a Fedora system. The source code is available on GitHub. Remark that you should check wheter `git`, `gcc` and `make` are installed beforehand. See the modules on RPM and DEB package management for more information.

```
[student@fedora ~]$ git clone https://github.com/devshane/zork.git
Cloning into 'zork'...
remote: Enumerating objects: 79, done.
remote: Total 79 (delta 0), reused 0 (delta 0), pack-reused 79
Receiving objects: 100% (79/79), 241.70 KiB | 2.14 MiB/s, done.
Resolving deltas: 100% (20/20), done.
```

```
[student@fedora ~]$ cd zork/
[student@fedora zork]$ ls
actors.c  demons.c  dmain.c  dso3.c  dso6.c  dtextc.dat  dverb2.c  history
 ↪  Makefile  np2.c  nrooms.c  README.md   sobjs.c   vars.h
ballop.c  dgame.c   dso1.c   dso4.c  dso7.c  dungeon.6   funcs.h   lightp.c
 ↪  nobjs.c   np3.c  objcts.c  readme.txt  supp.c    verbs.c
clockr.c  dinit.c   dso2.c   dso5.c  dsub.c  dverb1.c    gdt.c     local.c
 ↪  np1.c     np.c   parse.h  rooms.c     sverbs.c  villns.c
[student@fedora ~]$ make
cc -g    -c -o actors.o actors.c
cc -g    -c -o ballop.o ballop.c
cc -g    -c -o clockr.o clockr.c
[ ... etc ... ]
cc -g  -o zork actors.o ballop.o clockr.o demons.o dgame.o dinit.o dmain.o
 ↪  dso1.o dso2.o dso3.o dso4.o dso5.o dso6.o dso7.o dsub.o dverb1.o
 ↪  dverb2.o gdt.o lightp.o local.o nobjs.o np.o np1.o np2.o np3.o nrooms.o
 ↪  objcts.o rooms.o sobjs.o supp.o sverbs.o verbs.o villns.o -ltermcap
/usr/bin/ld: cannot find -ltermcap: No such file or directory
collect2: error: ld returned 1 exit status
make: *** [Makefile:69: dungeon] Error 1
```

As you can see, the make command fails because it cannot find the `termcap` library. This is a library that is used to control the terminal, and it is not installed on our system. This is a common problem when you try to install packages from source. You need to install these dependencies yourself and these are not always easy to find. In this case, we can install the `ncurses-devel` library, which is a modern replacement for `termcap`. How did we now that? We used `dnf provides` to find library files that contain the string `termcap` (remark that the command took a long time to finish):

```
[student@fedora zork]$ dnf provides '*libtermcap.so*'
Last metadata expiration check: 1:56:05 ago on Mon 26 Feb 2024 05:46:43 PM
 ↪  UTC.
ncurses-devel-6.4-7.20230520.fc39.i686 : Development files for the ncurses
 ↪  library
Repo        : fedora
Matched from:
Other       : *libtermcap.so*
[student@fedora ~]$ sudo dnf install ncurses-devel
[ ... etc ... ]
```

Let's try to compile again:

```
[student@fedora zork]$ make
cc -g    -c -o actors.o actors.c
cc -g    -c -o ballop.o ballop.c
[ ... etc ... ]
cc -g    -c -o villns.o villns.c
cc -g  -o zork actors.o ballop.o clockr.o demons.o dgame.o dinit.o dmain.o
 ↪  dso1.o dso2.o dso3.o dso4.o dso5.o dso6.o dso7.o dsub.o dverb1.o
 ↪  dverb2.o gdt.o lightp.o local.o nobjs.o np.o np1.o np2.o np3.o nrooms.o
 ↪  objcts.o rooms.o sobjs.o supp.o sverbs.o verbs.o villns.o -ltermcap
[student@fedora zork]$
```

The command seems to have succeeded. The current directory now contains a new file called `zork`. This is the compiled application and it has execute permissions. You can run it by typing `./zork`:

```
[student@fedora zork]$ ls -l zork
-rwxr-xr-x. 1 student student 400968 Feb 26 19:45 zork
[student@fedora zork]$ file zork
```

```
zork: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
  ↪  linked, interpreter /lib64/ld-linux-x86-64.so.2,
  ↪  BuildID[sha1]=3089e3cb1c1a7fc1cc1db41c3aa578c0b52f83f3, for GNU/Linux
  ↪  3.2.0, with debug_info, not stripped
[student@fedora zork]$ ./zork
Welcome to Dungeon.                     This version created 11-MAR-91.
You are in an open field west of a big white house with a boarded
front door.
There is a small mailbox here.
>
```

In this case, installing the game is as simple as copying the `zork` file to a directory in your `PATH`, like `/usr/local/bin` or (for a computer game) `/usr/local/games`. However, most Makefiles provide a way to install the application in the system, usually by running `make install`. This will copy the executable, manual pages and other documentation to the correct location.

```
[student@fedora zork]$ sudo make install
mkdir -p /usr/games  /usr/share/man/man6
cp zork /usr/games
cp dtextc.dat /usr/games/lib
cp dungeon.6 /usr/share/man/man6/
```

Remark that the "official" location where manually installed applications belong in a Linux directory structure is `/usr/local` (for applications that follow the Filesystem Hierarchy Standard) or `/opt` (for applications that want to keep all files in a single directory).

2.4.2. installing from a tarball

Before unpacking a tarball, it's useful to check its contents:

```
student@linux:~$ tar tf $downloadedFile.tgz
```

The `t` option lists the content of the archive, `f` should be followed by the filename of the tarball. For `.tgz`, you may add option `z` and for `.tar.bz2` option `j`. However, the `tar` command should recognize the compression method automatically.

Check whether the package archive unpacks in a subdirectory (which is the preferred case) or in the current directory and create a subdirectory yourself if necessary. After that, you can unpack the tarball:

```
student@linux:~$ tar xf $downloadedFile.tgz
```

Now, be sure to read the README file carefully! Normally the readme will explain what to do after download.

Usually the steps are always the same three:

1. running a script `./configure`. It will gather information about your system that is needed to compile the software so that it can actually run on your system
2. executing the command `make` (which is the actual compiling)
3. finally, executing `make install` to copy the files to their proper location.

2.5. practice: package management

1. Look up where your Linux distribution has published their package repositories. What is the URL? Can you find and download individual packages from there?
2. Open the graphical package manager tool on your system. What kind of software is available? Can you find the software you need? Or games that you would like to play?
3. Why would it be a bad idea to install Python packages with `pip` as root? When would you install a package (e.g. `pandas`) with `pip` instead of the package manager (e.g. `python3-pandas`)?
4. Check whether Snap or Flatpak is installed or available on your system. How can you search online for applications in these repositories?

2.6. solution: package management

1. Look up where your Linux distribution has published their package repositories. What is the URL? Can you find and download individual packages from there?

   For Ubuntu, you can start by going to http://archive.ubuntu.com/ubuntu/. Information about which packages are available can be found in the `dists` directory. For example, the release Noble Nombat (24.04 LTS) has several subdirectories called `noble`, `noble-backports`, `noble-proposed`, and `noble-security`. The `pool` directory contains the actual `.deb` files.

   For Fedora, you can go to https://mirrormanager.fedoraproject.org and look for a mirror for the Fedora release you are using and the architecture of your machine (typically `x86_64`). An example of such a repository mirror is https://fedora.cu.be/linux/releases/ (in Belgium). You can then navigate to the subdirectory of your release (e.g. 40), and then to the `Everything` directory, `x86_64`, `os`, and `Packages` to find the RPM packages (sorted in subdirectories `a` to `z` according to the first letter of the package name).

2. Open the graphical package manager tool on your system. What kind of software is available? Can you find the software you need? Or games that you would like to play?

   On Ubuntu, you can use the *Software Center*. On Fedora, you can use the *Software* application. Both tools provide a graphical interface to search for and install software, just like you would on the Appstore, Google Play store, Windows store, etc. In fact, graphical package managers on Linux predate these stores by many years.

   You may not necessarily find commercial applications, but usually open source alternatives exist.

   Installing games is also possible, but the selection is not as large as on Steam. However, you can install Steam on Linux and play many games that are available on that platform.

3. Why would it be a bad idea to install Python packages with `pip` as root? When would you install a package (e.g. `pandas`) with `pip` instead of the package mnanager (e.g. `python3-pandas`)?

   Installing Python packages with `pip` as root could interfere with the Python packages installed with the system package manager. Generally, if you need Python packages to be available system-wide, use the system package manager.

If you only need the package for a specific user or a specific Python virtual environment, you can use `pip`. This is especially useful if the package is not available in the system package manager, or if you need a specific version of the package that is not available in the system package manager.

4. Check whether Snap or Flatpak is installed or available on your system. How can you search online for applications in these repositories?

   On Ubuntu, Snap is installed by default. You can search for applications in the Snap store by going to https://snapcraft.io/store. You can also use the `snap find` command to search for applications from the command line.

   On most other distributions, Flatpak is the default (if it is installed). You can search for applications in the Flathub repository by going to https://flathub.org/. You can also use the `flatpak search` command to search for applications from the command line.

   Remark that installing Snap is usually also possible on distributions that default to Flatpak.

3. package management on debian

(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/)

In this module, we discuss the Debian package format `.deb` and its tools `dpkg`, `apt-get` and `apt`. This is the package manager all Linux distributions derived from Debian, a.o. Ubuntu and Mint.

3.1. about deb

Most people use `apt` or `apt-get` (APT = Advanced Package Tool) to manage their Debian/Ubuntu family of Linux distributions. Both are a front end for `dpkg` and are themselves a back end for *synaptic* and other graphical tools.

3.2. dpkg

You could use `dpkg -i` to install a package and `dpkg -r` to remove a package, but you'd have to manually download the packge and keep track of dependencies. Using `apt-get` or `apt` is much easier (see below).

3.2.1. dpkg -l

The low level tool to work with `.deb` packages is `dpkg`. Among other things, you can use `dpkg` to list all installed packages on a Debian server.

```
student@debian:~$ dpkg -l | wc -l
365
```

Compare this to the same list on a Linux Mint system with a graphical desktop installed.

```
student@mint:~$ dpkg -l | wc -l
2118
```

3.2.2. dpkg -l $package

Here is an example on how to get information on an individual package. The `ii` at the beginning means the package is installed.

```
student@debian:~$ dpkg -l rsync
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-p
 ↪  end
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name           Version        Architecture Description
```

```
+++-===========================================================================
 ↪  ========================
ii  rsync          3.2.7-1ubuntu1 amd64        fast, versatile, remote (and
 ↪  local) file-copying tool
```

3.2.3. dpkg -S

You can find the package responsible for installing a certain file on your computer using dpkg -S. This example shows how to find the package for three files on a typical Debian server.

```
student@debian:~$ dpkg  -S /usr/share/doc/tmux/ /etc/ssh/ssh_config
 ↪  /sbin/ifconfig
dpkg-query: no path found matching pattern /usr/share/doc/tmux/
openssh-client: /etc/ssh/ssh_config
net-tools: /sbin/ifconfig
```

3.2.4. dpkg -L

In reverse, you can also get a list of all files and directories that have been installed by a certain program. Below is the list for the curl package.

```
student@debian:~$ dpkg -L curl
/.
/usr
/usr/bin
/usr/bin/curl
/usr/share
/usr/share/doc
/usr/share/doc/curl
/usr/share/doc/curl/changelog.Debian.gz
/usr/share/doc/curl/changelog.gz
/usr/share/doc/curl/copyright
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/curl.1.gz
/usr/share/zsh
/usr/share/zsh/vendor-completions
/usr/share/zsh/vendor-completions/_curl
```

3.3. apt-get

Debian has been using apt-get to manage packages since 1998. Today Debian and many Debian-based distributions still actively support apt-get, though some experts claim apt, released in 2014, is better at handling dependencies than apt-get.

Both commands use the same configuration files and can be used interchangeably, at least on an interactive terminal. However, using apt in a script is not recommended, since the options and behaviour of apt are not considered to be stable.

We will start with apt-get and discuss apt in the next section. Whenever you see apt-get in documentation, feel free to type apt instead.

3.3.1. apt-get update

When typing `apt-get update` you are downloading the names, versions and short description of all packages available on all configured repositories for your system. Remark that you need to be root to run this command.

```
student@debian:~$ apt-get update
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission
↪ denied)
E: Unable to lock directory /var/lib/apt/lists/
student@debian:~$ sudo apt-get update
Hit:1 http://security.debian.org/debian-security bookworm-security InRelease
Hit:2 http://httpredir.debian.org/debian bookworm InRelease
Hit:3 http://httpredir.debian.org/debian bookworm-updates InRelease
Reading package lists... Done
```

In the example below you can see an interaction with an Ubuntu system. Some repositories are at the url `be.archive.ubuntu.com` because this computer was installed in Belgium. This mirror URL can be different for you.

```
student@ubuntu:~$ sudo apt-get update
Ign http://be.archive.ubuntu.com precise InRelease
Ign http://extras.ubuntu.com precise InRelease
Ign http://security.ubuntu.com precise-security InRelease
Ign http://archive.canonical.com precise InRelease
↪
Ign http://be.archive.ubuntu.com precise-updates InRelease
↪
...
Hit http://be.archive.ubuntu.com precise-backports/main Translation-en
↪
Hit http://be.archive.ubuntu.com precise-backports/multiverse Translation-en
↪
Hit http://be.archive.ubuntu.com precise-backports/restricted Translation-en
↪
Hit http://be.archive.ubuntu.com precise-backports/universe Translation-en
↪
Fetched 13.7 MB in 8s (1682 kB/s)
↪
Reading package lists... Done
student@ubuntu:~$
```

Tips:

- Run `apt-get update` every time before performing other package operations to ensure your metadata is up-to-date.
- Since the package repositories are hosted on web servers, you can open any repository URL in your browser to see how the repository is structured.

3.3.2. apt-get upgrade

One of the nicest features of `apt-get` is that it allows for a secure update of *all software currently installed* on your computer with just *one* command.

```
student@debian:~$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

The above transcript shows that all software is updated to the latest version available for my distribution. Below is an example of a system with software that can be updated. Some lines were ommitted for brevity.

```
student@debian:~$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages have been kept back:
  linux-image-amd64
The following packages will be upgraded:
  base-files bind9-dnsutils bind9-host bind9-libs cryptsetup cryptsetup-bin
  ↪ libcryptsetup12 libgnutls30 libnss-systemd libpam-systemd
  ↪ libsystemd-shared libsystemd0 libudev1 systemd systemd-sysv
  systemd-timesyncd tar tzdata udev usr-is-merged
20 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 13.0 MB of archives.
After this operation, 75.8 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://security.debian.org/debian-security bookworm-security/main
  ↪ amd64 bind9-host amd64 1:9.18.24-1 [305 kB]
[...]
Get:20 http://httpredir.debian.org/debian bookworm/main amd64 cryptsetup
  ↪ amd64 2:2.6.1-4~deb12u2 [213 kB]
Fetched 13.0 MB in 1s (20.3 MB/s)
Reading changelogs... Done
Preconfiguring packages ...
(Reading database ... 29205 files and directories currently installed.)
Preparing to unpack .../base-files_12.4+deb12u5_amd64.deb ...
Unpacking base-files (12.4+deb12u5) over (12.4+deb12u4) ...
Setting up base-files (12.4+deb12u5) ...
Installing new version of config file /etc/debian_version ...
[...]
Preparing to unpack .../5-cryptsetup_2%3a2.6.1-4~deb12u2_amd64.deb ...
Unpacking cryptsetup (2:2.6.1-4~deb12u2) over (2:2.6.1-4~deb12u1) ...
Setting up systemd-sysv (252.22-1~deb12u1) ...
[...]
Setting up bind9-dnsutils (1:9.18.24-1) ...
Processing triggers for initramfs-tools (0.142) ...
update-initramfs: Generating /boot/initrd.img-6.1.0-17-amd64
[...]
Processing triggers for mailcap (3.70+nmu1) ...
```

Tip: Have you noticed that almost every time that you update software on Windows, you are asked to reboot your computer? This is **not** the case with Linux! The only time you need to reboot is when you update the kernel.

3.3.3. apt-get clean

`apt-get` keeps a copy of downloaded packages in `/var/cache/apt/archives`, as can be seen in this screenshot.

```
student@debian:~$ ls /var/cache/apt/archives/ | head
base-files_12.4+deb12u5_amd64.deb
```

```
bind9-dnsutils_1%3a9.18.24-1_amd64.deb
bind9-host_1%3a9.18.24-1_amd64.deb
bind9-libs_1%3a9.18.24-1_amd64.deb
cryptsetup_2%3a2.6.1-4~deb12u2_amd64.deb
cryptsetup-bin_2%3a2.6.1-4~deb12u2_amd64.deb
libcryptsetup12_2%3a2.6.1-4~deb12u2_amd64.deb
libgnutls30_3.7.9-2+deb12u2_amd64.deb
libnss-systemd_252.22-1~deb12u1_amd64.deb
libpam-systemd_252.22-1~deb12u1_amd64.deb
```

Running `apt-get clean` removes all .deb files from that directory.

```
student@debian:~$ sudo apt-get clean
student@debian:~$ ls /var/cache/apt/archives/*.deb
ls: cannot access /var/cache/apt/archives/*.deb: No such file or directory
```

3.3.4. apt-cache search

Use `apt-cache search` to search for availability of a package. Here we look for `rsync`.

```
student@debian:~$ apt-cache search rsync | grep '^rsync'
rsync - fast, versatile, remote (and local) file-copying tool
rsyncrypto - rsync friendly encryption
```

3.3.5. apt-get install

You can install one or more applications by appending their name behind `apt-get install`. The following example shows how to install the `tftp-hpa` package (a TFTP server).

```
student@debian:~$ sudo apt-get install tftpd-hpa
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  pxelinux
The following NEW packages will be installed:
  tftpd-hpa
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 41.9 kB of archives.
After this operation, 117 kB of additional disk space will be used.
Get:1 http://httpredir.debian.org/debian bookworm/main amd64 tftpd-hpa amd64
 ↪  5.2+20150808-1.4 [41.9 kB]
Fetched 41.9 kB in 0s (241 kB/s)
Preconfiguring packages ...
Selecting previously unselected package tftpd-hpa.
(Reading database ... 29179 files and directories currently installed.)
Preparing to unpack .../tftpd-hpa_5.2+20150808-1.4_amd64.deb ...
Unpacking tftpd-hpa (5.2+20150808-1.4) ...
Setting up tftpd-hpa (5.2+20150808-1.4) ...
Processing triggers for man-db (2.11.2-2) ...
```

The `apt-get` command will ask the user to confirm the installation of the package by pressing "y" and ENTER. You can use the `-y` option to automatically answer yes to all questions.

The following example installs the `vim` package (VI iMproved, a powerful text editor for the terminal). **Remark** that some additional packages are installed as dependencies!

```
student@debian:~$ sudo apt-get install -y vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libgpm2 libsodium23 vim-runtime
Suggested packages:
  gpm ctags vim-doc vim-scripts
The following NEW packages will be installed:
  libgpm2 libsodium23 vim vim-runtime
0 upgraded, 4 newly installed, 0 to remove and 1 not upgraded.
Need to get 8,768 kB of archives.
After this operation, 41.5 MB of additional disk space will be used.
[...]
Setting up libsodium23:amd64 (1.0.18-1) ...
Setting up libgpm2:amd64 (1.20.7-10+b1) ...
Setting up vim-runtime (2:9.0.1378-2) ...
Setting up vim (2:9.0.1378-2) ...
[...]
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for libc-bin (2.36-9+deb12u4) ...
```

3.3.6. apt-get remove

You can remove one or more applications by appending their name behind `apt-get remove`.

```
student@debian:~$ sudo apt-get remove tftpd-hpa
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
  tftpd-hpa
0 upgraded, 0 newly installed, 1 to remove and 1 not upgraded.
After this operation, 117 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 29194 files and directories currently installed.)
Removing tftpd-hpa (5.2+20150808-1.4) ...
Processing triggers for man-db (2.11.2-2) ...
```

If we use `dpkg -l` to check the status of the `tftpd-hpa` package, we see that it is removed, some configuration (rc) files are left on the system. Indeed, the configuration file `/etc/init/tftpd-hpa.conf` is not removed! We'll solve this in the next section.

```
student@debian:~$ dpkg -l tftpd-hpa | tail -1
rc  tftpd-hpa      5.2+20150808-1.4 amd64        HPA's tftp server
student@debian:~$ ls -l /etc/init/tftpd-hpa.conf
-rw-r--r-- 1 root root 980 Oct 25  2022 /etc/init/tftpd-hpa.conf
```

The example below shows how to remove the `vim` package. Note that dependencies are **not** removed! You can execute `sudo apt autoremove` afterwards (as is suggested by the output of the command!) to remove those as well.

```
student@debian:~$ sudo apt-get remove vim
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer
 ↪  required:
  libsodium23 vim-runtime
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  vim
0 upgraded, 0 newly installed, 1 to remove and 1 not upgraded.
After this operation, 3,738 kB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 31257 files and directories currently installed.)
Removing vim (2:9.0.1378-2) ...
[...]
student@debian:~$ sudo apt-get autoremove
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
  libsodium23 vim-runtime
0 upgraded, 0 newly installed, 2 to remove and 1 not upgraded.
After this operation, 37.7 MB disk space will be freed.
Do you want to continue? [Y/n] y
(Reading database ... 31247 files and directories currently installed.)
Removing libsodium23:amd64 (1.0.18-1) ...
Removing vim-runtime (2:9.0.1378-2) ...
Removing 'diversion of /usr/share/vim/vim90/doc/help.txt to
 ↪ /usr/share/vim/vim90/doc/help.txt.vim-tiny by vim-runtime'
Removing 'diversion of /usr/share/vim/vim90/doc/tags to
 ↪ /usr/share/vim/vim90/doc/tags.vim-tiny by vim-runtime'
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for libc-bin (2.36-9+deb12u4) ...
```

3.3.7. apt-get purge

You can purge one or more applications by appending their name behind `apt-get purge`. Purging will also remove all existing configuration files related to that application. The screenshot shows how to purge the `tftpd-hpa` package.

```
student@debian:~$ ls -l /etc/init/tftpd-hpa.conf
-rw-r--r-- 1 root root 980 Oct 25  2022 /etc/init/tftpd-hpa.conf
student@debian:~$ sudo apt-get purge tftpd-hpa
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
  tftpd-hpa*
0 upgraded, 0 newly installed, 1 to remove and 1 not upgraded.
After this operation, 0 B of additional disk space will be used.
Do you want to continue? [Y/n] y
(Reading database ... 29182 files and directories currently installed.)
Purging configuration files for tftpd-hpa (5.2+20150808-1.4) ...
student@debian:~$ ls -l /etc/init/tftpd-hpa.conf
ls: cannot access '/etc/init/tftpd-hpa.conf': No such file or directory
```

Note that dpkg has no information about a purged package!

```
student@debian:~$ dpkg -l tftpd-hpa | tail -1 | tr -s ' '
dpkg-query: no packages found matching tftpd-hpa
```

3.4. apt

Nowadays, most people use `apt` for package management on Debian, Mint and Ubuntu systems. That does not mean that `apt-get` is no longer useful. In scripts, it is actually recommended to use `apt-get` because its options and behaviour are more stable and predictable than `apt`. For interactive use, `apt` is more user-friendly.

To synchronize with the repositories.

```
sudo apt update
```

To patch and upgrade all software to the latest version on Debian.

```
sudo apt upgrade
```

To patch and upgrade all software to the latest version on Ubuntu and Mint.

```
sudo apt safe-upgrade
```

To install an application with all dependencies.

```
sudo apt install $package
```

To search the repositories for applications that contain a certain string in their name or description.

```
apt search $string
```

To remove an application.

```
sudo apt remove $package
```

To remove an application and all configuration files.

```
sudo apt purge $package
```

3.5. /etc/apt/sources.list

Both `apt-get` and `apt` use the same configuration information in `/etc/apt/`. The main configuration file is `/etc/apt/sources.list` and the directory `/etc/apt/sources.list.d/` contains additional files. These contain a list of http or ftp sources where packages for the distribution can be downloaded. Third party software vendors may provide their own package repositories for Debian or Ubuntu. These repositories are typically added through a new file in `/etc/apt/sources.list.d/`.

This is what that list looks like on a Debian server system shortly after installation.

```
student@debian:~$ cat /etc/apt/sources.list
deb http://httpredir.debian.org/debian/ bookworm main non-free-firmware
deb-src http://httpredir.debian.org/debian/ bookworm main non-free-firmware

deb http://security.debian.org/debian-security bookworm-security main
 ↪ non-free-firmware
deb-src http://security.debian.org/debian-security bookworm-security main
 ↪ non-free-firmware

# bookworm-updates, to get updates before a point release is made;
deb http://httpredir.debian.org/debian/ bookworm-updates main
 ↪ non-free-firmware
deb-src http://httpredir.debian.org/debian/ bookworm-updates main
 ↪ non-free-firmware
```

If you use Linux as a daily driver, you may end up with a repository list with many more entries, like on this Ubuntu system:

```
student@ubuntu:~$ wc -l /etc/apt/sources.list
63 /etc/apt/sources.list
```

There is much more to learn about `apt`, explore commands like `add-apt-repository`, `apt-key` and `apropos apt`.

3.6. practice: package management on debian

1. On your Debian system, install updates for all available packages. Is there any difference with the Enterprise Linux way of doing this?
2. Do you always have to reboot your system after installing updates (like you usually have to do on Windows)? When do you have to reboot after updating?
3. Check whether the applications/commands `git`, `scp`, `gcc` and `zork` are installed.
4. Use the package manager to install them. Did you find them all? Did the installation include any dependencies?
5. Search the internet for `webmin` and figure out how to install it.

3.7. solutions: package management on debian

1. On your Debian system, install updates for all available packages. Is there any difference with the Enterprise Linux way of doing this?

   ```
   $ sudo apt update
   $ sudo apt upgrade
   ```

   You need two commands on Debian, only one on EL. `apt update` refreshes the list of available packages, `apt upgrade` installs the updates. On EL, `dnf update` and `dnf upgrade` are synonyms and do both.

2. Do you always have to reboot your system after installing updates (like you usually have to do on Windows)? When do you have to reboot after updating?

   On Linux, a reboot is usually only necessary when a new kernel update has been installed. This is because the kernel is loaded into memory at boot time and cannot be replaced while the system is running. Other updates can usually be applied without rebooting.

3. Check whether the applications/commands `git`, `scp`, `gcc` and `zork` are installed.

   You can check whether a package is installed with `dpkg -l`

   ```
   student@debian:~$ dpkg -l | grep git
   student@debian:~$ dpkg -l | grep scp
   student@debian:~$ dpkg -l | grep gcc
   student@debian:~$ dpkg -l | grep zork
   ```

   Remark that a package name is not necessarily the same as the name of the command. For example, the `scp` command may exist while no package called `scp` was found.

   ```
   student@ubuntu:~$ dpkg -l  |grep scp
   student@ubuntu:~$ which scp
   /usr/bin/scp
   ```

   So, let’s find the package that provides the `scp` command:

```
student@ubuntu:~$ dpkg -S /usr/bin/scp
openssh-client: /usr/bin/scp
```

4. Use the package manager to install them. Did you find them all? Did the installation include any dependencies?

```
$ sudo apt install git
$ sudo apt install openssh-client
$ sudo apt install gcc
```

   `gcc` is part of the Gnu Compiler Collection and may include many other packages installed as dependencies.

   `scp` should be installed as part of the `openssh-client` package.

   The `zork` package is not available in the default repositories. You can search for it on the internet and download it from a website, or you can install it from a package file if you have one.

5. Search the internet for `webmin` and figure out how to install it.

   A web search should point you to webmin.com. The download page helps you to download a repository file so you can install webmin with your package manager. The latest Webmin distribution is available in various package formats for download, a.o. .rpm, .deb, etc.

Part III.

Scripting 101

4. I/O redirection

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

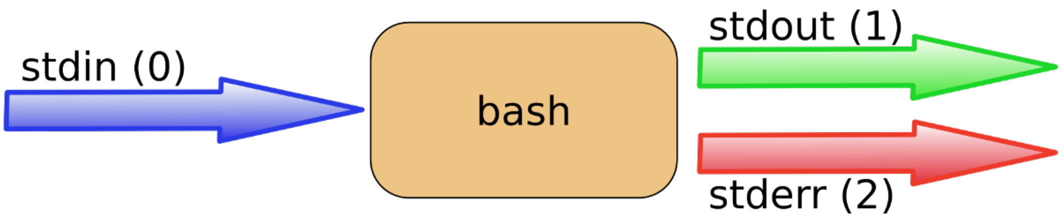
One of the powers of the Unix command line is the use of `input/output redirection` and `pipes`.

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

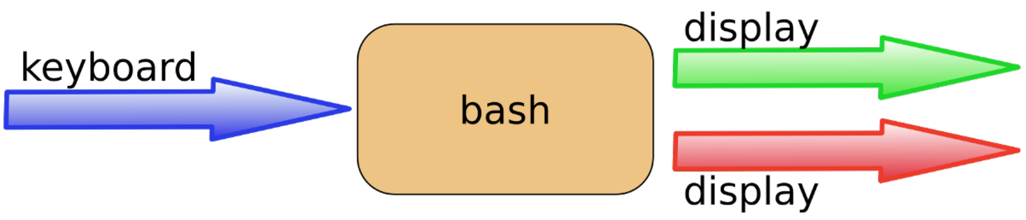
4.1. stdin, stdout, and stderr

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.

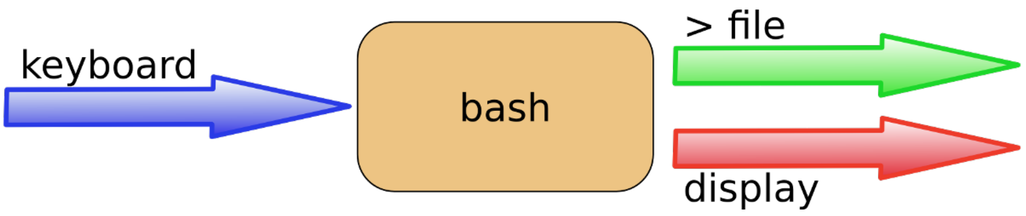


The next sections will explain how to redirect these streams.

4.2. output redirection

4.2.1. > 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.

4.2.2. 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 ~]$
```

4.2.3. 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 ~]$
```

4.2.4. 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 ~]$
```

4.2.5. » append

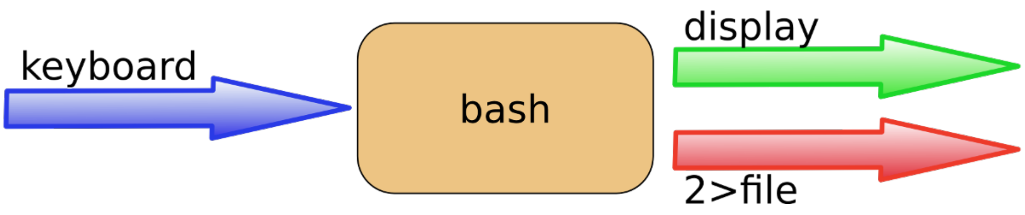
Use >> to append output to a file.

```
[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 ~]$
```

4.3. error redirection

4.3.1. 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 ~]$
```

4.3.2. 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.

4.4. 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
```

4.5. 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:~$
```

4.6. input redirection

4.6.1. < 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 ~]$
```

4.6.2. « 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 ~]$
```

4.6.3. «< 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`.

4.7. 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
```

4.8. 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
```

4.9. 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.

4.10. 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:~$
```

5. 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!

5.1. 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.

5.2. 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_
```

5.3. 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
```

5.4. shuf

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

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

5.5. 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
```

5.6. 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
```

5.7. 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
```

5.8. 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.

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

5.10. 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
```

5.11. 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
```

5.12. fmt

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

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

5.13. nl

Add line numbers to input text.

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

5.14. 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]$
```

5.15. 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@linux 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"
      }
   ]
}
```

5.16. 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
```

5.17. 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
```

5.18. 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
```

5.19. 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.

5.20. 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`.

5.21. pipe examples

5.21.1. 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
```

5.21.2. 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
```

5.21.3. 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
```

5.21.4. ip | awk

Display only IPv4 addresses of this computer.

```
student@linux:~/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@linux:~/pipes$ ip -br a | awk '{print $3}'
127.0.0.1/8
10.0.2.15/24
192.168.56.21/24
```

5.22. 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/`.

5.23. 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
```

6. shell variables

(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))

In this chapter we learn to manage shell **variables**, i.e. how to declare and initialize them, variable types, scope, and how to use them in scripts or in the terminal.

6.1. $ dollar sign

To retrieve the value of a **shell variable**, prefix the variable name with a dollar sign $. When you use the dollar sign syntax in a command, the shell will look for an existing variable named like the word following the $ and replace it with the value of that variable. This is called **parameter substitution**. When the variable does not exist, the shell will replace it with an empty string.

When you open a terminal, several variables are already set. 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 $HOSTNAME"
This is /bin/bash on computer linux.localdomain
[student@linux ~]$ echo "The userid of $USER is $UID"
The userid of studemt is 1000
[student@linux ~]$ echo "My homedir is $HOME"
My homedir is /home/student
```

The variable name can also be **enclosed in curly braces** `{}`, e.g. `${HOME}`, `${USER}`, etc. This is less ambiguous and in some situations even required to separate the variable name from text immediately following it. For 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
```

In a script, it's actually a best practice to *always* use the notation with the curly braces `${var}`.

6.2. case sensitive

This example shows that shell variables are case sensitive!

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

6.3. creating variables

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

```
[student@linux gen]$ my_var=555
[student@linux gen]$ echo "$my_var"
555
```

Remark!

- The variable name must start with a letter or an underscore and can contain letters, numbers, and underscores.
- There must be *no spaces* around the equal sign.
- Don't use the dollar sign when you assign a value to a variable. The dollar sign is exclusively used to retrieve the *value* of a variable.
- This is not set in stone, but it's recommended to use lowercase letters for most variable names and underscores to separate words, e.g. `${my_var}`, `${file_name}`, etc.
- Variables are usually *strings*, but can in some cases be interpreted as *integer numbers*.

6.4. quotes

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

```
[student@linux ~]$ my_var=555
[student@linux ~]$ echo ${my_var}
555
[student@linux ~]$ echo "${my_var}"
555
[student@linux ~]$ echo '${my_var}'
${my_var}
```

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.
```

In most cases, it is best practice to **always enclose variables in double quotes**. For example, if a variable contains a file name that has spaces in it, the shell may interpret this incorrectly:

```
student@linux:~/temp$ file_name='My file.txt'
student@linux:~/temp$ touch ${file_name}
student@linux:~/temp$ ls
file.txt  My
student@linux:~/temp$ ls -l
total 0
-rw-r--r-- 1 student student 0 Nov 21 19:28 file.txt
-rw-r--r-- 1 student student 0 Nov 21 19:28 My
```

What happened? The shell replaced `${file_name}` with the value of the variable resulting in:

```
touch My file.txt
```

Because there were no quotes in the command, word splitting occured and My and file.txt were interpreted as two separate arguments. This can be avoided by using double quotes:

```
student@linux:~/temp$ touch "${file_name}"
student@linux:~/temp$ ls -l
total 0
-rw-r--r-- 1 student student 0 Nov 21 19:28  file.txt
-rw-r--r-- 1 student student 0 Nov 21 19:28  My
-rw-r--r-- 1 student student 0 Nov 21 19:28 'My file.txt'
```

Your scripts will become much more robust if you always use double quotes around variables and will be able to handle file names with spaces or other special characters.

6.5. set

You can use the set command without options to display a list of shell variables. On Ubuntu and Debian systems, the set command will also list shell functions after the shell variables.

The set command can also be used to change the default behaviour of the shell. For example, set -o nounset (or set -u) at the start of a script will cause an error when an unbound variable is used, instead of replacing it with an empty string. Also, the script will exit immediately with a nonzero exit status, instead of continuing. Likewise, set -o errexit (or set -e) will cause the script to exit immediately when a command fails and set -o pipefail (no short option available) will print more informative error messages when a command in a pipeline fails. It is best practice to start each script with these three options!

```
set -o nounset
set -o errexit
set -o pipefail
```

These can be written in short form as:

```
set -euo pipefail
```

With set -x (or set -o xtrace), the shell will print each command before executing it, but after all forms of substitution have been applied. This is useful for debugging scripts, as it shows exactly how the shell has interpreted the command you executed.

```
student@linux:~$ set -x
student@linux:~$ echo "Hello ${USER}"
+ echo Hello student
Hello student
student@linux:~$ set +x
```

Turn the option off with set +x.

For other uses of the set command, see the builtins(7) man page.

6.6. unset

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

```
[student@linux ~]$ my_var=8472
[student@linux ~]$ echo "${my_var}"
8472
[student@linux ~]$ unset my_var
[student@linux ~]$ echo "${my_var}"

[student@linux ~]$
```

6.7. $PS1

The $PS1 variable determines your shell prompt. You can use backslash escaped special characters like \u for the username or \w for the working directory. The bash(1) manual has a complete reference (in the section PROMPTING).

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

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

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

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

6.8. $PATH

The $PATH variable is determines where the shell is looking for commands to execute (unless the command is builtin or aliased). This variable contains a list of directories, separated by colons.

```
[student@linux ~]$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

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 (which is a shortcut to *the current working directory*). This is not recommended, though!

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

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 (for commands relating to **s**ystem administration tasks).

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

```
Password:
[root@linux ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
```

6.9. variable scope

When you initialize a variable in some shell, it will not be available in child shells (also called *subshells*):

```
[student@linux ~]$ penguin=tux
[student@linux ~]$ echo $penguin
tux
[student@linux ~]$ bash          #<---- this is where we create a subshell
[student@linux ~]$ echo $penguin

[student@linux ~]$ exit
[student@linux ~]$ echo $penguin
tux
```

We can use the `export` command to turn the variable into an **environment variable**. This will make the variable available in child shells.

```
[student@linux ~]$ export penguin=tux
[student@linux ~]$ echo $penguin
tux
[student@linux ~]$ bash
[student@linux ~]$ echo $penguin
tux
[student@linux ~]$ exit
```

Remark that is actually a convention to give **environment variables uppercase names** and “normal” (non-exported) variables lowercase names.

When you execute a script, it will also run in a subshell, so only `export`ed variables will be available in the script.

If you try this example, first `unset` the variable `$penguin`.

```
[student@linux ~]$ unset penguin
[student@linux ~]$ penguin=pingu
[student@linux ~]$ echo 'echo "${penguin}"' > script.sh
[student@linux ~]$ bash script.sh

[student@linux ~]$ export penguin
[student@linux ~]$ bash script.sh
pingu
```

Beware that exported variables cannot be seen in the parent shell! Assume the variable `$penguin` is unset:

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

[student@linux ~]$ bash
[student@linux ~]$ export penguin=pingu
[student@linux ~]$ echo $penguin
pingu
[student@linux ~]$ exit
[student@linux ~]$ echo $penguin

[student@linux ~]$
```

6.10. env

The `env` command without options will display a list of **environment variables**, i.e. all variables that have been made available to subshells with `export`.

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 $USER'
/bin/bash /home/student student
[student@linux ~]$ 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$ touch Filea Fileb FileA FileB
student@linux:~/test$ env LANG=C bash -c 'ls File[[:alpha:]]'
FileA  FileB  Filea  Fileb
student@linux:~/test$ env LANG=en_US.UTF-8 bash -c 'ls File[[:alpha:]]'
Filea  FileA  Fileb  FileB
```

6.11. practice: shell variables

1. Use `echo` to display `Hello` followed by your username (use an existing shell variable!)
2. Create a variable `answer` with a value of `42`.
3. Copy the value of `$LANG` to `$my_lang`.
4. List all current shell variables.
5. List all exported shell variables (i.e. environment variables).
6. Do the `env` and `set` commands display your variable ?
7. Destroy your `answer` variable.
8. Create two variables, and `export` one of them.
9. Display these variables in an interactive child shell and check which one is still available.
10. Create a variable, give it the value 'Dumb', create another variable with value 'do'. Use `echo` and the two variables to print 'Dumbledore'.
11. Find the list of backslash escaped characters in the manual of bash. Add the time to your `PS1` prompt.

6.12. solution: shell variables

1. Use `echo` to display `Hello` followed by your username (use an existing shell variable!)

```
echo "Hello ${USER}"
```

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

```
answer=42
```

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

```
my_lang="${LANG}"
```

4. List all current shell variables.

```
set
```

5. List all exported shell variables (i.e. environment variables).

```
env
export
declare -x
```

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

```
student@linux:~$ set | grep my_lang
my_lang=en_US.UTF-8
student@linux:~$ set | grep answer
answer=42
student@linux:~$ env | grep my_lang
student@linux:~$ env | grep answer
```

   The `env` command does not display the `my_lang` and `answer` variables, because they are not exported.

7. Destroy your `answer` variable.

```
unset answer
```

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

```
var1=one
export var2=two
```

9. Display these variables in an interactive child shell and check which one is still available.

```
student@linux:~$ bash
bash
student@linux:~$ echo "${var1} ${var2}"
 two
student@linux:~$ exit
```

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

```
var_x=Dumb
var_y=do

echo "${var_x}le${var_y}re"
```

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

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

```
student@linux:~$ PS1='\t \u@\h \W$ '
21:52:48 student@linux ~$ pwd
/home/student
21:52:52 student@linux ~$
```

7. 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.

7.1. 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.

7.2. 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:~$
```

7.3. 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.

7.4. 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
```

7.5. 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.

7.6. 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
```

7.7. 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:~$
```

7.8. 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!

7.9. 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
```

7.10. 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'
```

7.11. 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`.

7.12. 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.

7.12.1. 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.

7.12.2. 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!

7.12.3. 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
```

7.12.4. 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.

7.13. 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 --.

7.14. 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?

7.15. 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!

Part IV.

Organising users

8. 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*.

8.1. file ownership

8.1.1. 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.

8.1.2. 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#
```

8.1.3. 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
```

8.2. 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
```

8.3. permissions

8.3.1. 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`)

8.3.2. 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.

8.3.3. 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.

8.3.4. 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!)

8.3.5. 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
```

8.3.6. 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
```

8.3.7. 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/
```

8.3.8. 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
```

8.4. 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.

8.5. 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 `rwxrwxr--` 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/
```

9. advanced file permissions

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

9.1. 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
```

9.2. 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
...
```

9.3. 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.

9.4. 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 $(which sudo)
---s--x--x. 1 root root 123832 Oct  7  2013 /usr/bin/sudo
student@linux:~$
```

9.5. 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.

9.6. 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#
```

10. 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`.

10.1. whoami

The `whoami` command tells you your username.

```
student@debian:~$ whoami
student
```

10.2. 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.

10.3. 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)
```

10.4. 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
```

10.5. 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
```

10.6. 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).

10.7. 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
```

10.8. 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).

10.9. 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.

10.10. 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.

10.11. 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:~#
```

10.12. 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.

10.13. 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.

10.14. 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.

11. 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.

11.1. 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.

11.2. /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.

11.3. /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$k/zvYGDp1caN0p50aa9QI.$svec3ZhaLcxykbHKh6SMb4VnhYiJKgdN2ZhDG
 ↪  BLApa6:20007:0:99999:7:::
```

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.

11.4. 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
```

11.5. 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/xnykHwGBVIzB⌋
  ↪  GLRg6:20011:0:99999:7:::
```

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
```

11.5.1. /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
```

11.6. 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.

11.6.1. 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
```

11.6.2. 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
```

11.6.3. 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
```

11.7. 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
```

11.8. 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
```

11.9. 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/
```

11.9.1. /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
```

11.9.2. 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
```

11.10. 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/
```

11.11. 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.

11.12. 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
```

12. 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.

12.1. 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
```

12.2. 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.

12.3. 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 ~]#
```

12.4. 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!

12.5. 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.
```

12.6. /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:~#
```

12.7. 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:~#
```

12.8. 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.

12.9. 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)
```

12.10. 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`.

12.11. 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
```

13. 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.

13.1. 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.

13.2. ~/.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 ~]#
```

13.3. ~/.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.

13.4. ~/.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.

13.5. ~/.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
```

13.6. ~/.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
```

13.7. Debian overview

Below is a table overview of when Debian is running any of these bash startup scripts.

Table 13.1.: 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

13.8. RHEL5 overview

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

Table 13.2.: 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

13.9. 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 ?

13.10. 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)
```

14. 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`.

14.1. 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
```

14.2. 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.

14.3. 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 ~]$
```

14.4. 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.

14.5. 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
```

14.6. groupdel

You can permanently remove a group with the groupdel command.

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

14.7. 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
```

14.8. 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#
```

14.9. 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.

14.10. 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.

14.11. 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 V.

Webserver; scripting 102

15. apache web server

(Written by Paul Cobbaut, https://github.com/paulcobbaut/, with contributions by: Hans Roes, https://github.com/Blokker-1999/, Alex M. Schapelle, https://github.com/zero-pytagoras/)

In this chapter we learn how to setup a web server with the `apache` software.

According to NetCraft (http://news.netcraft.com/archives/web_server_survey.html) about seventy percent of all web servers are running on Apache. The name is derived from a `patchy` web server, because of all the patches people wrote for the NCSA httpd server.

Later chapters will expand this web server into a LAMP stack (Linux, Apache, Mysql, Perl/PHP/Python).

15.1. introduction to apache

15.1.1. installing on Debian

This screenshot shows that there is no `apache` server installed, nor does the `/var/www` directory exist.

```
root@linux:~# ls -l /var/www
ls: cannot access /var/www: No such file or directory
root@linux:~# dpkg -l | grep apache
```

To install `apache` on Debian:

```
root@linux:~# aptitude install apache2
The following NEW packages will be installed:
  apache2 apache2-mpm-worker{a} apache2-utils{a} apache2.2-bin{a} apache2.2-com\
mon{a}  libapr1{a}  libaprutil1{a}  libaprutil1-dbd-sqlite3{a}  libaprutil1-ldap{a}\
 ssl-cert{a}
0 packages upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,487 kB of archives. After unpacking 5,673 kB will be used.
Do you want to continue? [Y/n/?]
```

After installation, the same two commands as above will yield a different result:

```
root@linux:~# ls -l /var/www
total 4
-rw-r--r-- 1 root root 177 Apr 29 11:55 index.html
root@linux:~# dpkg -l | grep apache | tr -s ' '
ii apache2 2.2.22-13+deb7u1 amd64 Apache HTTP Server metapackage
ii apache2-mpm-worker 2.2.22-13+deb7u1 amd64 Apache HTTP Server - high speed th\
readed model
ii apache2-utils 2.2.22-13+deb7u1 amd64 utility programs for webservers
ii apache2.2-bin 2.2.22-13+deb7u1 amd64 Apache HTTP Server common binary files
ii apache2.2-common 2.2.22-13+deb7u1 amd64 Apache HTTP Server common files
```

15.1.2. installing on RHEL/CentOS

Note that Red Hat derived distributions use `httpd` as package and process name instead of `apache`.

To verify whether `apache` is installed in CentOS/RHEL:

```
[root@linux ~]# rpm -q httpd
package httpd is not installed
[root@linux ~]# ls -l /var/www
ls: cannot access /var/www: No such file or directory
```

To install apache on CentOS:

```
[root@linux ~]# yum install httpd
```

After running the `yum install httpd` command, the Centos 6.5 server has apache installed and the `/var/www` directory exists.

```
[root@linux ~]# rpm -q httpd
httpd-2.2.15-30.el6.centos.x86_64
[root@linux ~]# ls -l /var/www
total 16
drwxr-xr-x. 2 root root 4096 Apr  3 23:57 cgi-bin
drwxr-xr-x. 3 root root 4096 May  6 13:08 error
drwxr-xr-x. 2 root root 4096 Apr  3 23:57 html
drwxr-xr-x. 3 root root 4096 May  6 13:08 icons
[root@linux ~]#
```

15.1.3. running apache on Debian

This is how you start `apache2` on Debian.

```
root@linux:~# service apache2 status
Apache2 is NOT running.
root@linux:~# service apache2 start
Starting web server: apache2apache2: Could not reliably determine the server's \
fully qualified domain name, using 127.0.1.1 for ServerName
.
```

To verify, run the `service apache2 status` command again or use `ps`.

```
root@linux:~# service apache2 status
Apache2 is running (pid 3680).
root@linux:~# ps -C apache2
  PID TTY          TIME CMD
 3680 ?        00:00:00 apache2
 3683 ?        00:00:00 apache2
 3684 ?        00:00:00 apache2
 3685 ?        00:00:00 apache2
root@linux:~#
```

Or use `wget` and `file` to verify that your web server serves an html document.

```
root@linux:~# wget 127.0.0.1
--2014-05-06 13:27:02--  http://127.0.0.1/
Connecting to 127.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[==================================================>]  177         --.-
K/s   in 0s

2014-05-06 13:27:02 (15.8 MB/s) - `index.html' saved [177/177]

root@linux:~# file index.html
index.html: HTML document, ASCII text
root@linux:~#
```

Or verify that apache is running by opening a web browser, and browse to the ip-address of your server. An Apache test page should be shown.

You can do the following to quickly avoid the 'could not reliably determine the fqdn' message when restarting apache.

```
root@linux:~# echo ServerName debian10 >> /etc/apache2/apache2.conf
root@linux:~# service apache2 restart
Restarting web server: apache2 ... waiting .
root@linux:~#
```

15.1.4. running apache on CentOS

Starting the `httpd` on RHEL/CentOS is done with the `service` command.

```
[root@linux ~]# service httpd status
httpd is stopped
[root@linux ~]# service httpd start
Starting httpd: httpd: Could not reliably determine the server's fully qualifie\
d domain name, using 127.0.0.1 for ServerName
                                                           [  OK  ]
[root@linux ~]#
```

To verify that `apache` is running, use `ps` or issue the `service httpd status` command again.

```
[root@linux ~]# service httpd status
httpd (pid  2410) is running...
[root@linux ~]# ps -C httpd
  PID TTY          TIME CMD
 2410 ?        00:00:00 httpd
 2412 ?        00:00:00 httpd
 2413 ?        00:00:00 httpd
 2414 ?        00:00:00 httpd
 2415 ?        00:00:00 httpd
 2416 ?        00:00:00 httpd
 2417 ?        00:00:00 httpd
 2418 ?        00:00:00 httpd
 2419 ?        00:00:00 httpd
[root@linux ~]#
```

To prevent the 'Could not reliably determine the fqdn' message, issue the following command.

```
[root@linux ~]# echo ServerName Centos65 >> /etc/httpd/conf/httpd.conf
[root@linux ~]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
[root@linux ~]#
```

15.1.5. index file on CentOS

CentOS does not provide a standard index.html or index.php file. A simple `wget` gives an error.

```
[root@linux ~]# wget 127.0.0.1
--2014-05-06 15:10:22--  http://127.0.0.1/
Connecting to 127.0.0.1:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2014-05-06 15:10:22 ERROR 403: Forbidden.
```

Instead when visiting the ip-address of your server in a web browser you get a `noindex.html` page. You can verify this using `wget`.

```
[root@linux ~]# wget http://127.0.0.1/error/noindex.html
--2014-05-06 15:16:05--  http://127.0.0.1/error/noindex.html
Connecting to 127.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5039 (4.9K) [text/html]
Saving to: “noindex.html”

100%[=============================================>] 5,039     --.-K/s   in 0s

2014-05-06 15:16:05 (289 MB/s) - “noindex.html” saved [5039/5039]

[root@linux ~]# file noindex.html
noindex.html: HTML document text
[root@linux ~]#
```

Any custom `index.html` file in `/var/www/html` will immediately serve as an index for this web server.

```
[root@linux ~]# echo 'Welcome to my website' > /var/www/html/index.html
[root@linux ~]# wget http://127.0.0.1
--2014-05-06 15:19:16--  http://127.0.0.1/
Connecting to 127.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 22 [text/html]
Saving to: “index.html”

100%[=============================================>] 22        --.-K/s   in 0s

2014-05-06 15:19:16 (1.95 MB/s) - “index.html” saved [22/22]

[root@linux ~]# cat index.html
Welcome to my website
```

15.1.6. default website

Changing the default website of a freshly installed apache web server is easy. All you need to do is create (or change) an index.html file in the DocumentRoot directory.

To locate the DocumentRoot directory on Debian:

```
root@linux:~# grep DocumentRoot /etc/apache2/sites-available/default
        DocumentRoot /var/www
```

This means that `/var/www/index.html` is the default web site.

```
root@linux:~# cat /var/www/index.html
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
root@linux:~#
```

This screenshot shows how to locate the `DocumentRoot` directory on RHEL/CentOS.

```
[root@linux ~]# grep ^DocumentRoot /etc/httpd/conf/httpd.conf
DocumentRoot "/var/www/html"
```

RHEL/CentOS have no default web page (only the noindex.html error page mentioned before). But an `index.html` file created in `/var/www/html/` will automatically be used as default page.

```
[root@linux ~]# echo '<html><head><title>Default website</title></head><body\
><p>A new web page</p></body></html>' > /var/www/html/index.html
[root@linux ~]# cat /var/www/html/index.html
<html><head><title>Default website</title></head><body><p>A new web page</p></b\
ody></html>
[root@linux ~]#
```

15.1.7. apache configuration

There are many similarities, but also a couple of differences when configuring `apache` on Debian or on CentOS. Both Linux families will get their own chapters with examples.

All configuration on RHEL/CentOS is done in `/etc/httpd`.

```
[root@linux ~]# ls -l /etc/httpd/
total 8
drwxr-xr-x. 2 root root 4096 May  6 13:08 conf
drwxr-xr-x. 2 root root 4096 May  6 13:08 conf.d
lrwxrwxrwx. 1 root root   19 May  6 13:08 logs -> ../../var/log/httpd
lrwxrwxrwx. 1 root root   29 May  6 13:08 modules -> ../../usr/lib64/httpd/modu\
les
lrwxrwxrwx. 1 root root   19 May  6 13:08 run -> ../../var/run/httpd
[root@linux ~]#
```

Debian (and ubuntu/mint/...) use `/etc/apache2`.

```
root@linux:~# ls -l /etc/apache2/
total 72
-rw-r--r-- 1 root root  9659 May  6 14:23 apache2.conf
drwxr-xr-x 2 root root  4096 May  6 13:19 conf.d
-rw-r--r-- 1 root root  1465 Jan 31 18:35 envvars
-rw-r--r-- 1 root root 31063 Jul 20  2013 magic
drwxr-xr-x 2 root root  4096 May  6 13:19 mods-available
drwxr-xr-x 2 root root  4096 May  6 13:19 mods-enabled
-rw-r--r-- 1 root root   750 Jan 26 12:13 ports.conf
drwxr-xr-x 2 root root  4096 May  6 13:19 sites-available
drwxr-xr-x 2 root root  4096 May  6 13:19 sites-enabled
root@linux:~#
```

15.2. port virtual hosts on Debian

15.2.1. default virtual host

Debian has a virtualhost configuration file for its default website in `/etc/apache2/sites-available/default`.

```
root@linux:~# head -2 /etc/apache2/sites-available/default
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
```

15.2.2. three extra virtual hosts

In this scenario we create three additional websites for three customers that share a clubhouse and want to jointly hire you. They are a model train club named `Choo Choo`, a chess club named `Chess Club 42` and a hackerspace named `hunter2`.

One way to put three websites on one web server, is to put each website on a different port. This screenshot shows three newly created `virtual hosts`, one for each customer.

```
root@linux:~# vi /etc/apache2/sites-available/choochoo
root@linux:~# cat /etc/apache2/sites-available/choochoo
<VirtualHost *:7000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/choochoo
</VirtualHost>
root@linux:~# vi /etc/apache2/sites-available/chessclub42
root@linux:~# cat /etc/apache2/sites-available/chessclub42
<VirtualHost *:8000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/chessclub42
</VirtualHost>
root@linux:~# vi /etc/apache2/sites-available/hunter2
root@linux:~# cat /etc/apache2/sites-available/hunter2
<VirtualHost *:9000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/hunter2
</VirtualHost>
```

Notice the different port numbers 7000, 8000 and 9000. Notice also that we specified a unique `DocumentRoot` for each website.

Are you using `Ubuntu` or `Mint`, then these configfiles need to end in `.conf`.

15.2.3. three extra ports

We need to enable these three ports on apache in the `ports.conf` file. Open this file with `vi` and add three lines to `listen` on three extra ports.

```
root@linux:~# vi /etc/apache2/ports.conf
```

Verify with `grep` that the `Listen` directives are added correctly.

```
root@linux:~# grep ^Listen /etc/apache2/ports.conf
Listen 80
Listen 7000
Listen 8000
Listen 9000
```

15.2.4. three extra websites

Next we need to create three `DocumentRoot` directories.

```
root@linux:~# mkdir /var/www/choochoo
root@linux:~# mkdir /var/www/chessclub42
root@linux:~# mkdir /var/www/hunter2
```

And we have to put some really simple website in those directories.

```
root@linux:~# echo 'Choo Choo model train Choo Choo' > /var/www/choochoo/inde\
x.html
root@linux:~# echo 'Welcome to chess club 42' > /var/www/chessclub42/index.ht\
ml
root@linux:~# echo 'HaCkInG iS fUn At HuNtEr2' > /var/www/hunter2/index.html
```

15.2.5. enabling extra websites

The last step is to enable the websites with the `a2ensite` command. This command will create links in `sites-enabled`.

The links are not there yet...

```
root@linux:~# cd /etc/apache2/
root@linux:/etc/apache2# ls sites-available/
chessclub42  choochoo  default  default-ssl  hunter2
root@linux:/etc/apache2# ls sites-enabled/
000-default
```

So we run the `a2ensite` command for all websites.

```
root@linux:/etc/apache2# a2ensite choochoo
Enabling site choochoo.
To activate the new configuration, you need to run:
  service apache2 reload
root@linux:/etc/apache2# a2ensite chessclub42
Enabling site chessclub42.
To activate the new configuration, you need to run:
  service apache2 reload
```

```
root@linux:/etc/apache2# a2ensite hunter2
Enabling site hunter2.
To activate the new configuration, you need to run:
  service apache2 reload
```

The links are created, so we can tell apache.

```
root@linux:/etc/apache2# ls sites-enabled/
000-default  chessclub42  choochoo  hunter2
root@linux:/etc/apache2# service apache2 reload
Reloading web server config: apache2.
root@linux:/etc/apache2#
```

15.2.6. testing the three websites

Testing the model train club named Choo Choo on port 7000.

```
root@linux:/etc/apache2# wget 127.0.0.1:7000
--2014-05-06 21:16:03--  http://127.0.0.1:7000/
Connecting to 127.0.0.1:7000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32 [text/html]
Saving to: `index.html'

100%[============================================>] 32        --.-K/s   in 0s

2014-05-06 21:16:03 (2.92 MB/s) - `index.html' saved [32/32]

root@linux:/etc/apache2# cat index.html
Choo Choo model train Choo Choo
```

Testing the chess club named Chess Club 42 on port 8000.

```
root@linux:/etc/apache2# wget 127.0.0.1:8000
--2014-05-06 21:16:20--  http://127.0.0.1:8000/
Connecting to 127.0.0.1:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 25 [text/html]
Saving to: `index.html.1'

100%[===========================================>] 25         --.-K/s   in 0s

2014-05-06 21:16:20 (2.16 MB/s) - `index.html.1' saved [25/25]

root@linux:/etc/apache2# cat index.html.1
Welcome to chess club 42
```

Testing the hacker club named hunter2 on port 9000.

```
root@linux:/etc/apache2# wget 127.0.0.1:9000
--2014-05-06 21:16:30--  http://127.0.0.1:9000/
Connecting to 127.0.0.1:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26 [text/html]
Saving to: `index.html.2'
```

```
100%[=========================================>] 26          --.-K/s   in 0s

2014-05-06 21:16:30 (2.01 MB/s) - `index.html.2' saved [26/26]

root@linux:/etc/apache2# cat index.html.2
HaCkInG iS fUn At HuNtEr2
```

Cleaning up the temporary files.

```
root@linux:/etc/apache2# rm index.html index.html.1 index.html.2
```

Try testing from another computer using the ip-address of your server.

15.3. named virtual hosts on Debian

15.3.1. named virtual hosts

The chess club and the model train club find the port numbers too hard to remember. They would prefere to have their website accessible by name.

We continue work on the same server that has three websites on three ports. We need to make sure those websites are accesible using the names `choochoo.local`, `chessclub42.local` and `hunter2.local`.

We start by creating three new virtualhosts.

```
root@linux:/etc/apache2/sites-available# vi choochoo.local
root@linux:/etc/apache2/sites-available# vi chessclub42.local
root@linux:/etc/apache2/sites-available# vi hunter2.local
root@linux:/etc/apache2/sites-available# cat choochoo.local
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName choochoo.local
        DocumentRoot /var/www/choochoo
</VirtualHost>
root@linux:/etc/apache2/sites-available# cat chessclub42.local
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName chessclub42.local
        DocumentRoot /var/www/chessclub42
</VirtualHost>
root@linux:/etc/apache2/sites-available# cat hunter2.local
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName hunter2.local
        DocumentRoot /var/www/hunter2
</VirtualHost>
root@linux:/etc/apache2/sites-available#
```

Notice that they all listen on `port 80` and have an extra `ServerName` directive.

15.3.2. name resolution

We need some way to resolve names. This can be done with DNS, which is discussed in another chapter. For this demo it is also possible to quickly add the three names to the /etc/hosts file.

```
root@linux:/etc/apache2/sites-available# grep ^192 /etc/hosts
192.168.42.50 choochoo.local
192.168.42.50 chessclub42.local
192.168.42.50 hunter2.local
```

Note that you may have another ip address...

15.3.3. enabling virtual hosts

Next we enable them with a2ensite.

```
root@linux:/etc/apache2/sites-available# a2ensite choochoo.local
Enabling site choochoo.local.
To activate the new configuration, you need to run:
  service apache2 reload
root@linux:/etc/apache2/sites-available# a2ensite chessclub42.local
Enabling site chessclub42.local.
To activate the new configuration, you need to run:
  service apache2 reload
root@linux:/etc/apache2/sites-available# a2ensite hunter2.local
Enabling site hunter2.local.
To activate the new configuration, you need to run:
  service apache2 reload
```

15.3.4. reload and verify

After a service apache2 reload the websites should be available by name.

```
root@linux:/etc/apache2/sites-available# service apache2 reload
Reloading web server config: apache2.
root@linux:/etc/apache2/sites-available# wget chessclub42.local
--2014-05-06 21:37:13--  http://chessclub42.local/
Resolving chessclub42.local (chessclub42.local)... 192.168.42.50
Connecting to chessclub42.local (chessclub42.local)|192.168.42.50|:80... conne\
cted.
HTTP request sent, awaiting response... 200 OK
Length: 25 [text/html]
Saving to: `index.html'

100%[=============================================>] 25       --.-K/s   in 0s

2014-05-06 21:37:13 (2.06 MB/s) - `index.html' saved [25/25]

root@linux:/etc/apache2/sites-available# cat index.html
Welcome to chess club 42
```

15.4. password protected website on Debian

You can secure files and directories in your website with a `.htaccess` file that refers to a `.htpasswd` file. The `htpasswd` command can create a `.htpasswd` file that contains a userid and an (encrypted) password.

This screenshot creates a user and password for the hacker named `cliff` and uses the `-c` flag to create the `.htpasswd` file.

```
root@linux:~# htpasswd -c /var/www/.htpasswd cliff
New password:
Re-type new password:
Adding password for user cliff
root@linux:~# cat /var/www/.htpasswd
cliff:$apr1$vujll0KL$./SZ4w9q0swhX93pQ0PVp.
```

Hacker `rob` also wants access, this screenshot shows how to add a second user and password to `.htpasswd`.

```
root@linux:~# htpasswd /var/www/.htpasswd rob
New password:
Re-type new password:
Adding password for user rob
root@linux:~# cat /var/www/.htpasswd
cliff:$apr1$vujll0KL$./SZ4w9q0swhX93pQ0PVp.
rob:$apr1$HNln1FFt$nRlpF0H.IW11/1DRq4lQo0
```

Both Cliff and Rob chose the same password (hunter2), but that is not visible in the `.htpasswd` file because of the different salts.

Next we need to create a `.htaccess` file in the `DocumentRoot` of the website we want to protect. This screenshot shows an example.

```
root@linux:~# cd /var/www/hunter2/
root@linux:/var/www/hunter2# cat .htaccess
AuthUserFile /var/www/.htpasswd
AuthName "Members only!"
AuthType Basic
require valid-user
```

Note that we are protecting the website on `port 9000` that we created earlier.

And because we put the website for the Hackerspace named hunter2 in a subdirectory of the default website, we will need to adjust the `AllowOvveride` parameter in `/etc/apache2/sites-available/default` as this screenshot shows (with line numbers on debian10, your may vary).

```
9         <Directory /var/www/>
10                Options Indexes FollowSymLinks MultiViews
11                AllowOverride Authconfig
12                Order allow,deny
13                allow from all
14         </Directory
```

Now restart the apache2 server and test that it works!

15.5. port virtual hosts on CentOS

15.5.1. default virtual host

Unlike Debian, CentOS has no virtualHost configuration file for its default website. Instead the default configuration will throw a standard error page when no index file can be found in the default location (/var/www/html).

15.5.2. three extra virtual hosts

In this scenario we create three additional websites for three customers that share a clubhouse and want to jointly hire you. They are a model train club named `Choo Choo`, a chess club named `Chess Club 42` and a hackerspace named `hunter2`.

One way to put three websites on one web server, is to put each website on a different port. This screenshot shows three newly created `virtual hosts`, one for each customer.

```
[root@CentOS65 ~]# vi /etc/httpd/conf.d/choochoo.conf
[root@CentOS65 ~]# cat /etc/httpd/conf.d/choochoo.conf
<VirtualHost *:7000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html/choochoo
</VirtualHost>
[root@CentOS65 ~]# vi /etc/httpd/conf.d/chessclub42.conf
[root@CentOS65 ~]# cat /etc/httpd/conf.d/chessclub42.conf
<VirtualHost *:8000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html/chessclub42
</VirtualHost>
[root@CentOS65 ~]# vi /etc/httpd/conf.d/hunter2.conf
[root@CentOS65 ~]# cat /etc/httpd/conf.d/hunter2.conf
<VirtualHost *:9000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html/hunter2
</VirtualHost>
```

Notice the different port numbers 7000, 8000 and 9000. Notice also that we specified a unique `DocumentRoot` for each website.

15.5.3. three extra ports

We need to enable these three ports on apache in the `httpd.conf` file.

```
[root@CentOS65 ~]# vi /etc/httpd/conf/httpd.conf
root@linux:~# grep ^Listen /etc/httpd/conf/httpd.conf
Listen 80
Listen 7000
Listen 8000
Listen 9000
```

15.5.4. SELinux guards our ports

If we try to restart our server, we will notice the following error:

```
[root@CentOS65 ~]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:
     (13)Permission denied: make_sock: could not bind to address 0.0.0.0:7000
no listening sockets available, shutting down
                                                           [FAILED]
```

This is due to SELinux reserving ports 7000 and 8000 for other uses. We need to tell SELinux we want to use these ports for http traffic

```
[root@CentOS65 ~]# semanage port -m -t http_port_t -p tcp 7000
[root@CentOS65 ~]# semanage port -m -t http_port_t -p tcp 8000
[root@CentOS65 ~]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
```

15.5.5. three extra websites

Next we need to create three `DocumentRoot` directories.

```
[root@CentOS65 ~]# mkdir /var/www/html/choochoo
[root@CentOS65 ~]# mkdir /var/www/html/chessclub42
[root@CentOS65 ~]# mkdir /var/www/html/hunter2
```

And we have to put some really simple website in those directories.

```
[root@CentOS65 ~]# echo 'Choo Choo model train Choo Choo' > /var/www/html/chooc\
hoo/index.html
[root@CentOS65 ~]# echo 'Welcome to chess club 42' > /var/www/html/chessclub42/\
index.html
[root@CentOS65 ~]# echo 'HaCkInG iS fUn At HuNtEr2' > /var/www/html/hunter2/ind\
ex.html
```

15.5.6. enabling extra websites

The only way to enable or disable configurations in RHEL/CentOS is by renaming or moving the configuration files. Any file in /etc/httpd/conf.d ending on .conf will be loaded by Apache. To disable a site we can either rename the file or move it to another directory.

The files are created, so we can tell `apache`.

```
[root@CentOS65 ~]# ls /etc/httpd/conf.d/
chessclub42.conf  choochoo.conf  hunter2.conf  README  welcome.conf
[root@CentOS65 ~]# service httpd reload
Reloading httpd:
```

15.5.7. testing the three websites

Testing the model train club named Choo Choo on port 7000.

```
[root@CentOS65 ~]# wget 127.0.0.1:7000
--2014-05-11 11:59:36--  http://127.0.0.1:7000/
Connecting to 127.0.0.1:7000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32 [text/html]
Saving to: `index.html'

100%[===========================================>] 32         --.-K/s   in 0s

2014-05-11 11:59:36 (4.47 MB/s) - `index.html' saved [32/32]

[root@CentOS65 ~]# cat index.html
Choo Choo model train Choo Choo
```

Testing the chess club named Chess Club 42 on port 8000.

```
[root@CentOS65 ~]# wget 127.0.0.1:8000
--2014-05-11 12:01:30--  http://127.0.0.1:8000/
Connecting to 127.0.0.1:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 25 [text/html]
Saving to: `index.html.1'

100%[===========================================>] 25         --.-K/s   in 0s

2014-05-11 12:01:30 (4.25 MB/s) - `index.html.1' saved [25/25]

root@linux:/etc/apache2# cat index.html.1
Welcome to chess club 42
```

Testing the hacker club named hunter2 on port 9000.

```
[root@CentOS65 ~]# wget 127.0.0.1:9000
--2014-05-11 12:02:37--  http://127.0.0.1:9000/
Connecting to 127.0.0.1:9000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26 [text/html]
Saving to: `index.html.2'

100%[===========================================>] 26         --.-K/s   in 0s

2014-05-11 12:02:37 (4.49 MB/s) - `index.html.2' saved [26/26]

root@linux:/etc/apache2# cat index.html.2
HaCkInG iS fUn At HuNtEr2
```

Cleaning up the temporary files.

```
[root@CentOS65 ~]# rm index.html index.html.1 index.html.2
```

15.5.8. firewall rules

If we attempt to access the site from another machine however, we will not be able to view the website yet. The firewall is blocking incoming connections. We need to open these incoming ports first

```
[root@CentOS65 ~]# iptables -I INPUT -p tcp --dport 80 -j ACCEPT
[root@CentOS65 ~]# iptables -I INPUT -p tcp --dport 7000 -j ACCEPT
[root@CentOS65 ~]# iptables -I INPUT -p tcp --dport 8000 -j ACCEPT
[root@CentOS65 ~]# iptables -I INPUT -p tcp --dport 9000 -j ACCEPT
```

And if we want these rules to remain active after a reboot, we need to save them

```
[root@CentOS65 ~]# service iptables save
iptables: Saving firewall rules to /etc/sysconfig/iptables:[  OK  ]
```

15.6. named virtual hosts on CentOS

15.6.1. named virtual hosts

The chess club and the model train club find the port numbers too hard to remember. They would prefere to have their website accessible by name.

We continue work on the same server that has three websites on three ports. We need to make sure those websites are accesible using the names `choochoo.local`, `chessclub42.local` and `hunter2.local`.

First, we need to enable named virtual hosts in the configuration

```
[root@CentOS65 ~]# vi /etc/httpd/conf/httpd.conf
[root@CentOS65 ~]# grep ^NameVirtualHost /etc/httpd/conf/httpd.conf
NameVirtualHost *:80
[root@CentOS65 ~]#
```

Next we need to create three new virtualhosts.

```
[root@CentOS65 ~]# vi /etc/httpd/conf.d/choochoo.local.conf
[root@CentOS65 ~]# vi /etc/httpd/conf.d/chessclub42.local.conf
[root@CentOS65 ~]# vi /etc/httpd/conf.d/hunter2.local.conf
[root@CentOS65 ~]# cat /etc/httpd/conf.d/choochoo.local.conf
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName choochoo.local
        DocumentRoot /var/www/html/choochoo
</VirtualHost>
[root@CentOS65 ~]# cat /etc/httpd/conf.d/chessclub42.local.conf
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName chessclub42.local
        DocumentRoot /var/www/html/chessclub42
</VirtualHost>
[root@CentOS65 ~]# cat /etc/httpd/conf.d/hunter2.local.conf
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName hunter2.local
        DocumentRoot /var/www/html/hunter2
</VirtualHost>
[root@CentOS65 ~]#
```

Notice that they all listen on `port 80` and have an extra `ServerName` directive.

15.6.2. name resolution

We need some way to resolve names. This can be done with DNS, which is discussed in another chapter. For this demo it is also possible to quickly add the three names to the `/etc/hosts` file.

```
[root@CentOS65 ~]# grep ^192 /etc/hosts
192.168.1.225 choochoo.local
192.168.1.225 chessclub42.local
192.168.1.225 hunter2.local
```

Note that you may have another ip address...

15.6.3. reload and verify

After a `service httpd reload` the websites should be available by name.

```
[root@CentOS65 ~]# service httpd reload
Reloading httpd:
[root@CentOS65 ~]# wget chessclub42.local
--2014-05-25 16:59:14--  http://chessclub42.local/
Resolving chessclub42.local... 192.168.1.225
Connecting to chessclub42.local|192.168.1.225|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 25 [text/html]
Saving to: âindex.htmlâ

100%[============================================>] 25        --.-K/s   in 0s

2014-05-25 16:59:15 (1014 KB/s) - `index.html' saved [25/25]

[root@CentOS65 ~]# cat index.html
Welcome to chess club 42
```

15.7. password protected website on CentOS

You can secure files and directories in your website with a `.htaccess` file that refers to a `.htpasswd` file. The `htpasswd` command can create a `.htpasswd` file that contains a userid and an (encrypted) password.

This screenshot creates a user and password for the hacker named `cliff` and uses the `-c` flag to create the `.htpasswd` file.

```
[root@CentOS65 ~]# htpasswd -c /var/www/.htpasswd cliff
New password:
Re-type new password:
Adding password for user cliff
[root@CentOS65 ~]# cat /var/www/.htpasswd
cliff:QNwTrymMLBctU
```

Hacker `rob` also wants access, this screenshot shows how to add a second user and password to `.htpasswd`.

```
[root@CentOS65 ~]# htpasswd /var/www/.htpasswd rob
New password:
Re-type new password:
Adding password for user rob
[root@CentOS65 ~]# cat /var/www/.htpasswd
cliff:QNwTrymMLBctU
rob:EC2vOCcrMXDoM
[root@CentOS65 ~]#
```

Both Cliff and Rob chose the same password (hunter2), but that is not visible in the `.htpasswd` file because of the different salts.

Next we need to create a `.htaccess` file in the `DocumentRoot` of the website we want to protect. This screenshot shows an example.

```
[root@CentOS65 ~]# cat /var/www/html/hunter2/.htaccess
AuthUserFile /var/www/.htpasswd
AuthName "Members only!"
AuthType Basic
require valid-user
```

Note that we are protecting the website on `port 9000` that we created earlier.

And because we put the website for the Hackerspace named hunter2 in a subdirectory of the default website, we will need to adjust the `AllowOvveride` parameter in `/etc/httpd/conf/httpd.conf` under the `<Directory "/var/www/html">` directive as this screenshot shows.

```
[root@CentOS65 ~]# vi /etc/httpd/conf/httpd.conf

<Directory "/var/www/html">

#
# Possible values for the Options directive are "None", "All",
# or any combination of:
#   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
#
# Note that "MultiViews" must be named *explicitly* --- "Options All"
# doesn't give it to you.
#
# The Options directive is both complicated and important.  Please see
# http://httpd.apache.org/docs/2.2/mod/core.html#options
# for more information.
#
    Options Indexes FollowSymLinks

#
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   Options FileInfo AuthConfig Limit
#
    AllowOverride Authconfig

#
# Controls who can get stuff from this server.
#
    Order allow,deny
    Allow from all

</Directory>
```

Now restart the apache2 server and test that it works!

15.8. troubleshooting apache

When apache restarts, it will verify the syntax of files in the configuration folder `/etc/apache2` on debian or `/etc/httpd` on CentOS and it will tell you the name of the faulty file, the line number and an explanation of the error.

```
root@linux:~# service apache2 restart
apache2: Syntax error on line 268 of /etc/apache2/apache2.conf: Syntax error o\
n  line  1  of  /etc/apache2/sites-enabled/chessclub42:  /etc/apache2/sites-
enabled\
/chessclub42:4: <VirtualHost> was not closed.\n/etc/apache2/sites-enabled/ches\
sclub42:1: <VirtualHost> was not closed.
Action 'configtest' failed.
The Apache error log may have more information.
 failed!
```

Below you see the problem... a missing / before on line 4.

```
root@linux:~# cat /etc/apache2/sites-available/chessclub42
<VirtualHost *:8000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/chessclub42
<VirtualHost>
```

Let us force another error by renaming the directory of one of our websites:

```
root@linux:~# mv /var/www/choochoo/ /var/www/chooshoo
root@linux:~# !ser
service apache2 restart
Restarting web server: apache2Warning: DocumentRoot [/var/www/choochoo] does n\
ot exist
Warning: DocumentRoot [/var/www/choochoo] does not exist
 ... waiting Warning: DocumentRoot [/var/www/choochoo] does not exist
Warning: DocumentRoot [/var/www/choochoo] does not exist
.
```

As you can see, apache will tell you exactly what is wrong.

You can also troubleshoot by connecting to the website via a browser and then checking the apache log files in `/var/log/apache`.

15.9. virtual hosts example

Below is a sample virtual host configuration. This virtual hosts overrules the default Apache `ErrorDocument` directive.

```
<VirtualHost 83.217.76.245:80>
ServerName cobbaut.be
ServerAlias www.cobbaut.be
DocumentRoot /home/paul/public_html
ErrorLog /home/paul/logs/error_log
CustomLog /home/paul/logs/access_log common
ScriptAlias /cgi-bin/ /home/paul/cgi-bin/
<Directory /home/paul/public_html>
    Options Indexes IncludesNOEXEC FollowSymLinks
    allow from all
</Directory>
ErrorDocument 404 http://www.cobbaut.be/cobbaut.php
</VirtualHost>
```

15.10. aliases and redirects

Apache supports aliases for directories, like this example shows.

```
Alias /paul/ "/home/paul/public_html/"
```

Similarly, content can be redirected to another website or web server.

```
Redirect permanent /foo http://www.foo.com/bar
```

15.11. more on .htaccess

You can do much more with `.htaccess`. One example is to use .htaccess to prevent people from certain domains to access your website. Like in this case, where a number of referer spammers are blocked from the website.

```
student@linux:~/cobbaut.be$ cat .htaccess
# Options +FollowSymlinks
RewriteEngine On
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-adipex.fw.nu.*$ [OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-levitra.asso.ws.*$ [NC,OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-tramadol.fw.nu.*$ [NC,OR]
RewriteCond %{HTTP_REFERER} ^http://(www\.)?buy-viagra.lookin.at.*$ [NC,OR]
 ...
RewriteCond %{HTTP_REFERER} ^http://(www\.)?www.healthinsurancehelp.net.*$ [NC]
RewriteRule .* - [F,L]
student@linux:~/cobbaut.be$
```

15.12. traffic

Apache keeps a log of all visitors. The `webalizer` is often used to parse this log into nice html statistics.

15.13. self signed cert on Debian

Below is a very quick guide on setting up Apache2 on Debian 7 with a self-signed certificate.

Chances are these packages are already installed.

```
root@linux:~# aptitude install apache2 openssl
No packages will be installed, upgraded, or removed.
0 packages upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B of archives. After unpacking 0 B will be used.
```

Create a directory to store the certs, and use `openssl` to create a self signed cert that is valid for 999 days.

```
root@linux:~# mkdir /etc/ssl/localcerts
root@linux:~# openssl req -new -x509 -days 999 -nodes -out /etc/ssl/local\
certs/apache.pem -keyout /etc/ssl/localcerts/apache.key
Generating a 2048 bit RSA private key
...
...
writing new private key to '/etc/ssl/localcerts/apache.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BE
State or Province Name (full name) [Some-State]:Antwerp
Locality Name (eg, city) []:Antwerp
Organization Name (eg, company) [Internet Widgits Pty Ltd]:linux-training.be
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Paul
Email Address []:
```

A little security never hurt anyone.

```
root@linux:~# ls -l /etc/ssl/localcerts/
total 8
-rw-r--r-- 1 root root 1704 Sep 16 18:24 apache.key
-rw-r--r-- 1 root root 1302 Sep 16 18:24 apache.pem
root@linux:~# chmod 600 /etc/ssl/localcerts/*
root@linux:~# ls -l /etc/ssl/localcerts/
total 8
-rw------- 1 root root 1704 Sep 16 18:24 apache.key
-rw------- 1 root root 1302 Sep 16 18:24 apache.pem
```

Enable the apache `ssl` mod.

```
root@linux:~# a2enmod ssl
Enabling module ssl.
See /usr/share/doc/apache2.2-common/README.Debian.gz on how to configure SSL\
 and create self-signed certificates.
To activate the new configuration, you need to run:
  service apache2 restart
```

Create the website configuration.

```
root@linux:~# vi /etc/apache2/sites-available/choochoos
root@linux:~# cat /etc/apache2/sites-available/choochoos
<VirtualHost *:7000>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/choochoos
        SSLEngine On
        SSLCertificateFile /etc/ssl/localcerts/apache.pem
        SSLCertificateKeyFile /etc/ssl/localcerts/apache.key
</VirtualHost>
root@linux:~#
```

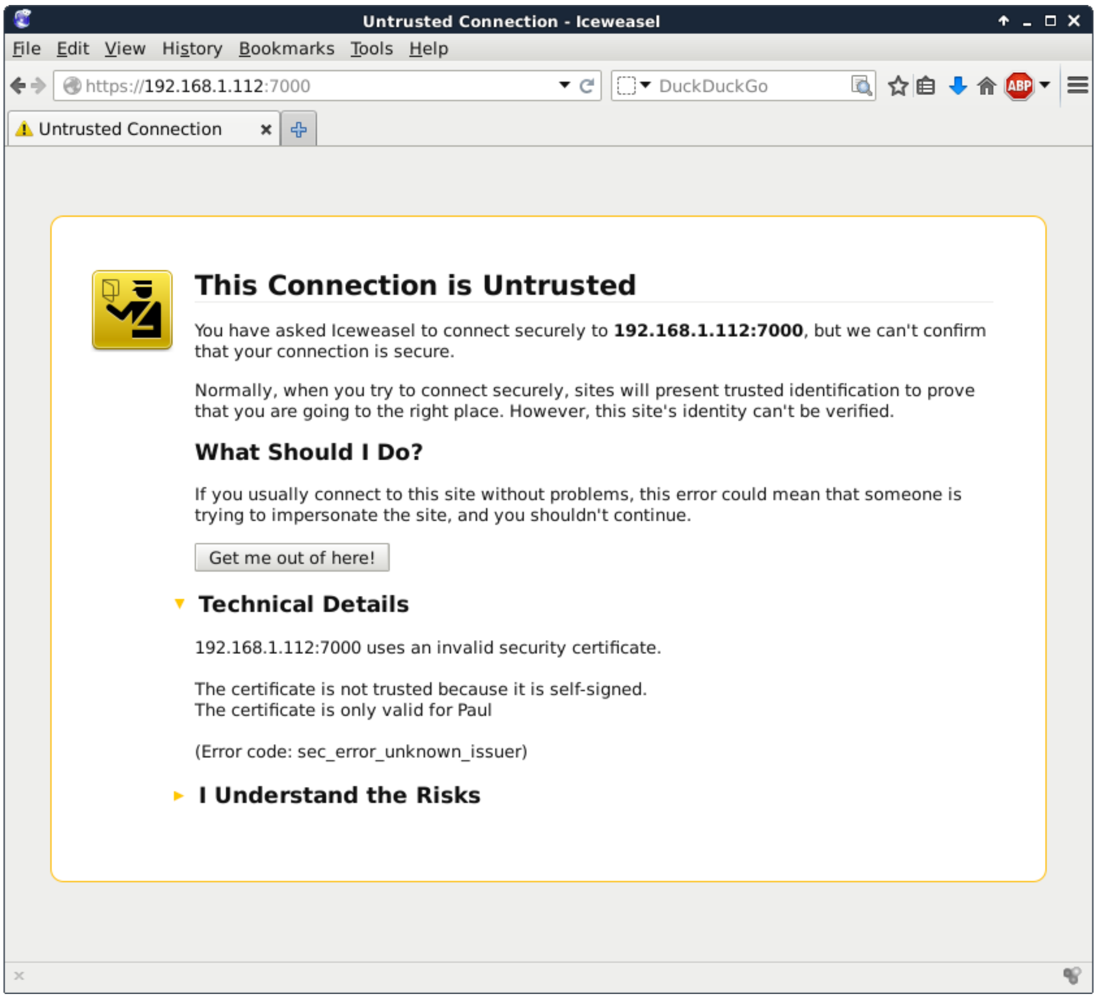
And create the website itself.

```
root@linux:/var/www/choochoos# vi index.html
root@linux:/var/www/choochoos# cat index.html
Choo Choo HTTPS secured model train Choo Choo
```

Enable the website and restart (or reload) apache2.

```
root@linux:/var/www/choochoos# a2ensite choochoos
Enabling site choochoos.
To activate the new configuration, you need to run:
  service apache2 reload
root@linux:/var/www/choochoos# service apache2 restart
Restarting web server: apache2 ... waiting .
```

Chances are your browser will warn you about the self signed certificate.


Untrusted Connection - Iceweasel
File Edit View History Bookmarks Tools Help
https://192.168.1.112:7000
DuckDuckGo
Untrusted Connection

This Connection is Untrusted

You have asked Iceweasel to connect securely to **192.168.1.112:7000**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Technical Details

192.168.1.112:7000 uses an invalid security certificate.

The certificate is not trusted because it is self-signed.
The certificate is only valid for Paul

(Error code: sec_error_unknown_issuer)

15.14. self signed cert on RHEL/CentOS

Below is a quick way to create a self signed cert for https on RHEL/CentOS. You may need these packages:

```
[root@paulserver ~]# yum install httpd openssl mod_ssl
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: ftp.belnet.be
 * extras: ftp.belnet.be
 * updates: mirrors.vooservers.com
base                                                   | 3.7 kB     00:00
Setting up Install Process
Package httpd-2.2.15-31.el6.centos.x86_64 already installed and latest version
Package openssl-1.0.1e-16.el6_5.15.x86_64 already installed and latest version
Package 1:mod_ssl-2.2.15-31.el6.centos.x86_64 already ins ... and latest version
Nothing to do
```

We use `openssl` to create the certificate.

```
[root@paulserver ~]# mkdir certs
[root@paulserver ~]# cd certs
[root@paulserver certs]# openssl genrsa -out ca.key 2048
```

```
Generating RSA private key, 2048 bit long modulus
.........+++
..........................................................+++
e is 65537 (0x10001)
[root@paulserver certs]# openssl req -new -key ca.key -out ca.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:BE
State or Province Name (full name) []:antwerp
Locality Name (eg, city) [Default City]:antwerp
Organization Name (eg, company) [Default Company Ltd]:antwerp
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:paulserver
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@paulserver certs]# openssl x509 -req -days 365 -in ca.csr -signkey ca.ke\
y -out ca.crt
Signature ok
subject=/C=BE/ST=antwerp/L=antwerp/O=antwerp/CN=paulserver
Getting Private key
```

We copy the keys to the right location (You may be missing SELinux info here).

```
[root@paulserver certs]# cp ca.crt /etc/pki/tls/certs/
[root@paulserver certs]# cp ca.key ca.csr /etc/pki/tls/private/
```

We add the location of our keys to this file, and also add the `NameVirtualHost *:443` directive.

```
[root@paulserver certs]# vi /etc/httpd/conf.d/ssl.conf
[root@paulserver certs]# grep ^SSLCerti /etc/httpd/conf.d/ssl.conf
SSLCertificateFile /etc/pki/tls/certs/ca.crt
SSLCertificateKeyFile /etc/pki/tls/private/ca.key
```

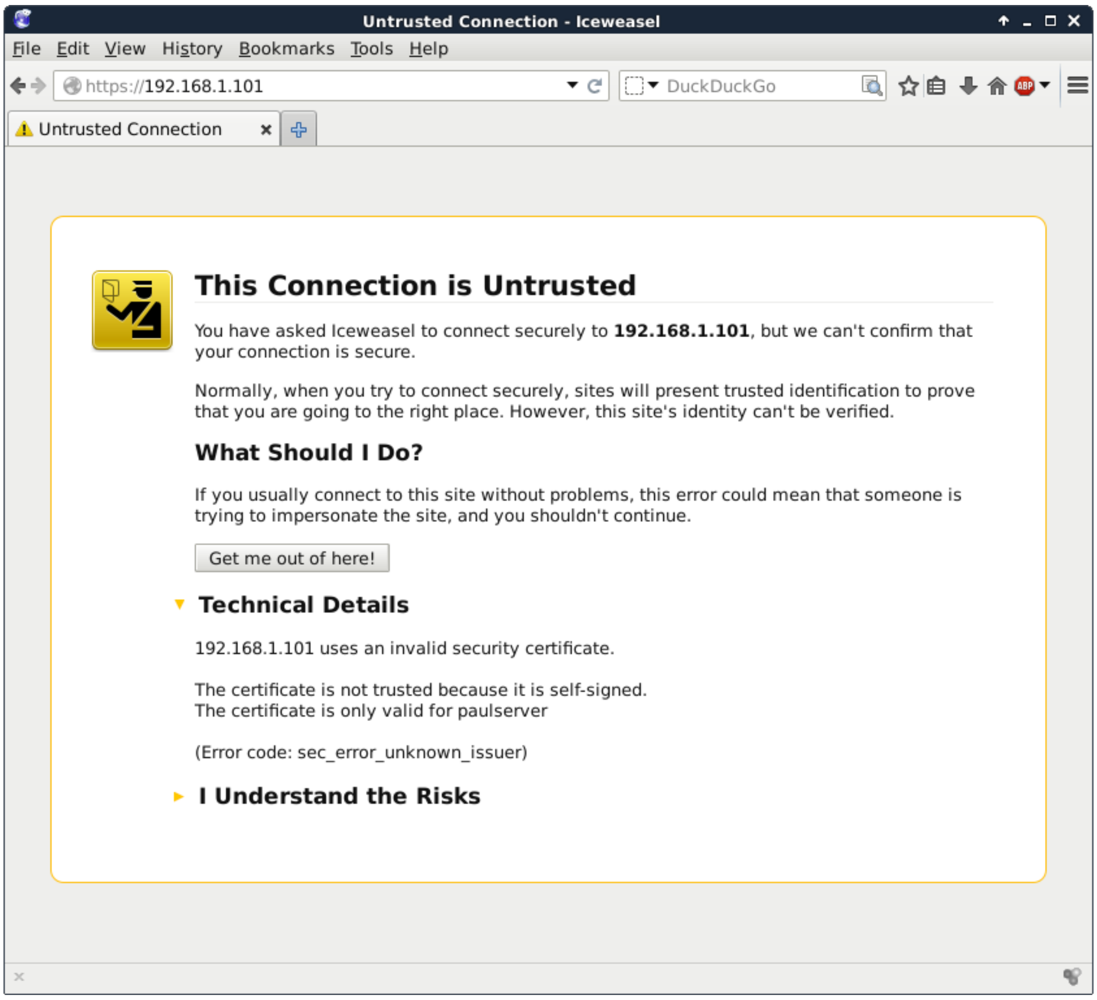
Create a website configuration.

```
[root@paulserver certs]# vi /etc/httpd/conf.d/choochoos.conf
[root@paulserver certs]# cat /etc/httpd/conf.d/choochoos.conf
<VirtualHost *:443>
        SSLEngine on
        SSLCertificateFile /etc/pki/tls/certs/ca.crt
        SSLCertificateKeyFile /etc/pki/tls/private/ca.key
        DocumentRoot /var/www/choochoos
        ServerName paulserver
</VirtualHost>
[root@paulserver certs]#
```

Create a simple website and restart apache.

```
[root@paulserver certs]# mkdir /var/www/choochoos
[root@paulserver certs]# echo HTTPS model train choochoos > /var/www/choochoos/\
index.html
[root@paulserver httpd]# service httpd restart
Stopping httpd:                                            [  OK  ]
Starting httpd:                                            [  OK  ]
```

And your browser will probably warn you that this certificate is self signed.


Untrusted Connection - Iceweasel
File Edit View History Bookmarks Tools Help
https://192.168.1.101
DuckDuckGo
Untrusted Connection

This Connection is Untrusted

You have asked Iceweasel to connect securely to **192.168.1.101**, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

Get me out of here!

Technical Details

192.168.1.101 uses an invalid security certificate.

The certificate is not trusted because it is self-signed.
The certificate is only valid for paulserver

(Error code: sec_error_unknown_issuer)

I Understand the Risks

15.15. practice: apache

1. Verify that Apache is installed and running.

2. Browse to the Apache HTML manual.

3. Create three virtual hosts that listen on ports 8472, 31337 and 1201. Test that it all works.

4. Create three named virtual hosts startrek.local, starwars.local and stargate.local. Test that it all works.

5. Create a virtual hosts that listens on another ip-address.

6. Protect one of your websites with a user/password combo.

16. introduction to sql using mysql or mariadb

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

MySQL is a relational database server that understands Structured Query Language (`SQL`). MySQL was developed by the Swedish Company `MySQL AB` when it was first released in 1995. In 2008 MySQL AB was bought by Sun Microsystems, which was acquired in 2010 by Oracle Corporation. So, ironically, MySQL is now owned by the company that it originally sought to compete with. This was a big concern for the open source community, so a co-founder of MySQL, Michael Widenius, forked MySQL to create MariaDB. On many Linux distributions, MySQL has even been replaced by MariaDB as the default database server.

MySQL's rise to fame can for a significant part be attributed to the fact that it was used by early 'web 2.0' companies like Facebook, Amazon, etc. as part of the so-called LAMP stack. LAMP stands for Linux, Apache, MySQL, and PHP/Perl/Python. Together, these separate applications form a powerful combination that was (and still is) used as a platform to build dynamic web applications. MySQL is still very popular in the web development community, but it is also used in many other organizations with huge databases like Facebook, Flickr, Google, Nokia, Wikipedia and YouTube.

This chapter will teach you how to install and configure MySQL and MariaDB, how to create and use small databases and tables, and how to run SQL queries from the command line or in a script.

Remark that this is not a database course, so more advanced subjects like high availability, performance tuning, etc. are beyond the scope of this book.

16.1. Installing MySQL

On Ubuntu or Enterprise Linux you can install the package `mysql-server` with a package manager. This will install both the MySQL server and client application.

On recent versions of Debian (e.g. 12), the package `mysql-server` is *not* available in the default repositories. You can download a .deb package from the MySQL website, or use the MariaDB package instead by installing the package `default-mysql-server`.

On recent versions of Fedora (e.g. 39), installing the package `mysql-server` will in fact install MariaDB instead of MySQL. You can install MySQL from the package `community-mysql-server` or download a package from the MySQL website.

For example, the installation on Ubuntu 24.04 LTS:

```
student@ubuntu:~$ sudo apt install mysql-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl
 ↪  libevent-pthreads-2.1-7t64 libfcgi-bin libfcgi-perl libfcgi0t64
 ↪  libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl
```

```
  libhttp-date-perl libhttp-message-perl libio-html-perl
↪ liblwp-mediatypes-perl libmecab2 libprotobuf-lite32t64 libtimedate-perl
↪ liburi-perl mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-common mysql-server-8.0 mysql-server-core-8.0
Suggested packages:
  libdata-dump-perl libipc-sharedcache-perl libio-compress-brotli-perl
↪ libbusiness-isbn-perl libregexp-ipv6-perl libwww-perl mailx tinyca
The following NEW packages will be installed:
  libcgi-fast-perl libcgi-pm-perl libclone-perl libencode-locale-perl
↪ libevent-pthreads-2.1-7t64 libfcgi-bin libfcgi-perl libfcgi0t64
↪ libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl
  libhttp-date-perl libhttp-message-perl libio-html-perl
↪ liblwp-mediatypes-perl libmecab2 libprotobuf-lite32t64 libtimedate-perl
↪ liburi-perl mecab-ipadic mecab-ipadic-utf8 mecab-utils mysql-client-8.0
  mysql-client-core-8.0 mysql-common mysql-server mysql-server-8.0
↪ mysql-server-core-8.0
0 upgraded, 28 newly installed, 0 to remove and 175 not upgraded.
Need to get 29.6 MB of archives.
After this operation, 242 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://archive.ubuntu.com/ubuntu noble/main amd64 mysql-common all
↪ 5.8+1.1.0build1 [6,746 B]
[ ... some output omitted ... ]
student@ubuntu:~$ systemctl status mysql
● mysql.service - MySQL Community Server
     Loaded: loaded (/usr/lib/systemd/system/mysql.service; enabled; preset:
↪ enabled)
     Active: active (running) since Wed 2024-10-09 09:50:34 UTC; 1min 40s ago
    Process: 2527 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre
↪ (code=exited, status=0/SUCCESS)
   Main PID: 2536 (mysqld)
     Status: "Server is operational"
      Tasks: 37 (limit: 2275)
     Memory: 365.2M (peak: 379.5M)
        CPU: 1.568s
     CGroup: /system.slice/mysql.service
             └─2536 /usr/sbin/mysqld

Oct 09 09:50:34 ubuntu systemd[1]: Starting mysql.service - MySQL Community
↪ Server ...
Oct 09 09:50:34 ubuntu systemd[1]: Started mysql.service - MySQL Community
↪ Server.
```

As you can see, the database server is started immediately on Ubuntu. This will not be the case on Enterprise Linux, where you have to start the service manually:

```
[student@el ~]$ sudo dnf install -y mysql-server
[ ... output omitted ... ]
[student@el ~]$ systemctl status mysqld.service
○ mysqld.service - MySQL 8.0 database server
     Loaded: loaded (/usr/lib/systemd/system/mysqld.service; disabled;
↪ preset: disabled)
     Active: inactive (dead)
[student@el ~]$ sudo systemctl enable --now mysqld.service
Created symlink /etc/systemd/system/multi-user.target.wants/mysqld.service →
↪ /usr/lib/systemd/system/mysqld.service.
[student@el ~]$ systemctl status mysqld.service
● mysqld.service - MySQL 8.0 database server
```

```
     Loaded: loaded (/usr/lib/systemd/system/mysqld.service; enabled;
  ↪  preset: disabled)
     Active: active (running) since Wed 2024-10-09 09:56:00 UTC; 2s ago
    Process: 7915 ExecStartPre=/usr/libexec/mysql-check-socket (code=exited,
  ↪  status=0/SUCCESS)
    Process: 7937 ExecStartPre=/usr/libexec/mysql-prepare-db-dir
  ↪  mysqld.service (code=exited, status=0/SUCCESS)
   Main PID: 8011 (mysqld)
     Status: "Server is operational"
      Tasks: 38 (limit: 11128)
     Memory: 455.9M
        CPU: 2.541s
     CGroup: /system.slice/mysqld.service
             └─8011 /usr/libexec/mysqld --basedir=/usr
```

To verify the installed version, use `dpkg -l` on Ubuntu:

```
student@ubuntu:~$ dpkg -l mysql-server | tail -1 | awk '{ print $2,$3 }'
mysql-server 8.0.39-0ubuntu0.24.04.2
```

Issue `rpm -q` to get version information about MySQL on Enterprise Linux:

```
[student@el ~]$ rpm -q mysql-server
mysql-server-8.0.36-1.el9_3.x86_64
```

Finally, to check if MySQL is listening on a network socket (and which one), use `ss`:

```
[student@el ~]$ sudo ss -tlnp
State   Recv-Q  Send-Q Local Address:Port  Peer Address:Port  Process
  ↪
LISTEN  0       128          0.0.0.0:22         0.0.0.0:*
  ↪  users:(("sshd",pid=629,fd=3))
LISTEN  0       4096         0.0.0.0:111        0.0.0.0:*
  ↪  users:(("rpcbind",pid=517,fd=4),("systemd",pid=1,fd=31))
  ↪
LISTEN  0       70                 *:33060            *:*
  ↪  users:(("mysqld",pid=847,fd=21))
LISTEN  0       151                *:3306             *:*
  ↪  users:(("mysqld",pid=847,fd=23))
LISTEN  0       128             [::]:22            [::]:*
  ↪  users:(("sshd",pid=629,fd=4))
LISTEN  0       4096            [::]:111           [::]:*
  ↪  users:(("rpcbind",pid=517,fd=6),("systemd",pid=1,fd=34))
```

MySQL listens on port 3306 by default. Some systems may have a default configuration causing the service to only listen on the loopback interface. We'll discuss below how to change this. In this example, we see that MySQL listens on all interfaces (`*:3306`). It also listens on port 33060, which is used for the X Protocol, a new protocol used by MySQL Shell.

16.2. Installing MariaDB

On Debian, the package `mysql-server` is *not* available. You can install the package `default-mysql-server` or `mariadb-server` instead, which has the same result.

```
student@debian:~$ sudo apt install default-mysql-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  galera-4 gawk libcgi-fast-perl libcgi-pm-perl libclone-perl
    ↪ libconfig-inifiles-perl libdaxctl1 libdbd-mariadb-perl libdbi-perl
    ↪ libencode-locale-perl libfcgi-bin libfcgi-perl libfcgi0ldbl libgpm2
  libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl
    ↪ libhttp-date-perl libhttp-message-perl libio-html-perl
    ↪ liblwp-mediatypes-perl liblzo2-2 libmariadb3 libncurses6 libndctl6
    ↪ libpmem1
  libregexp-ipv6-perl libsigsegv2 libsnappy1v5 libterm-readkey-perl
    ↪ libtimedate-perl liburi-perl liburing2 mariadb-client
    ↪ mariadb-client-core mariadb-common mariadb-plugin-provider-bzip2
  mariadb-plugin-provider-lz4 mariadb-plugin-provider-lzma
    ↪ mariadb-plugin-provider-lzo mariadb-plugin-provider-snappy
    ↪ mariadb-server mariadb-server-core mysql-common psmisc pv socat
Suggested packages:
  gawk-doc libmldbm-perl libnet-daemon-perl libsql-statement-perl gpm
    ↪ libdata-dump-perl libipc-sharedcache-perl libbusiness-isbn-perl
    ↪ libwww-perl mailx mariadb-test netcat-openbsd doc-base
The following NEW packages will be installed:
  default-mysql-server galera-4 gawk libcgi-fast-perl libcgi-pm-perl
    ↪ libclone-perl libconfig-inifiles-perl libdaxctl1 libdbd-mariadb-perl
    ↪ libdbi-perl libencode-locale-perl libfcgi-bin libfcgi-perl
  libfcgi0ldbl libgpm2 libhtml-parser-perl libhtml-tagset-perl
    ↪ libhtml-template-perl libhttp-date-perl libhttp-message-perl
    ↪ libio-html-perl liblwp-mediatypes-perl liblzo2-2 libmariadb3 libncurses6
    ↪ libndctl6
  libpmem1 libregexp-ipv6-perl libsigsegv2 libsnappy1v5 libterm-readkey-perl
    ↪ libtimedate-perl liburi-perl liburing2 mariadb-client
    ↪ mariadb-client-core mariadb-common mariadb-plugin-provider-bzip2
  mariadb-plugin-provider-lz4 mariadb-plugin-provider-lzma
    ↪ mariadb-plugin-provider-lzo mariadb-plugin-provider-snappy
    ↪ mariadb-server mariadb-server-core mysql-common psmisc pv socat
    0 upgraded, 48 newly installed, 0 to remove and 58 not upgraded.
Need to get 19.4 MB of archives.
After this operation, 196 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

On EL (including Fedora), install Mariadb with `sudo dnf install mariadb-server`.

Again, on Debian, the service will be started automatically, while on EL, you have to do this yourself.

```
[student@fedora ~]$ sudo dnf install mariadb-server
[ ... output omitted ... ]
[student@fedora ~]$ systemctl status mariadb.service
○ mariadb.service - MariaDB 10.5 database server
     Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled;
↪ preset: disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: inactive (dead)
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
[student@fedora ~]$ sudo systemctl enable --now mariadb.service
Created symlink /etc/systemd/system/mysql.service →
↪ /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/mysqld.service →
↪ /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service
↪ → /usr/lib/systemd/system/mariadb.service.
```

```
[student@fedora ~]$ systemctl status mariadb.service
● mariadb.service - MariaDB 10.5 database server
     Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled;
  ↪  preset: disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: active (running) since Wed 2024-10-09 10:10:41 UTC; 1s ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
    Process: 4593 ExecStartPre=/usr/libexec/mariadb-check-socket
  ↪  (code=exited, status=0/SUCCESS)
    Process: 4616 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir
  ↪  mariadb.service (code=exited, status=0/SUCCESS)
    Process: 4717 ExecStartPost=/usr/libexec/mariadb-check-upgrade
  ↪  (code=exited, status=0/SUCCESS)
   Main PID: 4699 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 15 (limit: 2322)
     Memory: 66.2M
        CPU: 357ms
     CGroup: /system.slice/mariadb.service
             └─4699 /usr/libexec/mariadbd --basedir=/usr
```

16.3. Securing MySQL/MariaDB

After installation, some default settings are in place that are not very secure. For example, MySQL has its own user management system, which is separate from the Linux users. It also has a root/admin user and initially, the password is empty.

You can (and should) run the `mysql_secure_installation` script to improve security. This script will ask you to set a root password, remove anonymous users, disallow root login remotely and remove the test database.

Remark that for MariaDB, the script is also called `mysql_secure_installation` (although it may be a symbolic link to a script called `mariadb-secure-installation`)!

The example below shows how the script runs on Debian. Carefully read the output and answer the questions.

```
student@debian:~$ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n]
Enabled successfully!
```

```
Reloading privilege tables..
 ... Success!

You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] n
 ... skipping.

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

The script performs the following tasks:

- Secure the root user. There are two ways to do this:
  - Switch to `unix_socket` authentication: this means that the root user does not get a password, but you can only log in as the database root user if you have superuser privileges on the system (i.e. with the command `sudo mysql`). This is a more recent method and may not be available on older distributions or MySQL/MariaDB versions.
  - Set a root password: you enter a password that you later can use to authenticate with the command `mysql -uroot -pPASSWORD` (see below). In this case, you don't need superuser privileges on the system, so it can be more convenient (but arguably less secure).

  - In the example above, we chose for `unix_socket` authentication, so we did *not* set a root password.

- Remove anonymous users: these are users with a blank user name and are used to define default access permissions. However, this can pose a security risk so it's better to remove them.
- Disallow root login remotely: the root user should only be allowed to connect from the local machine. This is a security measure to prevent unauthorized access.
- Remove the default test database that is present in new installations. There is no reason to keep it, especially in a production system.

16.4. Accessing MySQL

16.4.1. Linux users

The installation of `mysql` creates a user account in `/etc/passwd` and a group account in `/etc/group`. Details of the user may differ between distributions. For example, here is the output on Debian:

```
student@debian:~$ getent passwd mysql
mysql:x:104:109:MySQL Server,,,:/nonexistent:/bin/false
student@debian:~$ getent group mysql
mysql:x:109:
```

On Fedora, you get:

```
[student@fedora ~]$ getent passwd mysql
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin
[student@fedora ~]$ getent group mysql
mysql:x:27:
```

As you can see, it's not possible for the mysql user to log in, as the shell is set to `/bin/false` or `/sbin/nologin`. However, the process that runs the MySQL server will be started as this user:

```
student@debian:~$ ps -eo uid,user,gid,group,comm | grep maria
  104 mysql      109 mysql    mariadbd
```

16.4.2. MySQL Client Application

You can now access the MySQL or MariaDB server from the commandline using the `mysql` client application. If you chose `unix_socket` authentication, you can log in with `sudo mysql` (see the example below).

If you set a root password during the execution of the `mysql_secure_installation` script, you can log in with `mysql -uroot -p`. It will prompt you for the password.

```
student@debian:~$ sudo mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 39
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
 ↪  statement.

MariaDB [(none)]>
```

Or on Enterprise Linux, with MySQL installed:

```
[student@el ~]$ sudo mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.36 Source distribution

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
 ↪  statement.

mysql>
```

You can quit the MySQL client with the command `quit` or `exit`, or by pressing `Ctrl-D`.

You could also put the password in clear text on the command line, but that would not be very secure. Anyone with access to your bash history would be able to read your mysql root password. Remark that there should *not* be a space between the option `-p` and the password!

```
student@linux~$ mysql -uroot -phunter2
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

On the prompt, you can issue SQL statements (ending with a semicolon) to interact with the database. For example, to see which databases are available, use the command `show databases;`. More examples are given below.

Remark that the prompt has full history, even over previous sessions. You can use the up and down arrow keys to navigate through the history or search it using `Ctrl-R`.

16.4.3. ~/.my.cnf

You can save the MySQL configuration in your home directory in the hidden file `.my.cnf`. In the screenshot below we put the root user and password in .my.cnf.

```
student@linux:~$ pwd
/home/kevin
student@linux:~$ cat .my.cnf
[client]
user=root
password=hunter2
student@linux:~$
```

This enables us to log on as the `root mysql` user just by typing `mysql`.

```
student@linux:~$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 56
Server version: 5.5.24-0ubuntu0.12.04.1 (Ubuntu)
```

16.5. Interacting with MySQL/MariaDB databases

16.5.1. Listing all databases

You can use the `mysql` command to take a look at the databases and to execute SQL queries on them. The screenshots below show you how. First, we log on to our MySQL server and execute the command `show databases` to see which databases exist on our MySQL server.

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.004 sec)
```

16.5.2. Creating a database

You can create a new database with the `create database` command.

```
MariaDB [(none)]> create database famouspeople;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| famouspeople       |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.001 sec)
```

You can create the database on the condition that it doesn't already exist. If you try to create a database that already exists, you will get a warning.

```
MariaDB [(none)]> create database if not exists famouspeople;
Query OK, 0 rows affected, 1 warning (0.000 sec)

MariaDB [(none)]> show warnings;
+-------+------+----------------------------------------------------+
| Level | Code | Message                                            |
+-------+------+----------------------------------------------------+
| Note  | 1007 | Can't create database 'famouspeople'; database exists |
+-------+------+----------------------------------------------------+
1 row in set (0.000 sec)
```

16.5.3. Using a database

Next we tell `mysql` to use one particular database with the `use $database` command. This screenshot shows how to make `famouspeople` the current database (in use).

```
MariaDB [(none)]> use famouspeople;
Database changed
MariaDB [famouspeople]>
```

On MariaDB, the prompt shows the name of the current database. On MySQL, you can use the command `select database();` to see the current database.

```
mysql> use famouspeople;
Database changed
mysql> select database();
+--------------+
| database()   |
+--------------+
| famouspeople |
+--------------+
1 row in set (0.00 sec)
```

16.5.4. Database access

To create a database user and give them access to a MySQL database, use the `grant` command.

```
MariaDB [famouspeople]> grant all on famouspeople.* to kevin@localhost
 ↪  IDENTIFIED BY "hunter2";
Query OK, 0 rows affected (0.008 sec)
```

This user can now log in to the database `famouspeople` from the local machine:

```
student@debian:~$ mysql -ukevin -phunter2 famouspeople
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 43
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
 ↪  statement.

MariaDB [famouspeople]>
```

The `mysql` database has a table `user` that contains information about (database) users. You can use the `select` command to see this information. The table has a lot of columns, as you can see in the output of the `describe` query, so we will only select a few important fields: `Host`, `User` and `Password`.

```
MariaDB [(none)]> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [mysql]> describe user;
+------------------------+---------------------+------+-----+----------+----
 ↪  ---+
| Field                  | Type                | Null | Key | Default  | Extra |
+------------------------+---------------------+------+-----+----------+----
 ↪  ---+
| Host                   | char(255)           | NO   |     |          |       |
| User                   | char(128)           | NO   |     |          |       |
| Password               | longtext            | YES  |     | NULL     |       |
| Select_priv            | varchar(1)          | YES  |     | NULL     |       |
[ ... some output omitted ... ]
| default_role           | longtext            | NO   |     |          |       |
| max_statement_time     | decimal(12,6)       | NO   |     | 0.000000 |
↪   |
+------------------------+---------------------+------+-----+----------+----
↪   ---+
47 rows in set (0.002 sec)

MariaDB [mysql]> select User,Host,Password from user;
+-------------+-----------+-------------------------------------------+
| User        | Host      | Password                                  |
+-------------+-----------+-------------------------------------------+
| mariadb.sys | localhost |                                           |
| root        | localhost | invalid                                   |
| mysql       | localhost | invalid                                   |
| kevin       | localhost | *58815970BE77B3720276F63DB198B1FA42E5CC02 |
+-------------+-----------+-------------------------------------------+
4 rows in set (0.001 sec)
```

We see that there are four users in the `mysql` database. The `root` user has an invalid password, as does the `mysql` user. This means that they can not log in using the `mysql -uUSER` command. The `kevin` user does have a password hash, so he can log in with `mysql -ukevin -p`.

The `Host` field mentions `localhost`, which means that the user can only log in from the local machine. If you want to allow a user to log in from any machine, you can use the wildcard `%`, e.g.

```
MariaDB [mysql]> grant all on famouspeople.* to kyra@'%' IDENTIFIED BY
↪   "prey42";
Query OK, 0 rows affected (0.008 sec)

MariaDB [mysql]> select User,Host,Password from user;
+-------------+-----------+-------------------------------------------+
| User        | Host      | Password                                  |
+-------------+-----------+-------------------------------------------+
| mariadb.sys | localhost |                                           |
| root        | localhost | invalid                                   |
| mysql       | localhost | invalid                                   |
| kevin       | localhost | *58815970BE77B3720276F63DB198B1FA42E5CC02 |
| kyra        | %         | *AA6D4E2D10EF6CC8E577E2169F131F8A9144D29B |
+-------------+-----------+-------------------------------------------+
5 rows in set (0.001 sec)
```

16.5.5. Deleting a database

When a database is no longer needed, you can permanently remove it with the `drop database` command.

```
mysql> drop database demodb;
Query OK, 1 row affected (0.09 sec)
```

Here, too, you can add a condition to only drop the database if it exists.

```
MariaDB [(none)]> drop database if exists famouspeople;
Query OK, 0 rows affected (0.012 sec)

MariaDB [(none)]> show databases;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.000 sec)
```

16.5.6. Backup and restore a database

You can take a backup of a database, or move it to another computer using the `mysql` and `mysqldump` commands. In the screenshot below, we take a backup of the `famouspeople` database.

```
student@debian:~$ sudo mysqldump famouspeople > famouspeople-backup-$(date
 ↪  -I).sql
student@debian:~$ ls
famouspeople-backup-2024-10-09.sql
student@debian:~$ file mysql-backup-2024-10-09.sql
famouspeople-backup-2024-10-09.sql: Unicode text, UTF-8 text, with very long
 ↪  lines (2374)
```

To restore the database, you can use the `mysql` command.

```
student@debian:~$ sudo mysql famouspeople <
 ↪  famouspeople-backup-2024-10-09.sql
```

16.5.7. Using MySQL non-interactively

When you are automating the process of initialising a database on a new system, you may want to execute SQL queries non-interactively. The `mysql` command provides two ways of doing this:

- The `-e "$sql_query"` option
- The `stdin` stream of the command

The latter is useful when you need to execute multiple queries. In this case, you can use a *here document* to pass the queries to the `mysql` command.

This example shows the `-e` option, used by the root user:

```
student@debian:~$ sudo mysql -e 'select user,host,password from mysql.user;'
+-------------+-----------+-------------------------------------------+
| User        | Host      | Password                                  |
+-------------+-----------+-------------------------------------------+
| mariadb.sys | localhost |                                           |
| root        | localhost | invalid                                   |
| mysql       | localhost | invalid                                   |
| kevin       | localhost | *58815970BE77B3720276F63DB198B1FA42E5CC02 |
| kyra        | %         | *AA6D4E2D10EF6CC8E577E2169F131F8A9144D29B |
+-------------+-----------+-------------------------------------------+
```

And here is an example for a “normal” database user executing multiple queries in a *here document* (the database tables in the output will be created in the next section).

```
student@debian:~$ mysql -ukevin -phunter2 famouspeople << _EOF_
SHOW TABLES;
DESCRIBE people;
_EOF_
Tables_in_famouspeople
country
invoices
people
Field   Type    Null    Key Default Extra
name    varchar(70) YES     NULL
field   varchar(20) YES     NULL
birthyear   int(5)  YES     NULL
countrycode char(3) YES     NULL
```

16.5.8. Accessing a MySQL database over the network

In order to allow users to access the database server over the network, you may need to reconfigure it to listen on external network interfaces. On some installations, the default install only listens on the loopback interface `lo`. You can check this with `ss`:

```
student@debian:~$ ss -tln
State   Recv-Q  Send-Q     Local Address:Port     Peer Address:Port  Process
LISTEN  0       4096             0.0.0.0:111           0.0.0.0:*
LISTEN  0       128              0.0.0.0:22            0.0.0.0:*
LISTEN  0       80             127.0.0.1:3306          0.0.0.0:*
LISTEN  0       4096                [::]:111              [::]:*
LISTEN  0       128                 [::]:22               [::]:*
```

The `127.0.0.1:3306` output shows that MariaDB/MySQL is in this case only listening on the loopback interface. We need to change this setting in the configuration. The structure of the MariaDB/MySQL configuration files (stored in `/etc/mysql` on Debian or `/etc/my.cnf.d/` on EL) is a bit complicated, so it's not always clear where to make the necessary changes.

In the screenshot below, we search all configuration files for the setting we need, i.e. `bind-address` and print the file and line number where it is found.

```
student@debian:~$ find /etc/mysql/ -type f -exec grep -nH '^bind-address' {}
 ↪  \;
grep: /etc/mysql/debian.cnf: Permission denied
/etc/mysql/mariadb.conf.d/50-server.cnf:27:bind-address            =
 ↪  127.0.0.1
```

In this case, the setting we need is in the file `/etc/mysql/mariadb.conf.d/50-server.cnf`. We can change the setting to `0.0.0.0` to make the server listen on all interfaces. Alternatively, you could specify the IP address of a specific network interface to only listen on that interface.

```
student@debian:~$ sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
student@debian:~$ find /etc/mysql/ -type f -exec grep -nH ' bind-address' {}
 ↪  \;
grep: /etc/mysql/debian.cnf: Permission denied
/etc/mysql/mariadb.conf.d/50-server.cnf:27:bind-address            = 0.0.0.0
student@debian:~$ sudo systemctl restart mariadb
student@debian:~$ ss -tlnp
State   Recv-Q  Send-Q     Local Address:Port     Peer Address:Port  Process
LISTEN  0       80               0.0.0.0:3306          0.0.0.0:*
LISTEN  0       4096             0.0.0.0:111           0.0.0.0:*
LISTEN  0       128              0.0.0.0:22            0.0.0.0:*
LISTEN  0       4096                 [::]:111              [::]:*
LISTEN  0       128                  [::]:22               [::]:*
```

After making the necessary change and restarting the service, the server now listens on all interfaces (`0.0.0.0:3306`). You can then access the database from another machine using the IP address (or hostname, if it has a DNS record) of the server.

```
[student@fedora ~]$ mysql -h 192.168.56.21 -ukyra -pprey42 famouspeople -e
 ↪  'show tables;'
+------------------------+
| Tables_in_famouspeople |
+------------------------+
| country                |
| invoices               |
| people                 |
+------------------------+
```

Remark that you may need to adjust the firewall settings on the server to allow incoming connections on port 3306! Also, the user should be allowed to connect from the remote host. In the example above, the user `kyra` is allowed to connect from any host (`%`). If you do the same for the user `kevin`, this will fail since he is only allowed to connect from `localhost`.

```
[student@fedora ~]$ mysql -h 192.168.56.21 -ukevin -phunter2 -e 'show
 ↪  tables;'
ERROR 1045 (28000): Access denied for user 'kevin'@'192.168.56.109' (using
 ↪  password: YES)
```

16.6. MySQL tables

In a relational database, you can create tables to store data. A table is a collection of rows and columns. Each row represents a record and each column represents a field in the record.

16.6.1. Listing tables

You can see a list of tables in the current database with the `show tables;` command. Our `famouspeople` database has no tables yet.

```
MariaDB [(none)]> use famouspeople;
Database changed
MariaDB [famouspeople]> show tables;
Empty set (0.000 sec)
```

16.6.2. Creating a table

The `create table` command will create a new table.

This screenshot shows the creation of a `country` table. We use the `countrycode` as a `primary key` (all country codes are uniquely defined). Most country codes are two or three letters, so a `char` of three uses less space than a `varchar` of three. The `country name` and the name of the capital are both defined as `varchar`. The population can be seen as an `integer`.

```
MariaDB [famouspeople]> create table country (
    -> countrycode char(3) not null,
    -> countryname varchar(70) not null,
    -> population int,
    -> countrycapital varchar(50),
    -> primary key (countrycode)
    -> );
Query OK, 0 rows affected (0.017 sec)

MariaDB [famouspeople]> show tables;
+------------------------+
| Tables_in_famouspeople |
+------------------------+
| country                |
+------------------------+
1 row in set (0.001 sec)
```

You are allowed to type the `create table` command on one long line, but administrators often use multiple lines to improve readability.

```
MariaDB [famouspeople]> create table country ( countrycode char(3) NOT NULL,
 ↪  countryname varchar(70) NOT NULL, population int, countrycapital
 ↪  varchar(50), primary key (countrycode) );
Query OK, 0 rows affected (0.18 sec)
```

16.6.3. Describing a table

To see a description of the structure of a table, issue the `describe $tablename` command as shown below.

```
MariaDB [famouspeople]> describe country;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| countrycode    | char(3)     | NO   | PRI | NULL    |       |
| countryname    | varchar(70) | NO   |     | NULL    |       |
| population     | int(11)     | YES  |     | NULL    |       |
| countrycapital | varchar(50) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
4 rows in set (0.002 sec)
```

16.6.4. Removing a table

To remove a table from a database, issue the `drop table $tablename` command as shown below.

```
MariaDB [famouspeople]> drop table country;
Query OK, 0 rows affected (0.020 sec)
```

16.7. MySQL records

16.7.1. Creating records

Use `insert` to enter data into the table. The screenshot shows several insert statements that insert values depending on the position of the data in the statement.

```
MariaDB [famouspeople]> insert into country values
 ↪  ('BE','Belgium','11977634','Brussels');
Query OK, 1 row affected (0.003 sec)

```

```
MariaDB [famouspeople]> insert into country values
 ↪  ('DE','Germany','84119100','Berlin');
Query OK, 1 row affected (0.002 sec)

MariaDB [famouspeople]> insert into country values
 ↪  ('JP','Japan','123201945','Tokyo');
Query OK, 1 row affected (0.001 sec)
```

Some administrators prefer to use uppercase for SQL keywords. The MySQL client accepts both.

```
MariaDB [famouspeople]> INSERT INTO country VALUES
 ↪  ('FR','France','68374591','Paris');
Query OK, 1 row affected (0.003 sec)
```

Note that you get an error when using a duplicate `primary key`.

```
MariaDB [famouspeople]> insert into country values
 ↪  ('DE','Germany','84119100','Berlin');
ERROR 1062 (23000): Duplicate entry 'DE' for key 'PRIMARY'
```

16.7.2. Viewing all records

Below an example of a simple `select` query to look at the contents of a table.

```
MariaDB [famouspeople]> select * from country;
+-------------+-------------+------------+----------------+
| countrycode | countryname | population | countrycapital |
+-------------+-------------+------------+----------------+
| BE          | Belgium     |   11977634 | Brussels       |
| DE          | Germany     |   84119100 | Berlin         |
| FR          | France      |   68374591 | Paris          |
| JP          | Japan       |  123201945 | Tokyo          |
+-------------+-------------+------------+----------------+
4 rows in set (0.001 sec)
```

16.7.3. Updating records

Consider the following `insert` statement. The capital of Spain is not Barcelona, it is Madrid.

```
MariaDB [famouspeople]> insert into country values
 ↪  ('ES','Spain','47280433','Barcelona');
Query OK, 1 row affected (0.003 sec)
```

Using an `update` statement, the record can be updated.

```
MariaDB [famouspeople]> update country set countrycapital='Madrid' where
 ↪  countrycode='ES';
Query OK, 1 row affected (0.003 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

16.7.4. Viewing selected records

We can use a `select` statement to verify this change, using a `where` clause to only show the record(s) we want to see.

```
MariaDB [famouspeople]> select * from country where countrycode='ES';
+-------------+-------------+------------+----------------+
| countrycode | countryname | population | countrycapital |
+-------------+-------------+------------+----------------+
| ES          | Spain       |   47280433 | Madrid         |
+-------------+-------------+------------+----------------+
1 row in set (0.000 sec)
```

Another example of the `where` clause with the wildcard `%`. This will show all countries with a country code ending in E.

```
MariaDB [famouspeople]> select * from country where countrycode like '%E';
+-------------+-------------+------------+----------------+
| countrycode | countryname | population | countrycapital |
+-------------+-------------+------------+----------------+
| BE          | Belgium     |   11977634 | Brussels       |
| DE          | Germany     |   84119100 | Berlin         |
+-------------+-------------+------------+----------------+
2 rows in set (0.000 sec)
```

What is the total population of these countries?

```
MariaDB [famouspeople]> select sum(population) from country where
 ↪  countrycode like '%E';
+-----------------+
| sum(population) |
+-----------------+
|        96096734 |
+-----------------+
1 row in set (0.000 sec)
```

16.7.5. Ordering records

We know that `select` allows us to see all records in a table. Using the `order by` clause, we can change the order in which the records are presented.

```
MariaDB [famouspeople]> select countryname,population from country order by
 ↪  population;
+-------------+------------+
| countryname | population |
+-------------+------------+
| Belgium     |   11977634 |
| Spain       |   47280433 |
| France      |   68374591 |
| Germany     |   84119100 |
| Japan       |  123201945 |
+-------------+------------+
5 rows in set (0.001 sec)
```

16.7.6. Grouping records

Consider this table of people. The screenshot shows how to use the `avg` function to calculate an average.

```
MariaDB [famouspeople]> select * from people;
+-----------------+-----------+-----------+-------------+
| name            | field     | birthyear | countrycode |
+-----------------+-----------+-----------+-------------+
| Barack Obama    | politics  |      1961 | US          |
| Deng Xiaoping   | politics  |      1904 | CN          |
| Guy Verhofstadt | politics  |      1953 | BE          |
| Justine Henin   | tennis    |      1982 | BE          |
| Kim Clijsters   | tennis    |      1983 | BE          |
| Li Na           | tennis    |      1982 | CN          |
| Liu Yang        | astronaut |      1978 | CN          |
| Serena Williams | tennis    |      1981 | US          |
| Venus Williams  | tennis    |      1980 | US          |
+-----------------+-----------+-----------+-------------+
9 rows in set (0.001 sec)

MariaDB [famouspeople]> select AVG(birthyear) from people;
+----------------+
| AVG(birthyear) |
+----------------+
|      1967.1111 |
+----------------+
1 row in set (0.001 sec)
```

Using the `group by` clause, we can have an average per field.

```
MariaDB [famouspeople]> select field,avg(birthyear) from people group by
  ↪  field;
+-----------+----------------+
| field     | avg(birthyear) |
+-----------+----------------+
| astronaut |      1978.0000 |
| politics  |      1939.3333 |
| tennis    |      1981.6000 |
+-----------+----------------+
3 rows in set (0.000 sec)
```

16.7.7. Deleting records

You can use `delete` to permanently remove a record from a table.

```
MariaDB [famouspeople]> delete from country where countryname='Spain';
Query OK, 1 row affected (0.06 sec)

MariaDB [famouspeople]> select * from country where countryname='Spain';
Empty set (0.00 sec)
```

16.8. Joining two tables

16.8.1. Inner join

With an `inner join` you can take values from two tables and combine them in one result. Consider the country and the people tables from the previous section when looking at this screenshot of an `inner join`.

```
MariaDB [famouspeople]> select name,field,countryname from country inner
  ↪  join people on people.countrycode=country.countrycode;
+-----------------+-----------+-------------+
| name            | field     | countryname |
+-----------------+-----------+-------------+
| Deng Xiaoping   | politics  | China       |
| Guy Verhofstadt | politics  | Belgium     |
| Justine Henin   | tennis    | Belgium     |
| Kim Clijsters   | tennis    | Belgium     |
| Li Na           | tennis    | China       |
| Liu Yang        | astronaut | China       |
+-----------------+-----------+-------------+
6 rows in set (0.000 sec)
```

This `inner join` will show only records with a match on `countrycode` in both tables.

16.8.2. Left join

A `left join` is different from an `inner join` in that it will take all rows from the left table, regardless of a match in the right table.

```
MariaDB [famouspeople]> select Name,Field,countryname from country left join
  ↪  people on people.countrycode=country.countrycode;
+-----------------+-----------+-------------+
| Name            | Field     | countryname |
+-----------------+-----------+-------------+
| Deng Xiaoping   | politics  | China       |
| Guy Verhofstadt | politics  | Belgium     |
| Justine Henin   | tennis    | Belgium     |
| Kim Clijsters   | tennis    | Belgium     |
| Li Na           | tennis    | China       |
| Liu Yang        | astronaut | China       |
| NULL            | NULL      | Germany     |
| NULL            | NULL      | Spain       |
| NULL            | NULL      | France      |
| NULL            | NULL      | Japan       |
+-----------------+-----------+-------------+
10 rows in set (0.000 sec)
```

You can see that some countries are present, even when they have no matching records in the `people` table.

16.9. MySQL triggers

16.9.1. Using a *before* trigger

Consider the following `create table` command. The last field (`amount`) is the multiplication of the two fields named `unitprice` and `unitcount`.

```
MariaDB [famouspeople]> create table invoices (
    -> id char(8) NOT NULL,
    -> customerid char(3) NOT NULL,
    -> unitprice int,
    -> unitcount smallint,
    -> amount int );
Query OK, 0 rows affected (0.018 sec)
```

We can let MySQL do the calculation for that by using a `before trigger`. The screenshot below shows the creation of a trigger that calculates the amount by multiplying two fields that are about to be inserted.

```
MariaDB [famouspeople]> create trigger total_amount before INSERT on invoices
    -> for each row set new.amount = new.unitprice * new.unitcount ;
Query OK, 0 rows affected (0.02 sec)
```

Here we verify that the trigger works by inserting a new record, without providing the total amount. In the first attempt, we provide an empty value for the `amount` field. This will result in an error. In the second attempt, we do provide a value (0), but it will be overwritten by the trigger.

```
MariaDB [famouspeople]> insert into invoices values
 ↪  ('20241009','ABC','199','10','');
ERROR 1366 (22007): Incorrect integer value: '' for column
 ↪  `famouspeople`.`invoices`.`amount` at row 1
MariaDB [famouspeople]> insert into invoices values
 ↪  ('20241009','DEF','159','40','0');
Query OK, 1 row affected (0.004 sec)
```

Looking at the record proves that the trigger works.

```
MariaDB [famouspeople]> select * from invoices;
+----------+------------+-----------+-----------+--------+
| id       | customerid | unitprice | unitcount | amount |
+----------+------------+-----------+-----------+--------+
| 20241009 | ABC        |       199 |        10 |   1990 |
+----------+------------+-----------+-----------+--------+
1 row in set (0.000 sec)
```

16.9.2. Removing a trigger

When a `trigger` is no longer needed, you can delete it with the `drop trigger` command.

```
mysql> drop trigger total_amount;
Query OK, 0 rows affected (0.00 sec)
```

16.10. practice: mysql

Write a script that, without any user interaction, performs the following tasks on your favourite Linux distribution. Use variables to store the database name, user name and password. Use here documents for the SQL queries.

1. Install MariaDB or MySQL using the system package manager.
2. Execute the queries performed by the `mysql_secure_installation` script.

   Tip: in the case of MariaDB, you can locate the script and peruse its contents. In the script, a function `do_query` is defined that executes SQL statements. Look for lines containing `do_query` and extract the SQL statements.

3. Create a database user that has all privileges on the database `famouspeople` and that can log in from any host.
4. With this user, reconstruct the database `famouspeople` described above with all its contents: tables and records.

16.11. solution: mysql

In this solution, we install MariaDB on an Enterprise Linux system (tested on AlmaLinux 9). You can adapt to your favourite Linux distribution.

In order to find which queries are performed by `mysql_secure_installation`, you can do the following:

```
[student@el ~]$ which mysql_secure_installation
/usr/bin/mysql_secure_installation
[student@el ~]$ file /usr/bin/mysql_secure_installation
/usr/bin/mysql_secure_installation: symbolic link to mariadb-secure-installation
[student@el my.cnf.d]$ file /usr/bin/mariadb-secure-installation
/usr/bin/mariadb-secure-installation: a /usr/bin/sh script, ASCII text executable
[student@el ~]$ grep do_query /usr/bin/mariadb-secure-installation
do_query() {
    do_query "show create user root@localhost"
    do_query "UPDATE mysql.global_priv SET priv=json_set(priv, '$.plugin', 'mysql_native_password', '$.authentication_string', PASSWORD('$esc_pass')) WHERE User='root';"
    do_query "DELETE FROM mysql.global_priv WHERE User='';"
    do_query "DELETE FROM mysql.global_priv WHERE User='root' AND Host NOT IN ('localhost', '127.0.0.1', '::1');"
    do_query "DROP DATABASE IF EXISTS test;"
    do_query "DELETE FROM mysql.db WHERE Db='test' OR Db='test\\_%'"
    do_query "   "
  do_query "UPDATE mysql.global_priv SET priv=json_set(priv, '$.password_last_changed', UNIX_TIMESTAMP(), '$.plugin', 'mysql_native_password', '$.authentication_string', 'invalid', '$.auth_or', json_array(json_object(), json_object('plugin', 'unix_socket'))) WHERE User='root';"
```

We're not going to use the query to set the user password, since we'll keep `unix_socket` authentication.

```bash
#!/bin/bash
#
# init-db.sh -- Install and initialize a MariaDB database instance

set -o nounset
set -o errexit
set -o pipefail

# Variables
db_name="famouspeople"
db_user="kevin"
db_password="hunter2"

## Install the MariaDB server
dnf install -y mariadb-server
```

```

## Allow remote access to the database
sed -i 's/^#bind-address.*$/bind-address=0.0.0.0/'
↪  /etc/my.cnf.d/mariadb-server.cnf

## Start and enable the MariaDB service
if ! sudo systemctl is-enabled mariadb; then
  systemctl enable --now mariadb
else
  # Restart the service if it's already enabled
  systemctl restart mariadb
fi

## Perform the tasks of mysql_secure_installation:
# Remove anonymous users, disallow root login remotely, remove test database
# Root password is not set, we'll use `unix_socket` authentication
echo "Securing the MariaDB installation"

mysql << _EOF_
DELETE FROM mysql.global_priv WHERE User='';
DELETE FROM mysql.global_priv WHERE User='root' AND Host NOT IN
↪  ('localhost', '127.0.0.1', '::1');
DROP DATABASE IF EXISTS test;
DELETE FROM mysql.db WHERE Db='test' OR Db='test\\_%';
FLUSH PRIVILEGES;
_EOF_

## Initialize the database

echo "Creating a database and user"

mysql << _EOF_
CREATE DATABASE IF NOT EXISTS ${db_name};
GRANT ALL PRIVILEGES ON ${db_name}.* TO '${db_user}'@'%' IDENTIFIED BY
↪  '${db_password}';
FLUSH PRIVILEGES;
_EOF_

echo "Creating tables and adding content as the newly created database user"

mysql -u"${db_user}" -p"${db_password}" ${db_name} << _EOF_
DROP TABLE IF EXISTS country;
CREATE TABLE country (
  countrycode char(3) NOT NULL,
  countryname varchar(70) NOT NULL,
  population int,
  countrycapital varchar(50),
  PRIMARY KEY (countrycode)
);
INSERT INTO country VALUES ('BE','Belgium','11977634','Brussels');
INSERT INTO country VALUES ('DE','Germany','84119100','Berlin');
INSERT INTO country VALUES ('JP','Japan','123201945','Tokyo');
INSERT INTO country VALUES ('FR','France','68374591','Paris');
INSERT INTO country VALUES ('ES','Spain','47280433','Madrid');

DROP TABLE IF EXISTS people;
CREATE TABLE people (
  name varchar(70) NOT NULL,
  field varchar(20),
  birthyear int,
  countrycode char(3)
);
INSERT INTO people VALUES ('Barack Obama','politics','1961','US');
INSERT INTO people VALUES ('Deng Xiaoping','politics','1904','CN');
INSERT INTO people VALUES ('Guy Verhofstadt','politics','1953','BE');
INSERT INTO people VALUES ('Justine Henin','tennis','1982','BE');
INSERT INTO people VALUES ('Kim Clijsters','tennis','1983','BE');
INSERT INTO people VALUES ('Li Na','tennis','1982','CN');
INSERT INTO people VALUES ('Liu Yang','astronaut','1978','CN');
INSERT INTO people VALUES ('Serena Williams','tennis','1981','US');
INSERT INTO people VALUES ('Venus Williams','tennis','1980','US');
_EOF_
```

Afterwards, we can test the result from the command line. In this case, the database VM has the IP address 192.168.56.11. On another machine on the same network where the `mysql` client is installed, issue (for example) the following command:

```
student@linux:~$ mysql -h 192.168.56.11 -ukevin -phunter2 famouspeople -e
 ↪  "select * from people;"
+-----------------+-----------+-----------+-------------+
| name            | field     | birthyear | countrycode |
+-----------------+-----------+-----------+-------------+
| Barack Obama    | politics  |      1961 | US          |
| Deng Xiaoping   | politics  |      1904 | CN          |
| Guy Verhofstadt | politics  |      1953 | BE          |
| Justine Henin   | tennis    |      1982 | BE          |
| Kim Clijsters   | tennis    |      1983 | BE          |
| Li Na           | tennis    |      1982 | CN          |
| Liu Yang        | astronaut |      1978 | CN          |
| Serena Williams | tennis    |      1981 | US          |
| Venus Williams  | tennis    |      1980 | US          |
+-----------------+-----------+-----------+-------------+
```

17. 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/)

17.1. 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
```

17.2. 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.

17.3. 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
```

17.4. 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]$
```

17.5. 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
```

17.6. 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
```

17.7. 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
```

17.8. 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
```

17.9. 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`.

17.10. 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
```

18. 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/)

18.1. 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.

18.2. 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.
```

18.3. 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.

18.4. 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
```

18.5. 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
```

18.6. 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.

18.7. 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
```

18.8. 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`.

18.9. 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
```

Part VI.

Advanced text processing

19. 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.

19.1. * 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
```

19.2. ? 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
```

19.3. [ ] 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
```

19.4. 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
```

19.5. 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.

19.6. $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
```

19.7. 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 "*"
*
```

19.8. 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`?

19.9. 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' *`.

20. 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`.

20.1. 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!`

20.2. grep

20.2.1. 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 content 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.

20.2.2. 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:~$
```

20.2.3. 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
```

20.2.4. 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:~$
```

20.2.5. 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
```

20.2.6. 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.

20.2.7. 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:~$
```

20.2.8. 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
```

20.2.9. 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
```

20.3. rename

20.3.1. 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, ...).
```

20.3.2. 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
```

20.3.3. 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.

20.3.4. 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).

20.3.5. 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$
```

20.3.6. 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.

20.4. sed

20.4.1. 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
```

20.4.2. 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
```

20.4.3. 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
```

20.4.4. 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
```

20.4.5. 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
```

20.4.6. 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`.

20.4.7. 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
```

20.4.8. 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
```

20.4.9. 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:~$
```

20.4.10. 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:~$
```

20.5. 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 VII.

Scripting 201; job scheduling

21. more scripting

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

21.1. 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
```

21.2. (( ))

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
```

21.3. 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
```

21.4. 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 $animal in
        "lion" | "tiger")
                echo "You better start running fast!"
        ;;
        "cat")
                echo "Let that mouse go..."
        ;;
        "dog")
                echo "Don't worry, give it a cookie."
        ;;
        "chicken" | "goose" | "duck" )
                echo "Eggs for breakfast!"
        ;;
        "liger")
                echo "Approach and say 'Ah you big fluffy kitty...'."
        ;;
        "babelfish")
                echo "Did it fall out your ear ?"
        ;;
        *)
                echo "You discovered an unknown animal, name it!"
        ;;
esac
[student@linux ~]$
```

21.5. 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
```

21.6. 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.

21.7. 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.

22. background jobs

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

22.1. background processes

22.1.1. jobs

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

```
root@linux ~# jobs
root@linux ~#
```

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

22.1.2. control-Z

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

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

```
[student@linux ~]$ vi procdemo.txt

[5]+  Stopped                 vim procdemo.txt
[student@linux ~]$ jobs
[5]+  Stopped                 vim procdemo.txt
```

22.1.3. & ampersand

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

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

22.1.4. jobs -p

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

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

22.1.5. fg

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

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

22.1.6. bg

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

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

```
[student@linux ~]$ jobs
[student@linux ~]$ sleep 5000 &
[1] 6702
[student@linux ~]$ sleep 3000

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

22.2. practice : background processes

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

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

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

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

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

6. Display all `jobs` in background.

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

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

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

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

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

22.3. solution : background processes

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

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

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

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

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

```
[student@linux ~]$ jobs
[1]+  Stopped                 vim text.txt
```

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

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

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

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

6. Display all `jobs` in background.

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

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

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

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

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

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

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

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

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

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

```
[student@linux ~]$ echo $$ $PPID
4224 4223
```

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

```
[student@linux ~]$ bash -c "echo $$ $PPID"
4224 4223
```

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

```
[student@linux ~]$ bash -c 'echo $$ $PPID'
5059 4224
```

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

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

23. scheduling

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

Linux administrators use the `at` to schedule one time jobs. Recurring jobs are better scheduled with `cron`. The next two sections will discuss both tools.

23.1. one time jobs with at

23.1.1. at

Simple scheduling can be done with the `at` command. This screenshot shows the scheduling of the date command at 22:01 and the sleep command at 22:03.

```
root@linux:~# at 22:01
at> date
at> <EOT>
job 1 at Wed Aug  1 22:01:00 2007
root@linux:~# at 22:03
at> sleep 10
at> <EOT>
job 2 at Wed Aug  1 22:03:00 2007
root@linux:~#
```

In real life you will hopefully be scheduling more useful commands ;-)

23.1.2. atq

It is easy to check when jobs are scheduled with the `atq` or `at -l` commands.

```
root@linux:~# atq
1       Wed Aug  1 22:01:00 2007 a root
2       Wed Aug  1 22:03:00 2007 a root
root@linux:~# at -l
1       Wed Aug  1 22:01:00 2007 a root
2       Wed Aug  1 22:03:00 2007 a root
root@linux:~#
```

The at command understands English words like tomorrow and teatime to schedule commands the next day and at four in the afternoon.

```
root@linux:~# at 10:05 tomorrow
at> sleep 100
at> <EOT>
job 5 at Thu Aug  2 10:05:00 2007
root@linux:~# at teatime tomorrow
at> tea
at> <EOT>
job 6 at Thu Aug  2 16:00:00 2007
root@linux:~# atq
6       Thu Aug  2 16:00:00 2007 a root
5       Thu Aug  2 10:05:00 2007 a root
root@linux:~#
```

23.1.3. atrm

Jobs in the at queue can be removed with `atrm`.

```
root@linux:~# atq
6       Thu Aug  2 16:00:00 2007 a root
5       Thu Aug  2 10:05:00 2007 a root
root@linux:~# atrm 5
root@linux:~# atq
6       Thu Aug  2 16:00:00 2007 a root
root@linux:~#
```

23.1.4. at.allow and at.deny

You can also use the `/etc/at.allow` and `/etc/at.deny` files to manage who can schedule jobs with at.

The `/etc/at.allow` file can contain a list of users that are allowed to schedule `at` jobs. When `/etc/at.allow` does not exist, then everyone can use `at` unless their username is listed in `/etc/at.deny`.

If none of these files exist, then everyone can use `at`.

23.2. cron

23.2.1. crontab file

The `crontab(1)` command can be used to maintain the `crontab(5)` file. Each user can have their own crontab file to schedule jobs at a specific time. This time can be specified with five fields in this order: minute, hour, day of the month, month and day of the week. If a field contains an asterisk (*), then this means all values of that field.

The following example means : run script42 eight minutes after two, every day of the month, every month and every day of the week.

```
8 14 * * * script42
```

Run script8472 every month on the first of the month at 25 past midnight.

```
25 0 1 * * script8472
```

Run this script33 every two minutes on Sunday (both 0 and 7 refer to Sunday).

```
*/2 * * * 0
```

Instead of these five fields, you can also type one of these: @reboot, @yearly or @annually, @monthly, @weekly, @daily or @midnight, and @hourly.

23.2.2. crontab command

Users should not edit the crontab file directly, instead they should type `crontab -e` which will use the editor defined in the EDITOR or VISUAL environment variable. Users can display their cron table with `crontab -l`.

23.2.3. cron.allow and cron.deny

The `cron daemon crond` is reading the cron tables, taking into account the `/etc/cron.allow` and `/etc/cron.deny` files.

These files work in the same way as `at.allow` and `at.deny`. When the `cron.allow` file exists, then your username has to be in it, otherwise you cannot use `cron`. When the `cron.allow` file does not exists, then your username cannot be in the `cron.deny` file if you want to use `cron`.

23.2.4. /etc/crontab

The `/etc/crontab` file contains entries for when to run hourly/daily/weekly/monthly tasks. It will look similar to this output.

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

20 3 * * *        root    run-parts --report /etc/cron.daily
40 3 * * 7        root    run-parts --report /etc/cron.weekly
55 3 1 * *        root    run-parts --report /etc/cron.monthly
```

23.2.5. /etc/cron.*

The directories shown in the next screenshot contain the tasks that are run at the times scheduled in `/etc/crontab`. The `/etc/cron.d` directory is for special cases, to schedule jobs that require finer control than hourly/daily/weekly/monthly.

```
student@linux:~$ ls -ld /etc/cron.*
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.d
drwxr-xr-x 2 root root 4096 2008-04-19 15:04 /etc/cron.daily
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.hourly
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.monthly
drwxr-xr-x 2 root root 4096 2008-04-11 09:14 /etc/cron.weekly
```

23.2.6. /etc/cron.*

Note that Red Hat uses `anacron` to schedule daily, weekly and monthly cron jobs.

```
root@linux:/etc# cat anacrontab
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
```

```
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1       5       cron.daily              nice run-parts /etc/cron.daily
7       25      cron.weekly             nice run-parts /etc/cron.weekly
@monthly 45     cron.monthly            nice run-parts /etc/cron.monthly
root@linux:/etc#
```

23.3. practice : scheduling

1. Schedule two jobs with `at`, display the `at queue` and remove a job.

2. As normal user, use `crontab -e` to schedule a script to run every four minutes.

3. As root, display the `crontab` file of your normal user.

4. As the normal user again, remove your `crontab` file.

5. Take a look at the `cron` files and directories in `/etc` and understand them. What is the `run-parts` command doing ?

23.4. solution : scheduling

1. Schedule two jobs with `at`, display the `at queue` and remove a job.

```
root@linux ~# at 9pm today
at> echo go to bed >> /root/todo.txt
at> <EOT>
job 1 at 2010-11-14 21:00
root@linux ~# at 17h31 today
at> echo go to lunch >> /root/todo.txt
at> <EOT>
job 2 at 2010-11-14 17:31
root@linux ~# atq
2   2010-11-14 17:31 a root
1   2010-11-14 21:00 a root
root@linux ~# atrm 1
root@linux ~# atq
2   2010-11-14 17:31 a root
root@linux ~# date
Sun Nov 14 17:31:01 CET 2010
root@linux ~# cat /root/todo.txt
go to lunch
```

2. As normal user, use `crontab -e` to schedule a script to run every four minutes.

```
student@linux ~$ crontab -e
no crontab for paul - using an empty one
crontab: installing new crontab
```

3. As root, display the `crontab` file of your normal user.

```
root@linux ~# crontab -l -u paul
*/4 * * * * echo `date` >> /home/paul/crontest.txt
```

4. As the normal user again, remove your `crontab` file.

```
student@linux ~$ crontab -r
student@linux ~$ crontab -l
no crontab for paul
```

5. Take a look at the `cron` files and directories in `/etc` and understand them. What is the `run-parts` command doing ?

```
run-parts runs a script in a directory
```

Part VIII.

SSH; Docker

24. ssh client and server

(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 **secure shell** or `ssh` is a collection of tools using a secure protocol for communications with remote Linux computers.

This chapter gives an overview of the most common commands related to the use of the `sshd` server and the `ssh` client.

24.1. about ssh

Secure SHell (ssh) is the common way to securely connect to a remote system, e.g. a server in a datacenter, and open a terminal to execute commands. It is considered an essential tool for managing Linux systems, so it is included by default even on minimal installations of most distributions. Server-installations of Linux have the `openssh-server` package installed and enabled.

Even some popular services, like Github, also use SSH for communication. Synchronizing code between a local source code repository and a remote repository on github.com is only allowed over SSH.

24.1.1. secure shell in general

The **SSH protocol** is secure in two ways. Firstly the connection is *authenticated* both ways, and secondly the connection is *encrypted*.

An SSH connection always starts with a cryptographic *handshake*. The *authentication* takes place (using user id/password or public/private keys) and communication can begin over the encrypted connection. When both sides are accepted as trustworthy parties, it is followed by *encryption* of the transport layer using a symmetric cypher. The latter is exchanged between the two parties directly after authenticating. In other words, the tunnel is already encrypted before you start typing anything.

The *SSH protocol* will remember the servers it handshaked with, and warn you in case something suspicious happened, e.g. when the signature or *fingerprint* of the server changed. This is a *'mitm'*-attack prevention.

The `openssh` package is maintained by the *OpenBSD* people and is distributed with a lot of operating systems, including Linux.

Older protocols like `telnet`, `rlogin` and `rsh` can also be used to remotely connect to your servers. However, these protocols do not encrypt the login session, which means confidential information about the server, including your user id and password, can be sniffed by tools like `wireshark` or `tcpdump`. Consequently, they are no longer installed by default on most Linux distributions and you should **never** use them in a production environment. To securely connect to your servers, use `ssh`.

24.1.2. public and private keys

The ssh protocol uses the well known system of *public and private keys*. The below explanation is succinct, more information about public key cryptography can be found on wikipedia.

Imagine Alice and Bob, two people that like to communicate with each other. Using *public and private keys* they can communicate with *confidentiality* and with *authentication*.

Alice and Bob both need to generate a key pair, each consisting of two cryptographic keys. A key pair has as property that a message that is encrypted with one key can only be decrypted with the other key. This concept is also called *asymmetric encryption*. One key is kept *private* and the other is made *public*.

When Alice wants to send an encrypted message to Bob, she uses the *public key* of Bob. Since Bob should be the only person to have his *private key*, Alice is certain that Bob is the only person that can read the encrypted message.

A *digital signature* is based on the same principle. Alice signs a message with her *private key*, which usually means that a *hash* of the message (i.e. a unique number that represents the message) is encrypted with her *private key*. When Bob wants to verify that the message came from Alice, Bob uses the *public key* of Alice to decrypt the signature. If the decrypted hash matches the hash of the received message, Bob can be certain the message came from Alice and was not altered during transfer.

24.1.3. public-key cryptosystems

This chapter does not explain the technical implementation of cryptographic algorithms, it only explains how to use the ssh tools some common key exchange types. More information about these algorithms can be found here:

- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/EdDSA
- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

24.1.4. ssh remote login: autentication fingerprint

The following screenshot shows how to use `ssh` to log on to a remote computer running Linux. The local user is named `paul` and he is logging on as user `admin42` on the remote system.

```
paul@linux:~$ ssh admin42@192.168.1.30
The authenticity of host '192.168.1.30 (192.168.1.30)' can't be established.
RSA key fingerprint is b5:fb:3c:53:50:b4:ab:81:f3:cd:2e:bb:ba:44:d3:75.
Are you sure you want to continue connecting (yes/no)?
```

As you can see, the user `paul` is presented with an `rsa` authentication fingerprint from the remote system. The user can accepts this by typing `yes`. We will see later that an entry will be added to the `~/.ssh/known_hosts` file.

```
paul@linux:~$ ssh admin42@192.168.1.30
The authenticity of host '192.168.1.30 (192.168.1.30)' can't be established.
RSA key fingerprint is b5:fb:3c:53:50:b4:ab:81:f3:cd:2e:bb:ba:44:d3:75.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.30' (RSA) to the list of known hosts.
admin42@192.168.1.30's password:
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-26-generic-pae i686)

```

```
    * Documentation:  https://help.ubuntu.com/

1 package can be updated.
0 updates are security updates.

Last login: Wed Jun  6 19:25:57 2024 from 172.28.0.131
admin42@ubuserver:~$
```

The user can get log out of the remote server by typing `exit` or by using `Ctrl-d`.

```
admin42@ubuserver:~$ exit
logout
Connection to 192.168.1.30 closed.
student@linux:~$
```

The fingerprint guarantees server-side authentication. If the user would connect a second time to this server, the authentication question will no longer be asked (as the fingerprint is retained in the `known_hosts` file). If for any reason the fingerprint of the server does **not** match the one in the `known_hosts` file, the user will be warned and the connection will be refused.

```
paul@linux:~$ ssh admin42@192.168.1.30
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
6e:45:f9:a8:af:38:3d:a1:a5:c7:76:1d:02:f8:77:00.
Please contact your system administrator.
Add correct host key in /home/paul/.ssh/known_hosts to get rid of this
 ↪  message.
Offending ECDSA key in /home/paul/.ssh/known_hosts:14
Host key for 192.168.1.30 has changed and you have requested strict checking.
Host key verification failed.
```

24.1.5. ssh beyond remote login

Next to 'simply' allowing to log in on a remote node, `ssh` also allows for remote exection of commands, and transferring files.

24.1.5.1. executing a command in remote

This screenshot shows how to execute the `pwd` command on the remote server. There is no need to `exit` the server manually.

```
paul@linux:~$ ssh admin42@192.168.1.30 pwd
admin42@192.168.1.30's password:
/home/admin42
paul@linux:~$
```

24.1.5.2. scp

The `scp` command works just like `cp`, but allows the source and destination of the copy to be behind `ssh`. Here is an example where we copy the `/etc/hosts` file from the remote server to the home directory of user paul.

```
pau;@linux:~$ scp admin42@192.168.1.30:/etc/hosts /home/paul/serverhosts
admin42@192.168.1.30's password:
hosts                                        100%  809     0.8KB/s   00:00
```

Here is an example of the reverse, copying a local file to a remote server.

```
student@linux:~$ scp ~/serverhosts admin42@192.168.1.30:/etc/hosts.new
admin42@192.168.1.30's password:
serverhosts                                  100%  809     0.8KB/s   00:00
```

Remark that the `scp` command *used to be* based on the *SCP protocol*. However, this protocol is now considered to be *insecure*. Fortunately, this does not mean that you can't use the `scp` command anymore, since it nowadays uses the actually secure *SFTP protocol* (Secure File Transfer Protocol) to copy files between hosts.

24.2. setting up passwordless ssh

Above, the server-side authentication based on a fingerprint was explained. Client-side authentication is by default based on a user/password combination. However: we could use a pub/priv keypair to take care of this authentication - avoiding to type your password every time you want to log in.

To set up passwordless ssh (client-side) authentication through public/private keys, use `ssh-keygen` to generate a key pair without a passphrase, and then copy your public key to the destination server.

Let's do this step by step. In the example that follows, we will set up ssh without password between Alice and Bob. Alice is using Ubuntu on her laptoph, and has as an account on an Enterprise Linux server named Bob. Server Bob wants to give Alice access using ssh and the public and private key system. This means that even if the password of Alice is changed on this server Bob, Alice will still have access.

24.2.1. generate with ssh-keygen

The example below shows how Alice uses `ssh-keygen` to generate an RSA key pair. Alice does not enter a passphrase[1].

```
[alice@linux ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/alice/.ssh/id_rsa):
Created directory '/home/alice/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alice/.ssh/id_rsa.
Your public key has been saved in /home/alice/.ssh/id_rsa.pub.
The key fingerprint is:
9b:ac:ac:56:c2:98:e5:d9:18:c4:2a:51:72:bb:45:eb alice@linux
[alice@linux ~]$
```

[1]It is possible to specify a passphrase when generating the keypair; that passphrase will be used to encrypt the private key part of this file using 128-bit AES.

With the `-t` option, you can specify the type of key to create. The default is `rsa-sha2-512` (i.e. RSA-key with SHA-2 as hash algorithm).

Safe key types include:

- `rsa` (default)
- `ecdsa` (Elliptic Curve Digital Signature Algorithm)
- `ed25519` (Edwards-curve Digital Signature Algorithm)

Key types to avoid include:

- `dsa` (Digital Signature Algorithm) is considered weak and should not be used. In SSH, this keytype is called `ssh-dss` and has been deprecated as of August 2015 - see https://www.openssh.com/txt/release-7.0 .
- `ssh-rsa` uses the SHA-1 hash algorithm, which is considered weak. The latter has been deprecated as af August 2021 - see https://www.openssh.com/txt/release-8.7 .

It an older server is still using these keys, you should consider updating the keypairs. A workaround, e.g. to use the deprecated key one more time be able to log in to work on the updates, could be found on https://www.openssh.com/legacy.html . This is further elaborated in the 'Troubleshooting' section below.

24.2.2. id_rsa and id_rsa.pub

The `ssh-keygen` command generate two keys in a hidden folder `.ssh`. The public key is named `~/.ssh/id_rsa.pub`. The private key is named `~/.ssh/id_rsa`.

```
[alice@linux ~]$ ls -l .ssh/id*
total 16
-rw------- 1 alice alice 1671 May  1 07:38 id_rsa
-rw-r--r-- 1 alice alice  393 May  1 07:38 id_rsa.pub
```

The files will be named `id_ecdsa` and `id_ecdsa.pub` when using `ecdsa` instead of `rsa`.

24.2.3. ~/.ssh

In general, the `.ssh` directory is used to store the user's `ssh` related files. If it doesn't exist, the hidden `.ssh` directory will be created automatically when generating the keypair. If you create the `.ssh` directory manually, then you need to chmod 700 it! Otherwise ssh will refuse to use the keys (world readable private keys are not secure!).

As you can see, the `.ssh` directory is secure in Alice's home directory on her laptop.

```
[alice@linux ~]$ ls -ld .ssh
drwx------ 2 alice alice 4096 May  1 07:38 .ssh
```

Server Bob decides to manually create the `.ssh` directory, so he needs to manually secure it.

```
bob@linux:~$ mkdir .ssh
bob@linux:~$ ls -ld .ssh
drwxr-xr-x 2 bob bob 4096 2024-05-14 16:53 .ssh
bob@linux:~$ chmod 700 .ssh/
```

Next to storing the keypair(s) generated by a user, it can also contain other files. Most notably is the file `known_hosts`, containing an entry for each fingerprint that has been accepted before when connecting to a new server.

24.2.4. .ssh/authorized_keys

In your `~/.ssh` directory, you can create a file called `authorized_keys`. This file can contain one or more public keys from people you trust. Those trusted people can use their private keys to prove their identity and gain access to your account via ssh (without password). The example shows Bob's authorized_keys file containing the public key of Alice.

```
bob@linux:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEApCQ9xzyLzJes1sR+hPyqW2vyzt1D4zTLqk\
MDWBR4mMFuUZD/O583I3Lg/Q+JIq0RSksNzaL/BNLDou1jMpBe2Dmf/u22u4KmqlJBfDhe\
yTmGSBzeNYCYRSMq78CT9l9a+y6x/shucwhaILsy8A2XfJ9VCggkVtu7XlWFDL2cum08/0\
mRFwVrfc/uPsAn5XkkTscl4g21mQbnp9wJC40pGSJXXMuFOk8MgCb5ieSnpKFniAKM+tEo\
/vjDGSi3F/bxu691jscrU0VUdIoOSo98HUfEf7jKBRikxGAC7I4HLa+/zX73OIvRFAb2hv\
tUhn6RHrBtUJUjbSGiYeFTLDfcTQ== alice@linux
```

24.2.5. copy the public key to the other computer

But wait a minute? How did this key get onto server Bob? To copy the public key from Alice's laptop tot server Bob, Alice decided to use `scp`.

```
[alice@linux .ssh]$ scp id_rsa.pub bob@192.168.48.92:~/.ssh/authorized_keys
bob@192.168.48.92's password:
id_rsa.pub                                    100%  393     0.4KB/s   00:00
```

Be careful when copying a second key! Do not overwrite the first key, instead append the key to the same `~/.ssh/authorized_keys` file!

```
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Alice could also have used `ssh-copy-id` like in this example.

```
ssh-copy-id -i .ssh/id_rsa.pub bob@192.168.48.92
```

24.2.6. passwordless ssh

Alice can now use ssh to connect passwordless to server Bob. In combination with `ssh`'s capability to execute commands on the remote host, this can be useful in pipes across different machines.

```
[alice@linux ~]$ ssh bob@192.168.48.92 "ls -l .ssh"
total 4
-rw-r--r-- 1 bob bob 393 2024-05-14 17:03 authorized_keys
[alice@linux ~]$
```

24.3. ssh-agent

When generating keys with `ssh-keygen`, you have the option to enter a passphrase to protect access to the keys by encrypting them with a symmetric cipher. To avoid having to type this passphrase every time, you can add the key to `ssh-agent` using `ssh-add`.

Most Linux distributions will start the `ssh-agent` automatically when you log on.

```
paul@linux~$ ps -ef | grep ssh-agent
paul     2405  2365  0 08:13 ?        00:00:00 /usr/bin/ssh-agent ...
```

If the `ssh-agent` is not running, you can start it with `eval $(ssh-agent)`.

```
student@debian:~$ ps -ef | grep ssh-agent
student      1608     1393  0 14:52 pts/0    00:00:00 grep ssh-agent
student@debian:~$ eval $(ssh-agent)
Agent pid 1610
student@debian:~$ ps -ef | grep ssh-agent
student      1610        1  0 14:52 ?        00:00:00 ssh-agent
student      1612     1393  0 14:52 pts/0    00:00:00 grep ssh-agent
```

This screenshot shows how to use `ssh-add` with option `-L` to list the keys that are currently added to the `ssh-agent`, and to add a key (without arguments):

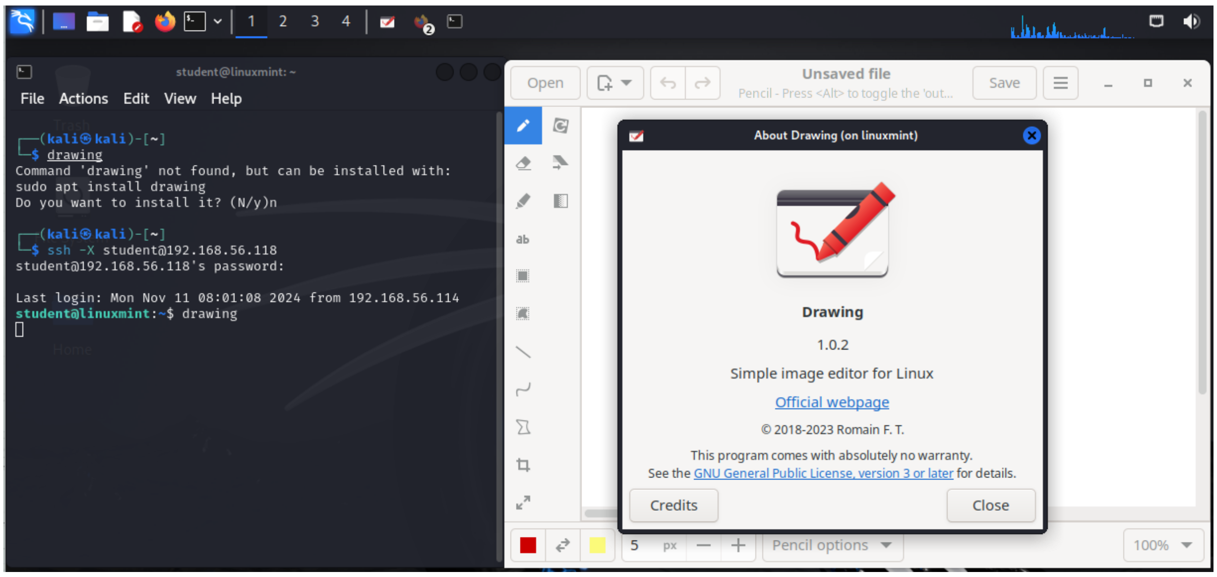
```
student@debian:~$ ssh-add -L
The agent has no identities.
student@debian:~$ ssh-add
Identity added: /home/student/.ssh/id_rsa (student@debian)
student@debian:~$ ssh-add -L
ssh-rsa AAAAB3NzaC1yc2EAAAAD......MeoDHPqR5/yUsCO6MzVOCaZpf8Toc=
 ↪  student@debian
```

All keys can be removed from the `ssh-agent` with `ssh-add -D`.

24.4. X forwarding via ssh

Another popular feature of `ssh` is called *X11 forwarding*. *X11* is the foundation of graphical user interfaces on Linux and Unix systems, so *X11 forwarding* allows you to run graphical applications on a remote computer and have them displayed on your local computer. On the server side, the `X11Forwarding` option must be set to `yes` in the `/etc/ssh/sshd_config` file. On the client side, add option `-X` to the `ssh` command.

Below an example of X forwarding between a Kali Linux VM (client) and a Linux Mint VM (server), both with a graphical desktop. The default user `kali` is logged in and verifies that the app `drawing` is not installed. Next, they log in to the Linux Mint VM as user `student` and start `drawing`. The application will run on the remote computer from `mint`, but will be displayed within the Kali VM, as shown in the screenshot.



```
┌──(kali㉿kali)-[~]
└─$ drawing
Command 'drawing' not found, but can be installed with:
sudo apt install drawing
Do you want to install it? (N/y)n

┌──(kali㉿kali)-[~]
└─$ ssh -X student@192.168.56.118
student@192.168.56.118's password:

Last login: Mon Nov 11 08:01:08 2024 from 192.168.56.114
student@linuxmint:~$ drawing
```

About Drawing (on linuxmint)

Drawing

1.0.2

Simple image editor for Linux

Official webpage

© 2018-2023 Romain F. T.

This program comes with absolutely no warranty.
See the GNU General Public License, version 3 or later for details.

Figure 24.1.: SSH X Forwarding.

24.5. configuration in /etc/ssh/

Configuration of *SSH* client and server is done in the `/etc/ssh` directory. In the next sections we will discuss most of the files found in `/etc/ssh/`.

24.5.1. sshd

The ssh server is provided by the `openssh-server` package.

```
root@linux~# dpkg -l openssh-server | tail -1
ii  openssh-server   1:5.9p1-5ubuntu1    secure shell (SSH) server,…
```

On Debian based distributions, the service is called `ssh`. On Enterprise Linux based distributions, the service is called `sshd`.

Example on EL:

```
[student@el ~]$ systemctl status sshd
● sshd.service - OpenSSH server daemon
     Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; preset:
↪  enabled)
     Active: active (running) since Mon 2024-11-11 10:03:58 UTC; 3h 24min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 668 (sshd)
      Tasks: 1 (limit: 11128)
     Memory: 6.5M
        CPU: 208ms
     CGroup: /system.slice/sshd.service
             └─668 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

On Debian:

```
vagrant@debian:~$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset:
↪  enabled)
     Active: active (running) since Mon 2024-11-11 10:04:31 UTC; 3h 22min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 577 ExecStartPre=/usr/sbin/sshd -t (code=exited,
↪  status=0/SUCCESS)
   Main PID: 611 (sshd)
      Tasks: 1 (limit: 2304)
     Memory: 7.6M
        CPU: 121ms
     CGroup: /system.slice/ssh.service
             └─611 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

The standard port for ssh is 22. You can change this in the `/etc/ssh/sshd_config` file.

```
[vagrant@el ~]$ sudo ss -tlnp
State   Recv-Q  Send-Q  Local Address:Port  Peer Address:Port  Process
LISTEN  0       4096          0.0.0.0:111        0.0.0.0:*
↪  users:(("rpcbind",pid=539,fd=4),("systemd",pid=1,fd=31))
LISTEN  0       128           0.0.0.0:22         0.0.0.0:*
↪  users:(("sshd",pid=668,fd=3))
```

```
LISTEN  0       4096               [::]:111            [::]:*
  ↪  users:(("rpcbind",pid=539,fd=6),("systemd",pid=1,fd=34))
LISTEN  0       128                [::]:22             [::]:*
  ↪  users:(("sshd",pid=668,fd=4))
```

24.5.2. sshd keys

The public keys used by the sshd server for fingerprints are located in `/etc/ssh` and are world readable. The private keys are only readable by root.

```
root@linux~# ls -l /etc/ssh/ssh_host_*
-rw------- 1 root root  668 Jun  7  2024 /etc/ssh/ssh_host_ecdsa_key
-rw-r--r-- 1 root root  598 Jun  7  2024 /etc/ssh/ssh_host_ecdsa_key.pub
-rw------- 1 root root 1679 Jun  7  2024 /etc/ssh/ssh_host_rsa_key
-rw-r--r-- 1 root root  390 Jun  7  2024 /etc/ssh/ssh_host_rsa_key.pub
```

24.6. troubleshooting ssh

Troubleshooting SSH can be a hard issue: as security standards have evolved through time, and even the SSH protocol has evolved from version 1 to 2, you might encounter compatibility issues between an SSH client and server.

Use `ssh -v` to get debug information about the ssh connection attempt.

```
student@linux:~$ ssh -v paul@192.168.1.192
OpenSSH_4.3p2 Debian-8ubuntu1, OpenSSL 0.9.8c 05 Sep 2006
debug1: Reading configuration data /home/paul/.ssh/config
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 192.168.1.192 [192.168.1.192] port 22.
debug1: Connection established.
debug1: identity file /home/paul/.ssh/identity type -1
debug1: identity file /home/paul/.ssh/id_rsa type 1
debug1: identity file /home/paul/.ssh/id_dsa type -1
debug1: Remote protocol version 1.99, remote software version OpenSSH_3
debug1: match: OpenSSH_3.9p1 pat OpenSSH_3.*
debug1: Enabling compatibility mode for protocol 2.0
...
```

24.6.1. ssh protocol versions

The `ssh` protocol has two versions, 1 and 2, that are incompatible. Version 1 is considered insecure since it contains some known vulnerabilities. If you encounter an installation that still uses protocol version version 1, you should endeavour to update it as soon as possible. In current installations, version 1 should no longer be supported. It was removed in 2017 with the release of OpenSSH 7.6.

Check the version of OpenSSH installed with `ssh -V` or with your package manager. For example, on Enterprise Linux:

```
[student@el ssh]$ ssh -V
OpenSSH_8.7p1, OpenSSL 3.0.7 1 Nov 2022
[student@el ssh]$ dnf list installed openssh*
Installed Packages
openssh.x86_64            8.7p1-38.el9_4.4    @baseos
```

```
openssh-clients.x86_64    8.7p1-38.el9_4.4    @baseos
openssh-server.x86_64     8.7p1-38.el9_4.4    @baseos
```

and on Debian:

```
student@debian:~$ ssh -V
OpenSSH_9.2p1 Debian-2+deb12u2, OpenSSL 3.0.11 19 Sep 2023
student@debian:~$ apt list --installed openssh*
Listing... Done
openssh-client/now 1:9.2p1-2+deb12u2 amd64 [installed,upgradable to:
 ↪ 1:9.2p1-2+deb12u3]
openssh-server/now 1:9.2p1-2+deb12u2 amd64 [installed,upgradable to:
 ↪ 1:9.2p1-2+deb12u3]
openssh-sftp-server/now 1:9.2p1-2+deb12u2 amd64 [installed,upgradable to:
 ↪ 1:9.2p1-2+deb12u3]
```

You can try to check the protocol version via `/etc/ssh/ssh_config` for the client side and `/etc/ssh/sshd_config` for the openssh-server daemon. However, on newer installations, the `Protocol` directive is not present in the configuration files. An example on an older installation:

```
student@linux:/etc/ssh$ grep Protocol ssh_config
#   Protocol 2,1
student@linux:/etc/ssh$ grep Protocol sshd_config
Protocol 2
```

24.6.2. dealing with older devices

Unfortunately, some older devices still use insecure key exchange or signing algorithms. For example, the example below is an attempt to connect to a Cisco device (with IOS version 16.09.08, released in 2021) from a Linux system with the OpenSSH client version 9.8:

```
[student@linux ~]$ ssh cisco@172.16.255.254
Unable to negotiate with 172.16.255.254 port 22: no matching key exchange
 ↪ method found. Their offer:
 ↪ diffie-hellman-group-exchange-sha1,diffie-hellman-group14-sha1
```

The Cisco device only uses the SHA-1 hash algorithm for signing messages. This is considered to be insecure, so is no longer supported by the OpenSSH client, at least not by default. Enabling SHA-1 can be done with the following command:

```
[student@linux ~]$ sudo update-crypto-policies --set DEFAULT:SHA1
Setting system policy to DEFAULT:SHA1
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
to fully take place.
```

Trying the connection again yields:

```
[student@linux ~]$ ssh cisco@172.16.255.254
Unable to negotiate with 172.16.255.254 port 22: no matching host key type
 ↪ found. Their offer: ssh-rsa
```

The Cisco device only supports the `ssh-rsa` host key type, which is also turned off on the client side. Turning it on can be done with the following command:

```
[student@linux ~]$ ssh -o HostKeyAlgorithms=+ssh-rsa cisco@172.16.255.254
The authenticity of host '172.16.255.254 (172.16.255.254)' can't be
  ↪ established.
RSA key fingerprint is SHA256:oDUj7I3RQi0FSKSaVX7gz8JfM1wpsYVbyF3ADNJTcAw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.16.255.254' (RSA) to the list of known hosts.
(cisco@172.16.255.254) Password:

[ ... Some output omitted ... ]

Router#
Router#exit
Connection to 172.16.255.254 closed.
[student@linux ~]$
```

Restoring the setting in order to no longer allow SHA-1 can be done with:

```
[student@linux ~]$ sudo update-crypto-policies --set DEFAULT
Setting system policy to DEFAULT
```

24.7. practice: ssh

Lab preparation: Make sure that you have access to *two Linux computers*, e.g. by setting up two VirtualBox VMs either manually or with an automation tool like Vagrant. Ensure that both VMs are attached to the same type of network (preferably Host-only or Internal) and that they have an IP address within the same subnet. One VM will assume the role of the `client`, the other the `server`.

1. On the `server`, check which version of the OpenSSH `server` is installed, which *should* be the case (especially for VMs created with Vagrant). If not, install it.
2. Verify in the ssh configuration files that the `Protocol` setting is absent, or that only protocol version 2 is allowed.
3. Use `ssh` to log on to the `server`, show your current directory and then exit the `server`.
4. Use `scp` to copy a file from the `client` to the `server`.
5. Use `scp` to copy a file from the `server` to the `client`.
6. Use `ssh-keygen` to create a key pair without passphrase on the `client`. Set up passwordless ssh between the `client` and `server`
7. Verify that the permissions on the `client` key files are correct; world readable for the public keys and only root access for the private keys.
8. On the `server`, check which public keys are authorized for the user you are using to log in.
9. Verify that the `ssh-agent` is running.
10. (optional) Protect your keypair with a `passphrase`, then add this key to the `ssh-agent` and test your passwordless ssh to the `server`.
11. (optional, only works when you have a graphical install of Linux) Use ssh with X forwarding to run a graphical on the `server`, but display it on your `client`.

24.8. solution: ssh

In this setup, we have two VMs created with Vagrant:

Hostname	Role	Distro	IP address
`debian`	SSH client	Debian 12	192.168.56.21/24
`el`	SSH server	AlmaLinux 9	192.168.56.11/24

1. On the `server`, check which version of the OpenSSH `server` is installed, which *should* be the case (especially for VMs created with Vagrant). If not, install it.

```
[vagrant@el ~]$ dnf list installed openssh*
Installed Packages
openssh.x86_64                  8.7p1-38.el9_4.4         @baseos
openssh-clients.x86_64          8.7p1-38.el9_4.4         @baseos
openssh-server.x86_64           8.7p1-38.el9_4.4         @baseos
[vagrant@el ~]$ ssh -V
OpenSSH_8.7p1, OpenSSL 3.0.7 1 Nov 2022
[vagrant@el ~]$ systemctl status sshd
● sshd.service - OpenSSH server daemon
    Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; preset: enabled)
    Active: active (running) since Mon 2024-11-11 10:03:58 UTC; 4h 22min ago
    Docs: man:sshd(8)
            man:sshd_config(5)
Main PID: 668 (sshd)
    Tasks: 1 (limit: 11128)
    Memory: 6.5M
        CPU: 208ms
    CGroup: /system.slice/sshd.service
            └─668 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

2. Verify in the ssh configuration files that the `Protocol` setting is absent, or that only protocol version 2 is allowed.

```
[vagrant@el ~]$ sudo find /etc/ssh/ -type f -exec grep -nHi protocol {} \;
[vagrant@el ~]$
```

   No output, so the Protocol setting is absent. Since we have OpenSSH 8.7p1, this is what we can expect, as support for Protocol 1 was dropped in OpenSSH 7.6.

3. Use `ssh` to log on to the `server`, show your current directory and then exit the `server`.

```
vagrant@debian:~$ ssh vagrant@192.168.56.11 pwd
The authenticity of host '192.168.56.11 (192.168.56.11)' can't be established.
ED25519 key fingerprint is SHA256:yw/ZrRXF4zpgbcjCSvM47MyFfg2DfOoiO1tMLVlthGU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.11' (ED25519) to the list of known hosts.
vagrant@192.168.56.11's password:
/home/vagrant
```

4. Use `scp` to copy a file from the `client` to the `server`.

   On the server, the home directory of user `vagrant` is now empty:

```
[vagrant@el ~]$ ls -l
total 0
```

   On the client, we create a file and copy it to the server:

```
vagrant@debian:~$ echo "Hello from Debian" > hello.txt
vagrant@debian:~$ scp hello.txt vagrant@192.168.56.11:/home/vagrant
vagrant@192.168.56.11's password:
hello.txt
```

   On the server, the file is now present:

```
[vagrant@el ~]$ ls -l
total 4
-rw-r--r--. 1 vagrant vagrant 18 Nov 11 14:37 hello.txt
[vagrant@el ~]$ cat hello.txt
Hello from Debian
```

5. Use `scp` to copy a file from the `server` to the `client`.

   Let's remove the file locally and copy it back from the server:

```
vagrant@debian:~$ ls
hello.txt
vagrant@debian:~$ rm hello.txt
vagrant@debian:~$ ls -l
total 0
vagrant@debian:~$ scp vagrant@192.168.56.11:/home/vagrant/hello.txt .
vagrant@192.168.56.11's password:
hello.txt                                  100%   18     7.3KB/s   00:00
vagrant@debian:~$ ls -l
total 4
-rw-r--r-- 1 vagrant vagrant 18 Nov 11 14:36 hello.txt
vagrant@debian:~$ cat hello.txt
Hello from Debian
```

6. Use `ssh-keygen` to create a key pair without passphrase on the `client`. Set up passwordless ssh between the `client` and `server`.

```
vagrant@debian:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:U1EQJVRL/BV8IX9XwVBGIBfAJfdtLkds5VNvM70kFxU vagrant@debian
The key's randomart image is:
+---[RSA 3072]----+
|         .B@BOOE@|
|           =*.==@|
|          . .o X#|
|         .    =*B|
|        S     ..o|
|         .     o |
|                 |
|                 |
|                 |
+----[SHA256]-----+
vagrant@debian:~$ ssh-copy-id -i .ssh/id_rsa.pub vagrant@192.168.56.11
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
  ↪  ".ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
  ↪  filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you
  ↪  are prompted now it is to install the new keys
vagrant@192.168.56.11's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'vagrant@192.168.56.11'"
and check to make sure that only the key(s) you wanted were added.

vagrant@debian:~$ ssh vagrant@192.168.56.11

This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento

Use of this system is acceptance of the OS vendor EULA and License
  ↪  Agreements.
Last login: Mon Nov 11 10:04:49 2024 from 10.0.2.2
[vagrant@el ~]$
```

7. Verify that the permissions on the `client` key files are correct; world readable for the public keys and only root access for the private keys.

```
vagrant@debian:~$ ls -ld .ssh/
drwx------ 2 vagrant root 4096 Nov 11 14:41 .ssh/
vagrant@debian:~$ ls -l .ssh/
total 16
-rw------- 1 vagrant vagrant   89 Oct 11 14:35 authorized_keys
-rw------- 1 vagrant vagrant 2602 Nov 11 14:40 id_rsa
-rw-r--r-- 1 vagrant vagrant  568 Nov 11 14:40 id_rsa.pub
-rw------- 1 vagrant vagrant  978 Nov 11 14:30 known_hosts
```

8. On the `server`, check which public keys are authorized for the user you are using to log in.

```
[vagrant@el ~]$ ls -l .ssh/
total 4
-rw-------. 1 vagrant vagrant 657 Nov 11 14:47 authorized_keys
[vagrant@el ~]$ cat .ssh/authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1N......KVM04 vagrant
ssh-rsa AAAAB3NzaC1yc2EAAAADA......8Toc= vagrant@debian
```

The first key is the one used by Vagrant to log in to the VM, the other one is the key we just added.

9. Verify that the `ssh-agent` is running.

In this case, the SSH agent is *not* running on the client, so we start it manually:

```
vagrant@debian:~$ ps -ef | grep ssh-agent
vagrant     1608    1393  0 14:52 pts/0    00:00:00 grep ssh-agent
vagrant@debian:~$ eval $(ssh-agent)
Agent pid 1610
vagrant@debian:~$ ps -ef | grep ssh-agent
```

```
vagrant     1610       1  0 14:52 ?        00:00:00 ssh-agent
vagrant     1612    1393  0 14:52 pts/0    00:00:00 grep ssh-agent
```

10. (optional) Protect your keypair with a `passphrase`, then add this key to the `ssh-agent` and test your passwordless ssh to the `server`.

    We removed the RSA keypair and created a new ED25519 keypair with a passphrase.

```
vagrant@debian:~$ rm -rf .ssh/id_rsa*
vagrant@debian:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase): [ENTER PASSPHRASE HERE]
Enter same passphrase again: [ENTER PASSPHRASE HERE]
Your identification has been saved in /home/vagrant/.ssh/id_ed25519
Your public key has been saved in /home/vagrant/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:yo7yb0/iQK6TOGazv7DjibRb0pS5XPkiJ5xdxptoVhw vagrant@debian
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|                 |
|       E         |
|    o + .        |
|   +.o *S        |
|  =o=.*.o        |
| = @oB++.        |
|==@.*=oo         |
|=OB*+o+..        |
+----[SHA256]-----+
```

Next, we copy the public key to the server. Remark that we are *not* prompted for the passphrase here! This is only necessary when we want to access the *private key*.

```
vagrant@debian:~$ ssh-copy-id -i .ssh/id_ed25519.pub
 ↪  vagrant@192.168.56.11
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
 ↪  ".ssh/id_ed25519.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
 ↪  filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you
 ↪  are prompted now it is to install the new keys

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'vagrant@192.168.56.11'"
and check to make sure that only the key(s) you wanted were added.
```

We remove all current keys from the agent and add the new key. At this point, we are prompted for the passphrase.

```
vagrant@debian:~$ ssh-add -D
All identities removed.
vagrant@debian:~$ ssh-add -L
The agent has no identities.
vagrant@debian:~$ ssh-add
Enter passphrase for /home/vagrant/.ssh/id_ed25519: [ENTER PASSPHRASE
 ↪  HERE]
Identity added: /home/vagrant/.ssh/id_ed25519 (vagrant@debian)
```

Now we can log in to the server without being prompted for the passphrase:

```
vagrant@debian:~$ ssh vagrant@192.168.56.11
Last login: Mon Nov 11 14:47:53 2024 from 192.168.56.21
[vagrant@el ~]$
```

11. (optional, only works when you have a graphical install of Linux) Use ssh with X forwarding to run a graphical on the `server`, but display it on your `client`.

    See example in the theory section.

25. Docker

(Written by Thomas Parmentier, https://github.com/ThomasParmentier/, with contributions by: Bert Van Vreckem https://github.com/bertvv)

Docker is by far the most popular containerization platform (even the most popular all-round development tool in general according to Stack Overflow). Its main goal is to provide a fast and consistent way to build and deploy applications.

This chapter gives a short overview of the history behind Docker and explains some of its basic concepts.

25.1. why Docker

Docker was first shown as a public demo by Solomon Hykes on PyCon US 2013. He worked for a cloud-provider company using Linux Containers (LXC) to rapidly deploy application servers for its customers. At the time, LXC was a known technology (which is still being used today), although it never got widely adopted due to its rather complex nature.

Docker became an almost universal development tool mainly because of the simplifications it provides (compared to LXC). Some more (recent) history:

- This is a recording of the (noticeably short) first live demonstration of Docker.
- In this presentation Solomon Hykes explains why he felt the need to develop Docker.

25.2. containerization

Essentially, the goal of containerization is the same as virtualization: isolating applications from each other and the hardware. Isolation can have multiple drivers: security, portability, uniform deployment, testing ...

Virtualization is achieved through a hypervisor running different parallel (guest) operating systems called *virtual machines*, while container platforms such as Docker run isolated applications bundled with all components required for them to operate properly: *containers*. These containers do *not* include a full-blown operating system, instead sharing the required features of the host kernel, much like LCX. This means containers are a more light-weight and flexible solution than virtual machines.

Bear in mind that Docker is not the only containerization platform - you can also use LXC, CRI-O or Podman - but it is by far the most popular one.

25.3. concepts

We will now highlight some of the core concepts of Docker, providing general information and demonstrating some of the most useful commands.

More detailed information can be found in the official DockerDocs. The paragraphs below mainly serve as a summary of these docs. It's strongly suggested to try out the steps from the Introduction, as this is the best way to familiarize yourself with the concepts and commands of Docker.

25.3.1. containers

A container is a packaged version of an application, including all its dependencies. It is a small, standalone, deployable unit able to run in semi-isolation on a host system. Docker containers are created using images.

25.3.2. images & layers

Images can be regarded as templates with instructions on how to create containers at run-time. You could call images the static version of containers: the same image can be used to create multiple run-time containers. Most container images are not tied to a container platform and can be used within other framework. Meaning you can f.e. use Docker images to start Podman containers.

Images can specify a wide range of options for the container(s) to be created, ranging from the contained application and its files to the networking and storage options. Most of these options affect the file system within the container, each such a change is called an *image layer*.

Container images are composed of these layers, starting from a base image and adding an additional layer to the image for each configuration change to the file system.Examples of base images are the Python image and the MongoDB image. Possible layers that can be added to these base images to create a new image can be f.e. source code files of an application you want to run and configuration files for its settings.

Note that images are immutable: once layers have been added to an image they can not be changed or removed. You can still use the image as a base for a new image, with additional layers. This is generally done using Dockerfiles.

25.3.3. Dockerfiles

Dockerfiles are text files used to create container images. They contain a fixed set of commands for the Docker image builder, dictating which base image should be used and which additional image layers should be added.

Some of the most common instructions in a Dockerfile include:

- `FROM <image>` - this specifies the base image that the build will extend.
- `WORKDIR <path>` - this instruction specifies the "working directory" or the path in the image where files will be copied and commands will be executed.
- `COPY <host-path> <image-path>` - this instruction tells the builder to copy files from the host and put them into the container image.
- `RUN <command>` - this instruction tells the builder to run the specified command.
- `ENV <name> <value>` - this instruction sets an environment variable that a running container will use.
- `EXPOSE <port-number>` - this instruction sets configuration on the image that indicates a port the image would like to expose.
- `CMD ["<command>", "<arg1>"]` - this instruction sets the default command a container using this image will run.

Example of a Dockerfile:

```
FROM python:3.12
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

```

```
# Copy in the source code
COPY src ./src
EXPOSE 5000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

To read through all of the instructions or go into greater detail, check out the Dockerfile reference.

25.3.4. volumes

Since containers are rather light-weight, they can be deployed and destroyed relatively quickly. Keep in kind that containers are ephemeral by nature, meaning its file system is deleted upon deleting the container. If you want your container data to be persistent, you can use volumes.

Volumes provide persistent storage to docker containers using a sort of link to storage outside the container. Docker can create and manage these volumes, allowing you to link them to containers when required.

Example of linking a persistent volume called `log-data` to the container directory `/logs`. The volume will effectively be mounted at this point in the container file system.

```
docker run -d -p 80:80 -v log-data:/logs docker/welcome-to-docker
```

25.3.5. Docker Compose

Starting and managing multiple containers can become cumbersome if done manually. That's where Docker Compose comes in. Compose allows you to bundle all containers you want to manage as one within a YAML file, allowing you to start or destroy all of them with a single command (`docker compose up`). For example, this command can allow you to spin up a completely containerized LAMP-stack given the correct Docker Compose file.

25.4. practice: Docker

- To get started with Docker: Introduction to Docker
- To get acquainted with the basic concepts: What's Next

A. git

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

This chapter is an introduction to using `git` on the command line. The `git repository` is hosted by `github`, but you are free to choose another server (or create your own).

There are many excellent online tutorials for `git`. This list can save you one Google query:

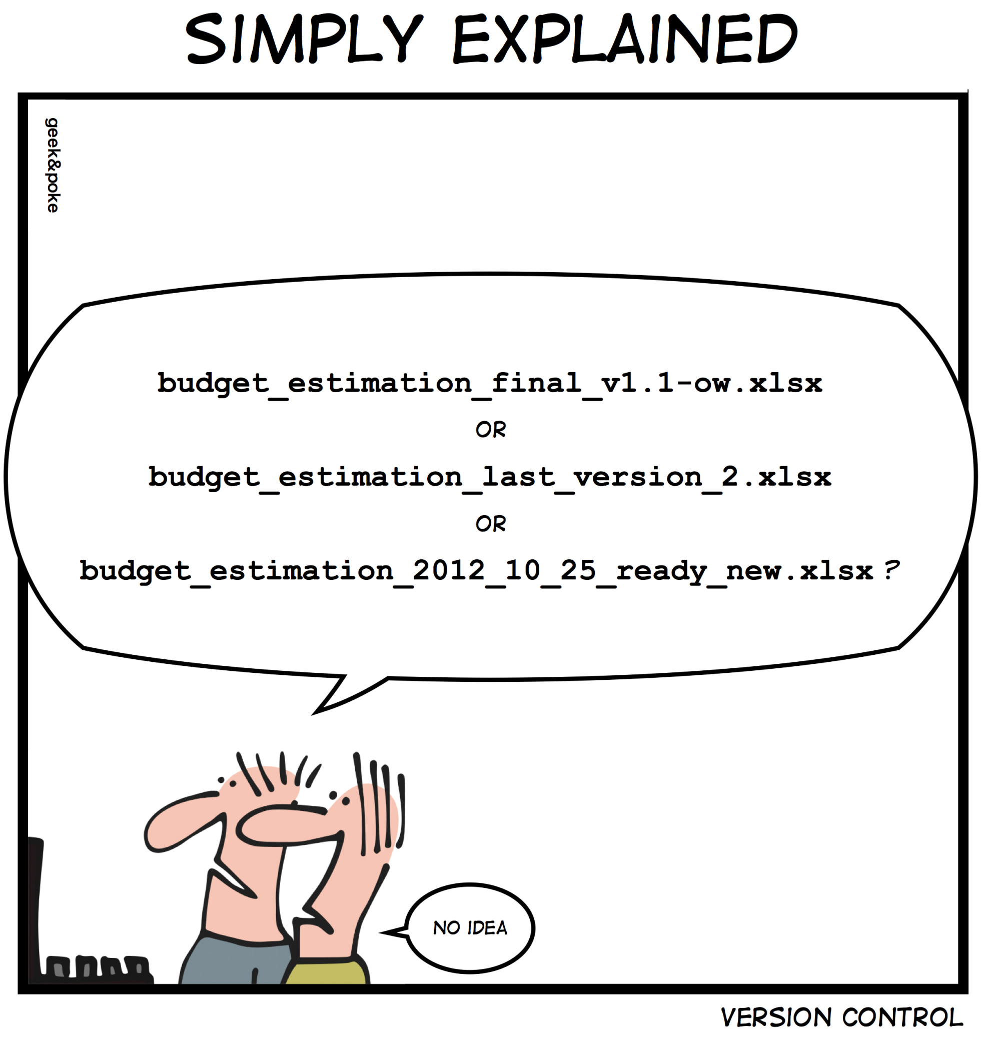
```
http://gitimmersion.com/
http://git-scm.com/book
```

A.1. git

Linus Torvalds created `git` back in 2005 when Bitkeeper changed its license and the Linux kernel developers where no longer able to use it for free.

`git` quickly became popular and is now the most widely used `distributed version control` system in the world.

Geek and Poke demonstrates why we need version control (image property of Geek and Poke CCA 3.0).


SIMPLY EXPLAINED
geek&poke
budget_estimation_final_v1.1-ow.xlsx
OR
budget_estimation_last_version_2.xlsx
OR
budget_estimation_2012_10_25_ready_new.xlsx ?
NO IDEA
VERSION CONTROL

Besides `source code` for software, you can also find German and Icelandic `law` on github (and probably much more by the time you are reading this).

A.2. installing git

We install `git` with `aptitude install git` as seen in this screenshot on Debian 6.

```
root@linux:~# aptitude install git
The following NEW packages will be installed:
  git libcurl3-gnutls{a} liberror-perl{a}
0 packages upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
 ...
Processing triggers for man-db ...
Setting up libcurl3-gnutls (7.21.0-2.1+squeeze2) ...
Setting up liberror-perl (0.17-1) ...
Setting up git (1:1.7.2.5-3) ...
```

A.3. starting a project

First we create a project directory, with a simple file in it.

```
student@linux~$ mkdir project42
student@linux~$ cd project42/
student@linux~/project42$ echo "echo The answer is 42." >> question.sh
```

A.3.1. git init

Then we tell `git` to create an empty git repository in this directory.

```
student@linux~/project42$ ls -la
total 12
drwxrwxr-x  2 paul paul 4096 Dec  8 16:41 .
drwxr-xr-x 46 paul paul 4096 Dec  8 16:41 ..
-rw-rw-r--  1 paul paul   23 Dec  8 16:41 question.sh
student@linux~/project42$ git init
Initialized empty Git repository in /home/paul/project42/.git/
student@linux~/project42$ ls -la
total 16
drwxrwxr-x  3 paul paul 4096 Dec  8 16:44 .
drwxr-xr-x 46 paul paul 4096 Dec  8 16:41 ..
drwxrwxr-x  7 paul paul 4096 Dec  8 16:44 .git
-rw-rw-r--  1 paul paul   23 Dec  8 16:41 question.sh
```

A.3.2. git config

Next we use `git config` to set some global options.

```
student@linux$ git config --global user.name Paul
student@linux$ git config --global user.email "paul.cobbaut@gmail.com"
student@linux$ git config --global core.editor vi
```

We can verify this config in `~/.gitconfig`:

```
student@linux~/project42$ cat ~/.gitconfig
[user]
    name = Paul
    email = paul.cobbaut@gmail.com
[core]
    editor = vi
```

A.3.3. git add

Time now to add file to our project with `git add`, and verify that it is added with `git status`.

```
student@linux~/project42$ git add question.sh
student@linux~/project42$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   question.sh
#
```

The `git status` tells us there is a new file ready to be committed.

A.3.4. git commit

With `git commit` you force git to record all added files (and all changes to those files) permanently.

```
student@linux~/project42$ git commit -m "starting a project"
[master (root-commit) 5c10768] starting a project
 1 file changed, 1 insertion(+)
 create mode 100644 question.sh
student@linux~/project42$ git status
# On branch master
nothing to commit (working directory clean)
```

A.3.5. changing a committed file

The screenshots below show several steps. First we change a file:

```
student@linux~/project42$ git status
# On branch master
nothing to commit (working directory clean)
student@linux~/project42$ vi question.sh
```

Then we verify the status and see that it is modified:

```
student@linux~/project42$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   question.sh
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Next we add it to the git repository.

```
student@linux~/project42$ git add question.sh
student@linux~/project42$ git commit -m "adding a she-bang to the main script"
[master 86b8347] adding a she-bang to the main script
 1 file changed, 1 insertion(+)
student@linux~/project42$ git status
# On branch master
nothing to commit (working directory clean)
```

A.3.6. git log

We can see all our commits again using `git log`.

```
student@linux~/project42$ git log
commit 86b8347192ea025815df7a8e628d99474b41fb6c
Author: Paul <paul.cobbaut@gmail.com>
Date:   Sat Dec 8 17:12:24 2012 +0100

    adding a she-bang to the main script

commit 5c10768f29aecc16161fb197765e0f14383f7bca
Author: Paul <paul.cobbaut@gmail.com>
Date:   Sat Dec 8 17:09:29 2012 +0100

    starting a project
```

The log format can be changed.

```
student@linux~/project42$ git log --pretty=oneline
86b8347192ea025815df7a8e628d99474b41fb6c adding a she-bang to the main script
5c10768f29aecc16161fb197765e0f14383f7bca starting a project
```

The log format can be customized a lot.

```
student@linux~/project42$ git log --pretty=format:"%an: %ar :%s"
Paul: 8 minutes ago :adding a she-bang to the main script
Paul: 11 minutes ago :starting a project
```

A.3.7. git mv

Renaming a file can be done with `mv` followed by a `git remove` and a `git add` of the new filename. But it can be done easier and in one command using `git mv`.

```
student@linux~/project42$ git mv question.sh thequestion.sh
student@linux~/project42$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   renamed:    question.sh -> thequestion.sh
#
student@linux~/project42$ git commit -m "improved naming scheme"
[master 69b2c8b] improved naming scheme
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename question.sh => thequestion.sh (100%)
```

A.4. git branches

Working on the project can be done in one or more `git branches`. Here we create a new branch that will make changes to the script. We will `merge` this branch with the `master branch` when we are sure the script works. (It can be useful to add `git status` commands when practicing).

```
student@linux~/project42$ git branch
* master
student@linux~/project42$ git checkout -b newheader
Switched to a new branch 'newheader'
student@linux~/project42$ vi thequestion.sh
student@linux~/project42$ git add thequestion.sh
student@linux~/project42$ source thequestion.sh
The answer is 42.
```

It seems to work, so we commit in this branch.

```
student@linux~/project42$ git commit -m "adding a new company header"
[newheader 730a22b] adding a new company header
 1 file changed, 4 insertions(+)
student@linux~/project42$ git branch
  master
* newheader
student@linux~/project42$ cat thequestion.sh
#!/bin/bash
#
# copyright linux-training.be
#

echo The answer is 42.
```

Let us go back to the master branch and see what happened there.

```
student@linux~/project42$ git checkout master
Switched to branch 'master'
student@linux~/project42$ cat thequestion.sh
#!/bin/bash
echo The answer is 42.
```

Nothing happened in the master branch, because we worked in another branch.

When we are sure the branch is ready for production, then we merge it into the master branch.

```
student@linux~/project42$ cat thequestion.sh
#!/bin/bash
echo The answer is 42.
student@linux~/project42$ git merge newheader
Updating 69b2c8b..730a22b
Fast-forward
 thequestion.sh |    4 ++++
 1 file changed, 4 insertions(+)
student@linux~/project42$ cat thequestion.sh
#!/bin/bash
#
# copyright linux-training.be
#

echo The answer is 42.
```

The newheader branch can now be deleted.

```
student@linux~/project42$ git branch
* master
  newheader
student@linux~/project42$ git branch -d newheader
Deleted branch newheader (was 730a22b).
student@linux~/project42$ git branch
* master
```

A.5. to be continued...

The `git` story is not finished.

There are many excellent online tutorials for `git`. This list can save you one Google query:

```
http://gitimmersion.com/
http://git-scm.com/book
```

A.6. github.com

Create an account on `github.com`. This website is a frontend for an immense git server with over two and a half million users and almost five million projects (including Fedora, Linux kernel, Android, Ruby on Rails, Wine, X.org, VLC...)

```
https://github.com/signup/free
```

This account is free of charge, we will use it in the examples below.

A.7. add your public key to github

I prefer to use github with a `public key`, so it probably is a good idea that you also upload your public key to github.com.

You can upload your own key via the web interface:

```
https://github.com/settings/ssh
```

Please do not forget to protect your `private key`!

A.8. practice: git

1.Crate local project called `git_practice`.

2.Create a project on `gitlab.com` to host a local project that you have created.

3.The project should have `REAMDE.md` file as well as `TODO.md` file in it.

4.Write in `REAMDE.md` file description of the project and what you think it might be.

5.Initialize your project with `git` command, setup your username, mail and remote server.

6.Use `git push -u origin master` to send project saves to remote host.

7.Verify on `gitlab.com` that the project has been setup and is updated with `REAMDE.md` and `TODO.md`.

8.Add git_hello.sh script that prints hello to username from its current location.

9.Push the script to gitlab repository.

A.9. solution: git

1.Crate local project called `git_practice`.

```
aschapelle@vaio3:~$ mkdir git_practice; cd git_practice
```

2.Create a project on `gitlab.com` to host a local project that you have created.

3.The project should have `REAMDE.md` file as well as `TODO.md` file in it.

```
aschapelle@vaio3:~/git_practice$ touch REAMDE.md TODO.md
```

4.Write in `REAMDE.md` file description of the project and what you think it might be.

```
aschapelle@vaio3:~/git_practice$ echo "This is readme file for git_practice project" > READM
        aschapelle@vaio3:~/git_practice$echo "This is todo file for git_practice project" >
```

5.Initialize your project with `git` command, setup your username, mail and remote server.

```
aschapelle@vaio3:~/git_practice$ git init
        aschapelle@vaio3:~/git_practice$ git config user.name alex.schapelle
      aschapelle@vaio3:~/git_practice$ git config user.mail alex@vaiolabs.com
     aschapelle@vaio3:~/git_practice$ git remote add origin https://gitlab.com/url_to_you
```

6.Use `git push -u origin master` to send project saves to remote host.

```
        aschapelle@vaio3:~/git_practice$ git push -u origin master
```

7.Verify on `gitlab.com` that the project has been setup and is updated with `REAMDE.md` and `TODO.md`.

8.Add git_hello.sh script that prints hello to username from its current location.

```
aschapelle@vaio3:~/git_practice$ git push -u origin master
```

9.Push the script to gitlab repository.

```
        aschapelle@vaio3:~/git_practice$ git push -u origin master
```

B. 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)`.

B.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 B.1.: getting to command mode

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

B.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 B.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

B.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 B.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

B.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 B.4.: undo and repeat

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

B.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 B.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

B.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 B.6.: cut, copy and paste lines

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

B.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 B.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

B.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 B.8.: join two lines

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

B.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 B.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

B.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 B.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).

B.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 B.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`)

B.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 B.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

B.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 B.13.: read files and input

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

B.14. text buffers

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

Table B.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

B.15. multiple files

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

Table B.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

B.16. abbreviations

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

Table B.16.: abbreviations

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

B.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>
```

B.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:~$
```

B.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 ?

B.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.

C. GNU Free Documentation License

License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

C.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

C.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

C.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

C.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.

C.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.

C.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.