# 2223 - Linux - AP4

## Table of contents

In this course and labs, we will have a look at Linux commands. You will learn how to use the terminal. Please find below the table of contents.

## Pre-requisites

### Windows

If you don’t have a Linux OS (shame on you!), you can activate the Windows Subsystem Linux (WSL).

> **ⓘ info**
>
> Ouvrez la fenêtre Fonctionnalités de Windows :
>
> - via la commande Exécuter : optionalfeatures.
> - via les Paramètres > Applications et fonctionnalités > Fonctionnalités facultatives > Plus de fonctionnalités Windows.

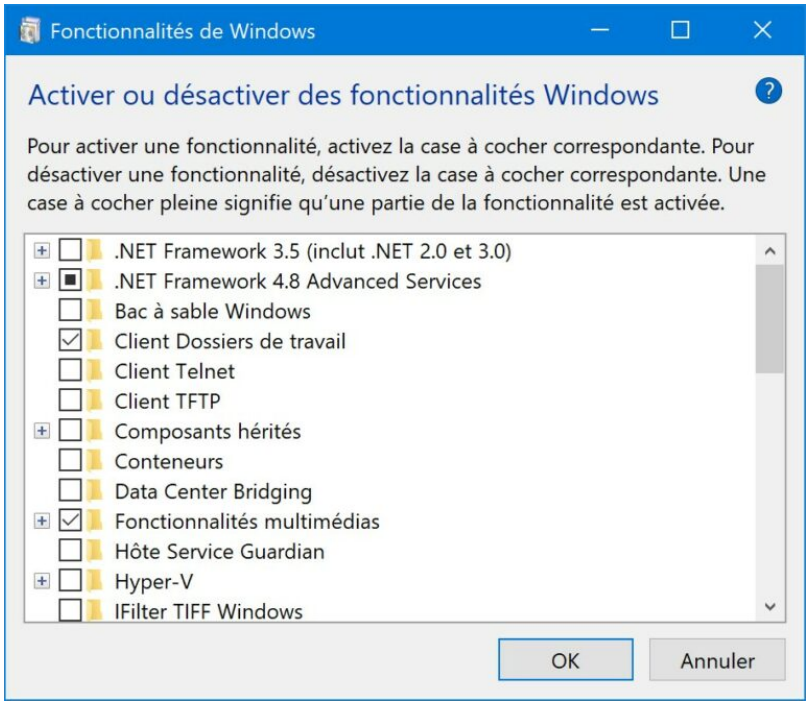


**Fig. 1: Activate Windows Subsystem Linux in the Windows Configuration Panel**

**ⓘ info**

- Cochez la case Sous-système Windows pour Linux puis faites OK.
- Définir WSL2 comme système par défaut
- Dans powershell, exécutez `wsl --set-default-version 2`.
- Installer une distribution Linux
- Soit via le Microsoft Store
- Via powershell `wsl --install ubuntu`

## Linux

Open a terminal and you should be set already!

# Linux distributions and forks

Linux is a very opened operating system. It is based on an open source kernel (which acts as the baseline to all input and output) and then you can add and attached all kinds of packages that will interact with the kernel. Based on that, anyone can create a distribution. Though, those distributions are based on *main* (or *base*) distributions that are actively developed. To name a few : Debian, Ubuntu (which is already a fork of Debian), Fedora, RedHat, Arch... The list goes on and on and on.

Almost one thousand Linux distributions exist. Because of the huge availability of software, distributions have taken a wide variety of forms, including those suitable for use on desktops, servers, laptops, netbooks, mobile phones and tablets, as well as in minimal environments typically for use in embedded systems. For instance, Android is based on a modified Linux kernel. You can find Linux on bank ATMs, airport flight screens or even sometimes when you order your burgers at your favorite fast-food restaurants.

In this course, we will use Linux as a desktop environment with a graphical user interface (GUI) and heavily use the terminal. Here is a list of the most popular, modern, distributions available :

| Project |
| --- |
| MX Linux |
| Mint |
| Manjaro |
| Debian |
| openSUSE |
| Ubuntu |
| Fedora |
| deepin |
| Solus |
| Pop!_OS |
| Zorin |
| Arch |
| KDE neon |
| elementary |
| EndeavourOS |
| Devuan |
| ArcoLinux |
| Lite |
| Kubuntu |
| Peppermint |
| Void |
| antiX |
| Slackware |
| Q4OS |
| Ubuntu MATE |
| Lubuntu |
| Mageia |
| PCLinuxOS |
| Garuda |
| Artix |
| Xubuntu |
| SparkyLinux |
| Puppy |
| Gentoo |
| Antergos |
| Linuxfx |
| Feren |
| Bodhi |
| Parrot |
| OpenBSD |

**Fig. 2: Distrowatch - List of 40-ish available distributions**

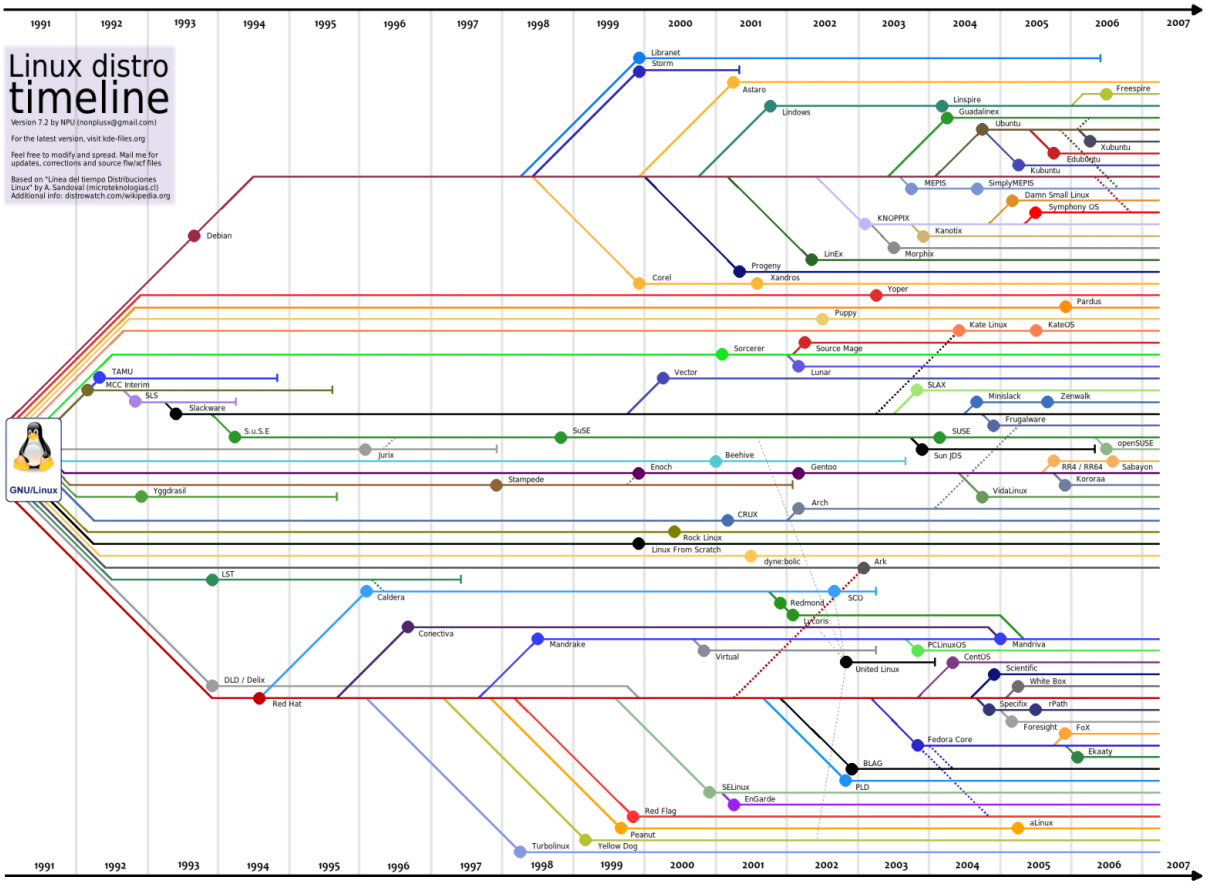


**Fig. 3: Since its release in 1991, forks have been everywhere! And it is difficult to follow**

# Labs

## Shells

There are several popular shells or command interpreters for the Unix / Linux systems.

- **sh**: Bourne Shell. The oldest and installed on all Unix-based OS, but poor in functionality compared to other shells.\
- **bash**: Bourne Again Shell. An improved shell, available by default on Linux and Mac OS X\
- **ksh**: Korn Shell. A powerful shell quite present on the proprietary Unix, but also available in free version, compatible with bash (Korn's shell includes Bourne's shell)\
- **csh**: C Shell. A shell using a syntax close to the C language\
- **tcsh**: Tenex C Shell. An improved C Shell\
- **zsh**: Z Shell. Pretty new shell with the best ideas of bash, ksh and tcsh

To exit a terminal type `exit`. It should be straightforward when you are using a graphical interface (you can click on the cross to exit the window) but when you are in remote instance, `exit` is the way to go.

Try this command `echo 'Hello world!'`. It should print `Hello world!` in the terminal. It is the equivalent of the print('Hello world!') in the terminal.

To edit files in the terminal, you can use `nano` or `vim`. `nano` is more simple in its approach than `vim` which requires knowledge on keybindings but has much more advanced features. If they are not present, please refer to the next section to install them.

## Install packages

How you install packages depend on the distributions you are using
Debian and Ubuntu : `apt-get`
Fedora : `dnf`
CentOS and RedHat : `yum`
Arch and Manjaro : `pacman`

Do not underestimate the use of the help command `man`.

> **? Give commands to...**
>
> - On your distribution, use the package manager to install the `python` and `lua` languages as well as an advanced command-line finder `fzf`. If not present, install as well `nano` and `vim` as terminal editors.
> - Install your preferred internet browser (`chrome` does not exist on Linux, it is `chromium`) and then remove it using the command line.
> - Use the package manager to search the software `gimp`.

## System identification

Use the `man` command to get help on commands described below (man command)

**date** : Show the date and time
**uname** : Get system identification
**who** : View the list of connected users
**whoami** : View the user of the current session

> **? Give the commands to...**
>
> - Display the date as jj-mm-aaaa format (use + format option)
> - Display the hour as hh:mm:ss format
> - Identify the operating system
> - Identify the operating system version
> - Identify the hardware platform
> - Identify the identity of the current user logged

## Process management

**command** : Launch a command
**command &** : Launch a command on background
**[ctrl z] followed by "bg"** : Put a command on background
**ps** : View the list of user processes
**kill** : Stop a process
**top** : View the list of all processes

> **? Give the commands to...**
>
> - Display all processes with details using `ps`
> - Launch less command to browse a file
> - Put the process corresponding to the "less" command to background
> - Find the PID (process ID) of this process
> - Stop the "less" process
> - Display all processes with `top` and sort the processes by CPU usage then MEM usage

## Folder structure

`/` : The root directory, represented by a forward slash (/), stores all the directories in Linux
`/boot` : The boot directory contains important files needed by the boot loader. The initial ram file system or initramfs is also stored here along with the kernel.
`/bin` : The /bin directory contains system commands and other executable programs. It is usually a link to `/usr/bin`. The binaries in this directory are available system-wide. `/etc` : The /etc directory contains vital system configuration files such as startup scripts, networking files, user account-related files, etc. You have to edit configuration files in the /etc directory to make any system-wide changes.
`/lib` or `/usr/lib`. A library is a collection of pre-compiled code that executable binaries can use.
`/opt` : The /opt directory contains optional software packages to facilitate better compatibility of certain applications. `/tmp` : Temporary folder. This folder is cleaned at each reboot.
`/srv` : Usually used for web or network services like network folders, ftp, etc.
`/home` : The /home directory stores an individual user's home directory. Each user has its own home directory.
`/proc` : The /proc directory is a pseudo-filesystem containing information about processes and kernel parameters. It is populated with data during boot-up and is cleaned when you shut down your Linux machine.
`/usr` : The /usr directory contains most of the files, libraries, programs, and system utilities.
`/var` : The /var directory is the storage space for system-generated variable files, and it includes logs, caches, and spool files.
`/var/log` : Logs are stored in this specific folder.\

Go to the root directory using `cd /` and list all the directory on a list using `ls -al`. What do you see? What are the differences with the list above?

### Paths

**Absolute path**: /dir/sub_dir_1/sub_dir_2/file
**Relative path form working directory "/dir"**: sub_dir_1/sub_dir_2/file

### Wildcards

`/` system's root directory
`.` working directory
`..` directory one level up
`~` or `$home` login directory
`~otheruser` otheruser's home directory, if otheruser permission access is granted
`*` all files

### Directory management

**pwd** : View the name of the working directory
**cd** : Change working directory
**ls** : View the content of a directory
**mkdir** : Create a directory
**rmdir** : Delete a directory

**? Give the commands to...**

- Go to the home directory
- Find the name of the working directory
- Create a directory named “lab_1”
- Display the content of the working directory
- Display the content of “lab_1” directory
- Change the working directory to “lab_1”
- Display the content of the working directory
- Create a directory named “lab_2”
- Display the detailed content of the working directory
- Display all hidden entries of the working directory
- Display the content of the parent directory of working directory
- Remove the “lab_1” directory

### File management

**touch** : Create an empty file and change file date/time
**mv** : Rename or move a file
**cp** : Copy a file into another file
**rm** : Delete a file

**? Give the commands to...**

- Create an empty file “new_file”
- Rename the “new_file” file to “file_1”
- Copy “file_1” file to “file_2”
- Create a directory named “lab_1” in the working directory
- Move the “file_1” file to “lab_1” directory
- Delete “file_1” “file_2” files and “lab_1” directory

### File content

**cat** : Display the content of a file or merge a list of files and display the result
**more** : Display the content of a file page per page
**less** : Display the content of a file page per page
**head** : Display the beginning of a file
**tail** : Display the end of a file
**wc** : Count the number of words, lines or characters
**diff** : Find differences between two files

**? Give the commands to...**

- Dump the `journalctl` content into a new file with the command `journalctl > journalctl.txt`
- Display the content of `journalctl.txt` file
- Display the content of `journalctl.txt` file page by page
- Display the content of `journalctl.txt` file page by page of 10 lines
- Display the content of `journalctl.txt` file page by page of 10 lines, skipping the first 20 lines
- Display the beginning of `journalctl.txt` file
- Display the end of `journalctl.txt` file
- Count the number of lines of `journalctl.txt` file

## Input, output

### Special devices

*/dev/null* and */dev/zero*

Data written to the `/dev/null` and `/dev/zero` special files is discarded.

Reads from `/dev/null` always return end of file.

Reads from `/dev/zero` always return bytes containing zero (`\0` characters).

*/dev/full*

Writes to the `/dev/full` device fail with an ENOSPC error. This can be used to test how a program handles disk-full errors. Reads from the `/dev/full` device will return \0 characters.

*/dev/random* and */dev/urandom*

Reads from the `/dev/urandom` device returns random bytes using a pseudorandom number generator seeded from the entropy pool. Writes to `/dev/random` or `/dev/urandom` will update the entropy pool with the data written.

At each process created (command or program in execution) are associated with one input and two standard outputs:

**The standard input** : the keyboard
**Standard output** : the terminal screen
**The standard error output** : the terminal screen

The shell allows to modify the default inputs / outputs of a process.

> **Information**
>
> - Redirection of standard input (`command < input_file_name`)
> - Redirection of standard input, keybord is used as input and the key_word is used to stop the input process (`command << key_word`)
> - Redirection of standard output (`command > output_file_name`) or (`command >> output_file_name`)
> - Redirection of error (`command 2> error_file_name`) or (`command 2>> error_file_name`)
> - Redirection of standard output and error (`command > output_file_name 2> error_file_name`) or (`command >> file_name 2> error_file_name`) or (`command > file_name 2>> error_file_name`) or (`command >> file_name 2>> error_file_name`)
> - Redirection of standard output and error (`command > common_file_name 2>&1`) or (`command >> common_file_name 2>&1`)

### Chaining commands

*Sequential chaining*

It's possible to launch several commands (processes) sequentially with the operator ";" (`command_1; command_2;...`)

*Pipelines*

It's possible to launch several commands (processes) sequentially and redirect the output of the previous process to the input of the next process with the operator "|" (`command_1 | command_2 |...`)

*Success and error*

It's possible to launch several commands (processes) sequentially with a condition of success and error. Command_2 is launch if the command_1 finish with success (`command_1 && command_2`) Command_2 is launch if the command_1 finish with error (`command_1 || command_2`)

**? Give the commands to...**

- Display all processes with details page by page of 10 lines
- Record the list of all processes with details to the “process_list” file
- Count the number of lines recorded in the “process_list” file
- Record the count of the number of lines recorded in the “process_list” file to the “number_of_lines” file
- Display the last line of the list of all processes with details
- Record the last line of the list of all processes with details to the “process_list” file
- Record the first and the last line of the list of all processes with details to the “process_list” file

## Find and filters

### System wise

**find**: search for files in a directory hierarchy
**fzf**: a command-line fuzzy finder
**awk**: pattern scanning and processing language
**sed**: stream editor for filtering and transforming text

**? Give the commands to...**

- Find files in your home directory
- Find directories in your home folder, use `$HOME` and `~` as wildcards for your home folder. What are the differences?
- List executables only in `/usr/`
- List directories only (without recursivity) in `/usr/lib`
- Sort the last result by name
- Pipe the result in `fzf` and select a directory, what is happening?
- Find all files in your home folder and filter to display only the `.txt` files
- Pipe the result in `fzf`
- Use a second pipe to open the select `.txt` file in a text editor, you need to use the command `xargs`
- Type the command `fzf` in the current directory, what is happening?

**ⓘ Information**

`fzf` is a **fuzzy finder**, it is not a search tool. It can search/filter within a list. The command that `fzf` is running behind depends on the command passed to the variable `FZF_DEFAULT_COMMAND`

**? Questions...**

- Print the command stored in the variable to your terminal using `echo $FZF_DEFAULT_COMMAND`, what do you see?
- Change the variable to find directories in your home folder (use `export FZF_DEFAULT_COMMAND=<your command>`), then run `fzf` again
- Change the variable to find all files in the current working directory
- Go to `/usr/lib` and run `fzf`
- Run `journalctl | tail -n 10` to display the last 10 lines of the linux journal. What is happening when you do `journalctl | tail -n 10 | awk '{print $1}'`?
- Change `$1` by `$2` then `$3` and so on... What do you see?
- Using the whole journal, try to filter all events occuring from yesterday after 10h00
- Using `df` you can see all filesystems mounted on your system. Try to isolate the name of filesystem which is mounted as root (mounted on `/`)

The command `awk` is a filter information. We will use it to isolate information and compare or condition them to write small functions. Many UNIX utilities generates rows and columns of information. AWK is an excellent tool for processing these rows and columns, and is easier to use AWK than most conventional programming languages. The syntax is `awk '{ action }'`, where action is what you want to do.

**? Give the commands to...**

- Let's start `cd $HOME/.config` and with `ls -al`. What do you see?
- Each column can be isolated using `$X` where X is the column number starting from `1`.
- Using pipe and `ls -al`, isolate the file name and the owner of the file.
- Isolate folders and files using `awk` that have been created before 2022.
- Using `NR` and `NF` (Number Record and Number Fields), print the fifth line and nineth row.
- Using `NR`, print all the lines except the first one.
- Using `NF`, print all the columns except the last one.

## In files

### awk

**ⓘ Information**

- You can as well use `awk`, in files. Output the content of `/etc/passwd` in the terminal and isolate the username. (Hint: you need to change the Field Separator (FS))

In order to quickly filter letters, strings, numbers... It is usually faster to use `grep`.

**? Give the commands to...**

- With the `journalctl.txt` file or the command `journalctl`, find the kernel version of your distribution. Find the occurence `Linux version`
- Using `grep`, filter all the text files (.txt) file in your home folder
- What are the differences with fzf?
- Using `grep`, display all the executables that start with an `a` in `/usr/bin`
- Display all the executables that do NOT contain the letter `e` in `/usr/bin`
- Display all the hidden directories in your home folder

### sed

`sed` is a powerful text stream editor. It can do insertion, deletion, search and replace (substitution).

**? Questions...**

- Create a text file with this text inside :

```
unix is great os. unix is opensource. unix is free os.
learn operating system.
unix linux which one you choose.
unix is easy to learn.unix is a multiuser os. Learn unix. Unix is powerful.
```

- We can find and substitute `unix` by `linux` by doing the command : `sed 's/unix/linux/' NOMDUFICHIER.txt`. The `/` are the delimiter for the strings. What do you see?
- Now to do it globally, we need to add `g` at the very end. It should be `sed 's/unix/linux/g' NOMDUFICHIER.txt`. Are there any `unix` left?
- We can as well delete lines or string. We need to finish the pattern by `d` for delete.
- Try to delete all `unix` occurences in the file.

## System commands

### Memory and disk space

**free**: display information on the memory and how much you have left
**df**: Show information about the file system on which each FILE resides, or all file systems by default.
**du**: Summarize device usage of the set of FILEs, recursively for directories.

> **? Give the commands to...**
>
> - Show how much memory you have left in megabytes and gigabytes
> - Show how much disk space you have in a human-readable format
> - Show how much disk space the `/etc` folder is using
> - Show how much disk space the `/usr/lib` folder is using
> - Do the same as the two last command but using human-readable format
> - Sort the results by size
> - Display only the 10 biggest folders

### Zip and unzip archives

**tar**: Creates and manipulates “archives” which are actually collections of many other files
**zip**: Package and compress (archive) files
**unzip**: List, test and extract compressed files in a ZIP archive

> **? Give the commands to...**
>
> - Create several empty files in a new folder and create a zip file using `tar` and `zip`
> - Inspect the newly created archive using `tar` and `unzip`
> - Finally extract the archive using `tar` and `unzip` and use command-line options to display all the extracting process (verbose)

## Users and groups

### User definition

User’s privileges are defined by a UID (User ID) and a GID (Group ID). Both are set in the `/etc/passwd` file. Groups are defined in the `/etc/group` file.

### UID

The UID `0` has a special role: it is always the root account. The UIDs `1` through `99` are traditionally reserved for special system users.

Some Linux distributions begin UIDs for non-privileged users at `100`. Others, such as Red Hat, begin them at `500`, and still others, such Debian, start them at `1000`. The UID `65534` is commonly reserved for nobody, a user with no system privileges, as opposed to an ordinary. Also, it can be convenient to reserve a block of UIDs for local users, such as 1000 through `9999`, and another block for remote users (i.e., users elsewhere on the network), such as `10000` to `65534`. The important thing is to decide on a scheme and adhere to it.

### GID

Unlike user 0 (the root user), group 0 does not have any special privilege at the kernel level. Traditionally, group 0 had special privileges on many unix variants — either the right to use su to become root (after typing the root password), or the right to become root without typing a password. Basically, the users in group 0 were the system administrators. When group 0 has special privileges, it is called wheel

### umask and file permissions

In computing, umask is a command that determines the settings of a mask that controls how file permissions are set for newly created files. (Wikipedia definition) In other words, you need permissions to be able to read, write or execute files or directories.

The file permissions works as follows:

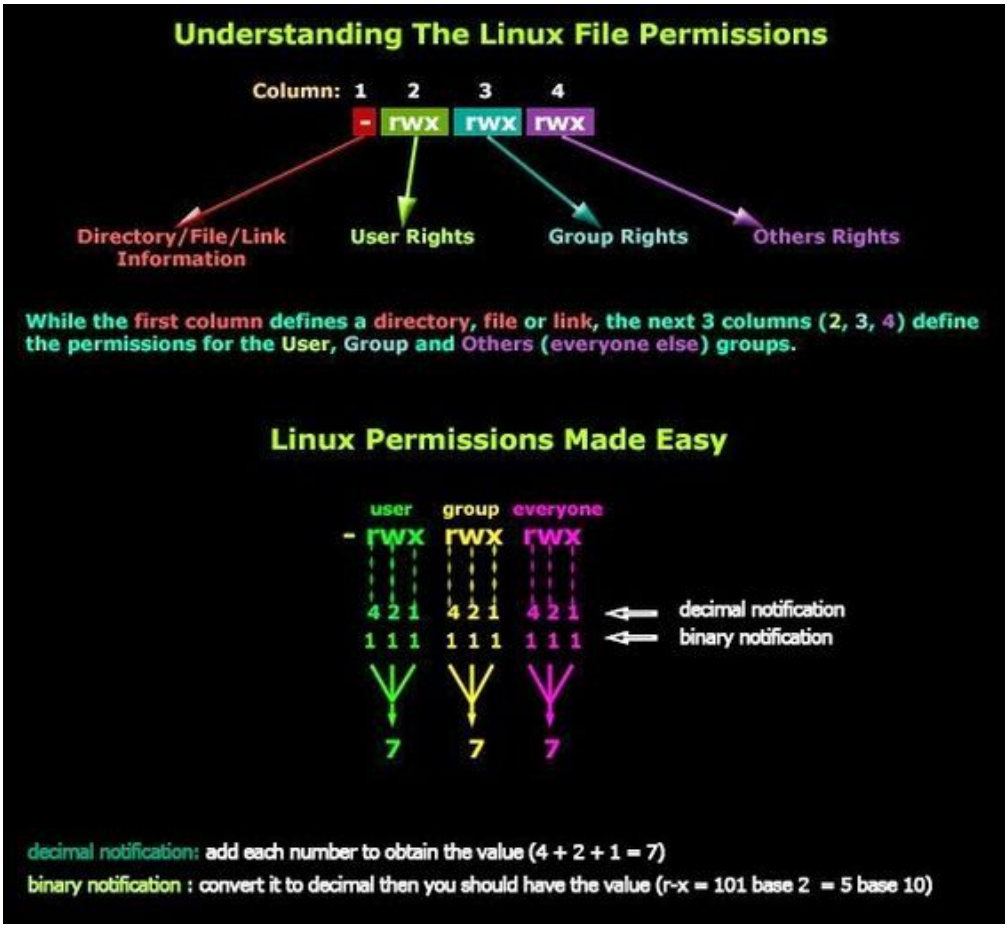


**Fig. 4: linux-file-permissions.jpg**

The umask is how you will restrict, by default, on newly created files. Think of working the opposite way of the file permissions describe in the image above.

Here is an example:

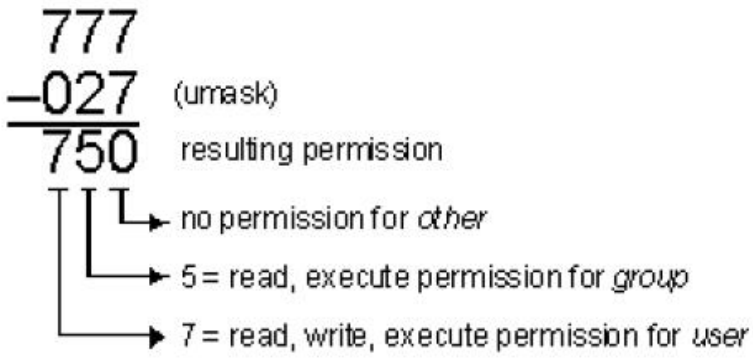


**Fig. 5: umask777.jpg**

You start with the full permission on read, write and execute for *owner*, *group* and *guests* then you substract what you want to restrict. Here we do not want *guests* to be able to use the files, and we restrict write permission for *group*

## User management

**id** : View the user of the current session
**useradd** or **adduser** : Add a user
**groupadd** or **addgroup** : Add a group
**passwd** : Create or change password

### Directory and file properties

**chmod** : Change the protections of a file
**chown** : Change the owner of a file

> **? Give the commands to...**
>
> - Identify the UID - GID numbers and symbolic identifiers of the current user logged
> - Login as root user
> - Change working directory to the folder where you created files earlier
> - Change the permissions as root on a file to give access to only the group `0`
> - Login as regular user and try to open or read the file
> - Identify the UID and GID of a library file in `/usr/lib`
> - Print the content of `/etc/passwd` in the terminal
> - In a new folder, create a new file named *file_new* and give the following permissions: read and write for *user*, read and execute for *group* and execute for *guests*

> **? Give the commands to...**
>
> - Add a new group “group_test”
> - Add a new user “user_test”, assign to this user the “/home/user_test” home directory, associate this user with group “group_test”
> - Create the “/home/user_test” home directory
> - Grant UID and GID to “/home/user_test” home directory according to the user “user_test”
> - Give a new password to the new user “user_test”

> **? Give the commands to...**
>
> - Grant read access to created files to all members of your group
> - Grant read access to the working directory to all members your group
> - Grant read permissions to everyone on the system to created files which you own so that everyone may read it
> - Grant read permissions on the working directory to everyone on the system
> - Grant modify or delete permissions to created files which you own for everyone in the group (!)
> - Deny read access to created files by everyone except yourself
> - Allow everyone in your group to be able to modify the dmesg file
> - Assign yourself full access to read and modify the dmesg file, allow members of the group to read it and do not allow any others access
> - Assign execute status to a script with the same access as the previous example. (Without it, a script is like any other text file)

## Mount filesystem

> **? Give the commands to...**
>
> - Create a file that will serve as a filesystem with the command `dd if=/dev/zero of=fs_test bs=block_size count=block_count`. Replace `block_size` by 2048 or 4096 and `block_count` by the amount you want.
> - For testing purposes, we will create an ext2 filesystem. Use the command `mkfs.ext2 fs_test`

Now that we have a filesystem, it is similar than plugging an USB key or an external hard drive.

> **? Give the commands to...**
>
> - Create a new folder that will be named `mountpoint_fs`
> - Mount the ext2 filesystem to this folder

This is a temporary mounting of this filesystem. In order to mount automatically this filesystem, we need to add it to the `/etc/fstab` file.

The `fstab` file is a configuration table designed to ease the burden of mounting and unmounting file systems to a machine.

It is composed of 6 columns, where each column designates a specific parameter and must be set up in the correct order.

> ⚠ **WARNING**
>
> **If this file is INCORRECTLY set up, your machine won't probably boot anymore. Use it with upmost care !**

**Device:** usually the given name or UUID of the mounted device (/dev/sda1 or /dev/sga2, it could be the path of the filesystem).
**Mount Point:** designates the directory where the device is/will be mounted.
**File System Type:** nothing trick here, shows the type of filesystem in use.
**Options:** lists any active mount options. If using multiple options they must be separated by commas.
**Backup Operation:** (the first digit) this is a binary system where 1 = dump utility backup of a partition. 0 = no backup. This is an outdated backup method and should NOT be used.
**File System Check Order:** (second digit) Here we can see three possible outcomes. 0 means that fsck will not check the filesystem. Numbers higher than this represent the check order. The root filesystem should be set to 1 and other partitions set to 2.

> ⓘ **Info**
>
> An `/etc/fstab` line could be something like this: `/dev/sda5 /media ext2 defaults 0 2`

## Network

### IP address

Your ip address is the label to identify your machine in a network. You usually have a local network (with a local ip address) and a DHCP () serves locally ip addresses. Nowadays, your internet box is serving as a DHCP. Your ip address usually ressembles to `192.168.X.X` where X is a number between 0 and 255. Though, certain numbers are restricted, like 0 (identify the network) or 254 (usually router address or gateaway), 255 serves as broadcast address. In practice, we usually use a range from 1 to 253. In order to get an ip address, your machine needs an interface. This interface is either the ethernet connexion or Wi-Fi cards. Respectively, they are named `eth0` and `wlp0`.

> ? **Give the commands to...**
>
> - `ifconfig -a` or `ip a` gives you the name of the interfaces on your machine
> - Identify the interfaces you have
> - Identify the ip address that is attributed to the active interface
> - Deactivate your active interface and reactivate it, look at the ip address. Is it the same?
> - Change the ip address of the active interface using `ifconfig`
> - Verify that the ip address your entered is the correct one
> - Using `ip` identify the ip address and gateway

### nmcli

The command `nmcli` (Network Manager Command LIne) is a tool which is used for controlling NetworkManager, you can then monitor, check, connect to networks via command line. It is more powerful than `ifconfig` and `ip` since you can see the signal from wireless networks for instance.

**? Give the commands to...**

- List the interfaces
- List the wireless networks with the SIGNAL, SSID and BSSID
- Filter (using `awk`) the wireless networks containing the word JUNIA
- Filter the wireless networks that is active on your interface. Do the same using the option `connexion`.
- Disconnect the wireless network from this interface
- Reconnect to a different network (or the same one) using the ID, then using the UUID

### Download files from internet

**? Give the commands to...**

- Do `curl google.com` and `wget google.com`, what are the outputs and differences? Try to use what you see and `man wget` and `man curl` to explain.
- You can as well use `git clone` to copy a git repository. Try to go to a new folder and do `git clone https://github.com/torvalds/linux.git`. What do you see?

### SSH

The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. The SSH protocol uses public key cryptography for authenticating hosts and users. The authentication keys, called SSH keys, are created using the `ssh-keygen` program.

**? Give the commands to...**

- Connect to the host of your partner. In order to achieve this create a new user with a password (choose whatever you want for username and password).
- Next step is to create an SSH key.

To set up an SSH key, type in the following command:

```
ssh-keygen -t rsa
```

Follow the instructions prompt to you. The next step is to copy the key to the host you want to SSH in. In order to do so:

```
ssh-copy-id -i ~/.ssh/id_rsa user@host
```

- Now, try to connect again to the host, what is happening?
- Rename the file `~/.ssh/id_rsa` to `~/.ssh/id_rsa.bak` and try again to connect to the host. Explain what is happening.
- Try to securely copy a file to the other computer using the command `scp`

### Firewall

**? Give the commands to...**

- Install `ufw`, an 'uncomplicated firewall'
- To see the status of the ufw (if it is enabled or not) use the command `sudo ufw status`
- If it is not enabled, use `sudo ufw enable`
- Now by using `sudo ufw status verbose`, you should see the rules set with this firewall.
- Try to explain what you see
- To allow connexions you can use `sudo ufw allow [rule]`, to deny a connexion `sudo ufw deny [rule]`
- Deny http and https connexions using port (80 and 443), you should be able to use your web browser to go to any site
- Allow them again
- By using `nslookup` (install it if necessary), try to find the ip address of www.google.fr and deny the connexion to it.

## Install softwares manually

### Compilation

**? Give the commands to...**

- If you previously installed `fzf`, remove it from your system. Don't forget to remove the dependancies. Refer to the manual of the package manager on your distribution (use `man <name of package manager>`)
- Then download the repository at `https://github.com/junegunn/fzf.git`
- Review the folder and try to find cues to build (compile) this project
- Once you have built it, what happened?

### Binaries

When you type a command into the terminal on Linux, most of the time, you just call programs that are located in `/usr/bin`. There are other places on your system that commonly hold executable programs as well; some common ones include `/usr/local/bin`. In fact, it is a bit more complicated than that. In linux, you can install programs wherever you want, you just need to update the variable that is called `$PATH`.

**? Give the commands to...**

- Run the command `echo $PATH`, what it is displaying?
- Run the command `fzf`. What is happening? Explain why it is not working!
- Now update the path by adding the directory where you cloned the fzf git repository and built it by doing export `PATH=$PATH:<put the path here>`
- Run the `fzf` command again, is it working?

### Libraries

Libraries (a collection of functions for the software to run) are working in the same fasion as the binaries. You need to export the folders where you have your libraries to a specific variable.

**? Give the commands to...**

- Find the variable related to the libraries.
- What are the usual folder where the libraries are stored?

## Scripting

Shell scripting is the set of commands to be executed such that the shell can execute them. It is said to be the combination of long and repeatable command sequences into one script so that it can be executed as and when required. The main idea behind creating a shell script is to lessen the load of the end-user.

A script file is just a text file with a shebang at the very beginning. A shebang is started with `#!` following by the binary of the shell you want to use. Here we will use `bash`, then the complete shebang is `#!/bin/bash`

You can set variables by doing for instance `test='this is a test string'` and to call the value of the variable you can use `$test`. Here let's print the string `echo $test`. It should display `this is a test string`. You can do the same with integers, floats, paths... (See below at functions to have more details)

Once your script is finished, you need to make it executable. To do so, in the terminal, you can type `chmod +x myScript.sh`, if the name of your script is `myScript.sh`. Then to execute your script you have use `myScript.sh` directly in your terminal.

### Script parameters

`$#` : This parameter represents the total number of arguments passed to the script.
`$0` : This parameter represents the script name.
`$n` : This parameter represents the positional arguments corresponding to a script when a script is invoked such `$1`, `$2`...
`$*` : This parameter describes the positional parameters to be distinct by space. For example, if there are two arguments passed to the script, this parameter will describe them as `$1`, `$2`.
`$$` : This parameter represents the process ID of a shell in which the execution is taking place.
`$!` : This parameter represents the process number of the background that was executed last.
`$@` : This parameter is similar to the parameter `$*`.
`$?` : This parameter represents exit status of the last command that was executed. Here 0 represents success and 1 represents failure.
`$_` : This parameter represents the command which is being executed previously.
`$-` : This parameter will print the current options flags where the set command can be used to modify the options flags.

```
$ cat  program.sh
echo "The File Name is: $0"
echo "The First argument is: $1"
echo "The Second argument is: $2"
```

```
$ sh program.sh ab cd
The File Name: program.sh
The First argument is: ab
The Second argument is: cd
```

## Functions

Bash functions are like usual functions in C or PYTHON. You can call them numerous times to do the same task. The syntax for declaring a bash function is straightforward.

```
function_name () {
  commands
}
```

Or you can specify the reserved word `function`:

```
function function_name {
  commands
}
```

As an exemple, a simple function printing `hello, world` is:

```
hello_world () {
   echo 'hello, world'
}
```

You can as well use variables, like in any other programmatic languages:

```
var1='A'
var2='B'

my_function () {
  local var1='C'
  var2='D'
  echo "Inside function: var1: $var1, var2: $var2"
}

echo "Before executing function: var1: $var1, var2: $var2"

my_function

echo "After executing function: var1: $var1, var2: $var2"
```

> **? Give the commands to...**
>
> - Execute this simple program and explain what `local` is doing (put this in a shell script and execute the script).

To return values or strings, you can use the following method:

```
my_function () {
  local func_result="some result"
  echo "$func_result"
}

func_result="$(my_function)"
echo $func_result
```

Here the local variable `func_result` is created, and you can print to STDOUT the result using `echo`. Then you can pass the result of the function to a global variable using `$(my_function)`.

Lastly, to have functions asking for an input, you can use the following:

```
greeting () {
  echo "Hello $1"
}

greeting "Joe"
```

Here calling the function `greeting` with `"Joe"` in front of it you pass the string `"Joe"` to the first positional argument.

> **? Give the commands to...**
>
> - Create a shell script that takes into input a text file and ask for a word (could be a simple string or a number) or a sentence to be found in the text file. Please think of potential errors (check if the given file is a text file, word or sentence and if it is not empty...)
> - Create a simple script containing a function that prints the current directory, then changes the directory to the desired directory (use positional arguments) and then prints again the directory
> - What is happening with your current directory, is it changed? Can you explain why?
> - Try to look at the PID of the shell used (you can add `echo $$` at the end of your script to print it in the terminal) and the one of your terminal that you are currently using.
> - How to solve this?

## Conditions and loops

Conditions are useful to control your script behaviour. It can bypass unnecessary parts if a condition is met for instance. In bash script, you can find the traditionnal `if`:

```
if [[ conditional_statement ]]

then

    command

fi
```

As for loops, you can use the `for`:

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

By combining both of them, we can do interesting things:

```
#!/bin/bash
FILES="$@"
for f in $FILES
do
        # if .bak backup file exists, read next file
    if [ -f ${f}.bak ]
    then
        echo "Skiping $f file..."
        continue  # read next file and skip the cp command
    fi
        # we are here means no backup file exists, just use cp command to copy file
    /bin/cp $f $f.bak
done
```

In the `if` condition, it checks a .bak file exists for each file, if not, it creates one.

> **? Give the commands to...**
>
> - You can do as well the same if a folder exists or not.
> - Create a small script to check if in the folder `/home/downloads/` you have a folder named `books`, and then checks if an `.pdf` file exists. If not, download the file at https://cloud.capiod.fr/s/WsMNqWnRAdSwnWk/download/Book - Linux Command Line and Shell Scripting Bible.pdf
> - The input `$@` is waiting for a folder, if you do not enter a folder, the script won't work. How to modify the script to make it more robust?

## The Linux kernel

In this section, we will not dive into modifying the Linux kernel, but it will present the kernel and what it is doing.

The following figure depicts the overall role of the linux kernel:

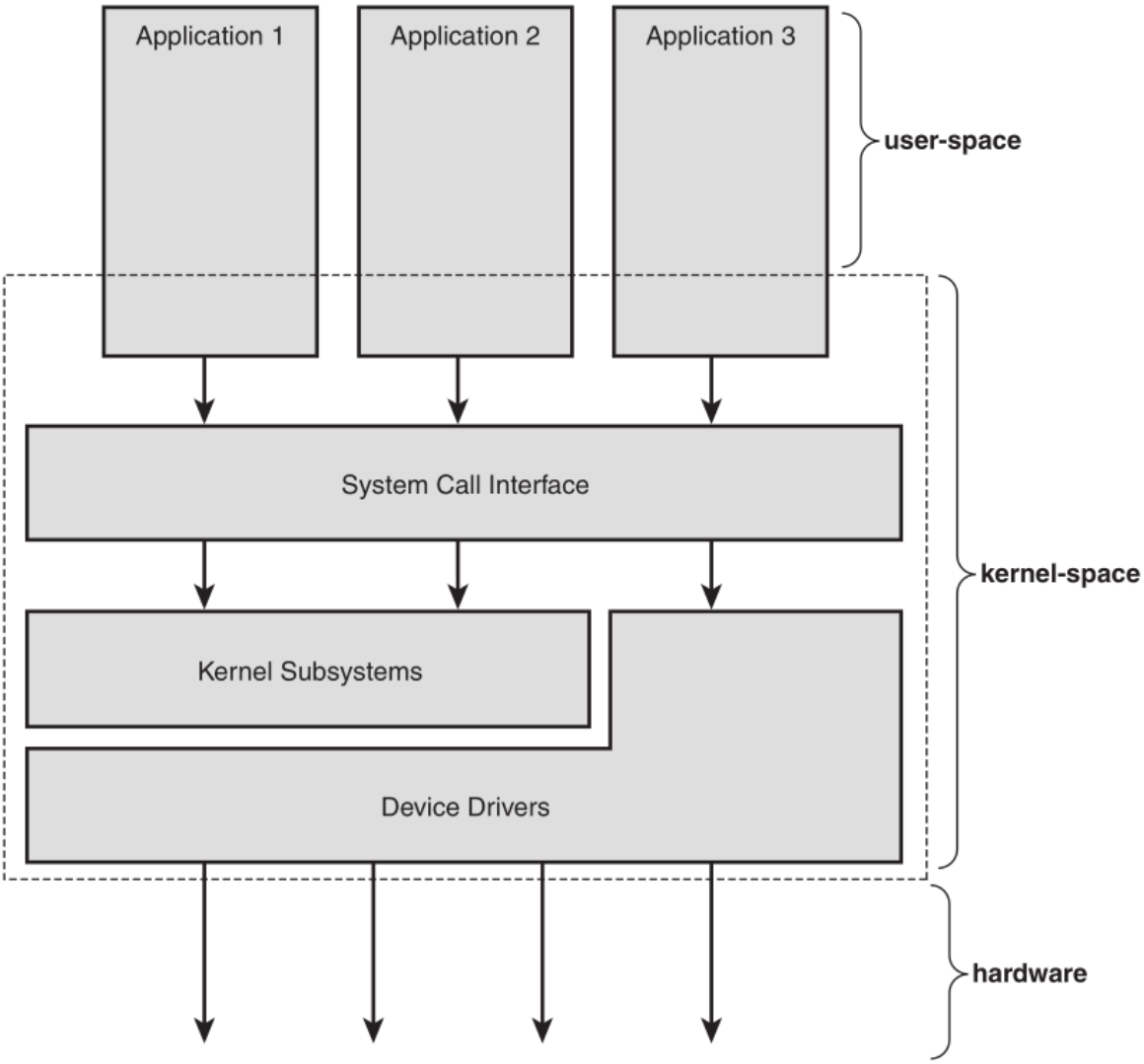


**Fig. 6: Kernel space is the link between the applications and the hardware**

Applications running on the system communicate with the kernel via **system calls**. An application typically calls functions in a library -for example, the **C library-** that in turn rely on the system call interface to instruct the kernel to carry out tasks on the application's behalf.

The kernel also manages the system's hardware. Nearly all architectures, including all systems that Linux supports, provide the concept of interrupts.When hardware wants to communicate with the system, it issues an interrupt that literally interrupts the processor, which in turn interrupts the kernel.

A number identifies interrupts and the kernel uses this number to execute a specific interrupt handler to process and respond to the interrupt. For example, as you type, the keyboard controller issues an interrupt to let the system know that there is new data in the keyboard buffer.The kernel notes the interrupt num- ber of the incoming interrupt and executes the correct interrupt handler.The interrupt handler processes the keyboard data and lets the keyboard controller know it is ready for more data.

The Core Subsystems of the Linux Kernel are as follows:

- The Process Scheduler
- The Memory Management Unit (MMU)
- The Virtual File System (VFS)
- The Networking Unit
- Inter-Process Communication Unit

For the purpose of this article we will only be focusing on the first three important subsystems of the Linux Kernel.

The basic functioning of each of the first three subsystems is elaborated below:

**The Process Scheduler:** This kernel subsystem is responsible for fairly distributing the CPU time among all the processes running on the system simultaneously.

**The Memory Management Unit:** This kernel sub-unit is responsible for proper distribution of the memory resources among the various processes running on the system. The MMU does more than just simply provide separate virtual address spaces for each of the processes.

**The Virtual File System:** This subsystem is responsible for providing a unified interface to access stored data across different filesystems and physical storage media.

**The Network Subsystem** This allows Linux systems to connect to other systems over a network. There are a number of possible hardware devices that are supported, and a number of network protocols that can be used. The network subsystem abstracts both of these implementation details so that user processes and other kernel subsystems can access the network without necessarily knowing what physical devices or protocol is being used.

**Inter-Process Communication** Processes communicate with each other and with the kernel to coordinate their activities. Linux supports a number of Inter-Process Communication (IPC) mechanisms. Signals and pipes are two of them but Linux also supports the System V IPC mechanisms named after the Unix TM release in which they first appeared.

| Directory | Description |
|---|---|
| arch | Architecture-specific source |
| block | Block I/O layer |
| crypto | Crypto API |
| Documentation | Kernel source documentation |
| drivers | Device drivers |
| firmware | Device firmware needed to use certain drivers |
| fs | The VFS and the individual filesystems |
| include | Kernel headers |
| init | Kernel boot and initialization |
| ipc | Interprocess communication code |
| kernel | Core subsystems, such as the scheduler |
| lib | Helper routines |
| mm | Memory management subsystem and the VM |
| net | Networking subsystem |
| samples | Sample, demonstrative code |
| scripts | Scripts used to build the kernel |
| security | Linux Security Module |
| sound | Sound subsystem |
| usr | Early user-space code (called initramfs) |
| tools | Tools helpful for developing Linux |
| virt | Virtualization infrastructure |

**Fig. 7: kernel-source-tree.png**

> **? Question**
>
> - Try to tell (when possible) if a folder/modules leans more towards the *user-space* side of the kernel, more torwards the *hardware side* or if it is a kernel core module.

Here are some references that you can use and read if you are curious[1]

## Automatization

Init.d and Systemd are both init daemons (these are the famous *hacky* white lines on a black background that are spawning when you start and shutdown your linux OS). It is better to use systemd as it is the one used in most of recent distributions. Though init.d is still present in some. Both of those init daemons works with the following methods:

- You can start a script in the current session (without enabling it). When you reboot, the script will not be run
- You can start a script in the current session (without starting it). It will not be started in the current session but will when you reboot.
- You can start and enable a script in the current session.

### Systemd

Systemd (system daemon) is an init daemon used by modern systems and starts system services in parallel which remove unnecessary delays and speeds up the initialization process. Systemd uses Unit Dependencies to define whether a service wants/requires other services to run successfully, and Unit Order to define whether a service needs other services to be started before/after it.

To create a service, you will need to write a `.service` file stored in the /etc/systemd/system directory. You would write a file `/etc/systemd/system/myService.service` that looks something like this:

```
[Unit]
Description=Some Description
Wants=network-online.target
After=network-online.target

[Service]
ExecStart=/usr/sbin/<command-to-start>
ExecStop=/usr/sbin/<command-to-stop>

[Install]
WantedBy=multi-user.target
```

In order to enable the script, the command is `sudo systemctl enable myService.service` and to start it `sudo systemctl start myService.service`. You can check the status of the script (if there is any issue with it for instance) with `sudo systemctl status myService`.

> **? Question...**
>
> - Using the man page, explain what the `Wants` and `After` options are meant for?

### init.d

Init.d also known as System V init, or SysVinit. It defines six run-levels (system states) and maps all system services to these run-levels. This allows all services (defined as scripts) to be started in a pre-defined sequence.

**Only if your distribution is using init.d**

Create a script and place in /etc/init.d (e.g /etc/init.d/myService). The script should have the following format:

```
#!/bin/bash
# chkconfig: 2345 20 80
# description: Description comes here....

# Source function library.
. /etc/init.d/functions

start() {
    # code to start app comes here
    # example: daemon program_name &
```

```
}

stop() {
    # code to stop app comes here
    # example: killproc program_name
}

case "$1" in
    start)
       start
       ;;
    stop)
       stop
       ;;
    restart)
       stop
       start
       ;;
    status)
       # code to check status of app comes here
       # example: status program_name
       ;;
    *)
       echo "Usage: $0 {start|stop|status|restart}"
esac

exit 0
```

The format is pretty standard, and you can view existing scripts in /etc/init.d. You can then use the script like so `/etc/init.d/myService start` or `chkconfig myService start`.

> **? Question...**
>
> - Using the man pages, explain the run levels `2 3 4 5, 20, 80` that are written at the very top of the script.

The example start, stop and status code uses helper functions defined in `/etc/init.d/functions`

Enable the script: `chkconfig --add myService` `chkconfig --level 2345 myService on`

Check the script is indeed enabled - you should see “on” for the levels you selected. `chkconfig --list | grep myService`

> **? Using the init daemon of your system:**
>
> - Write using either the systemd or init.d a script to run a package manager update every 10 minutes. Check that the script is effectively running.
> - Write a second script that dump the last 100 lines of the system’s journal, every 5 minutes to file in your home folder and make a backup of that file.

## Virtualization - If you have time !

We will have a look at two ways of virtualizations. First, using a software hypervisor (emulator). It runs as an application installed on an operating system:

- Simulation of physical components
- Access to the material through the host operating system
- Less efficient

You can use software packages such as VMWare Player or Oracle VirtualBox

The second way of virtualization is more recent and is more lightweight than virtualizing a whole operating system. It uses the same kernel as the host and only virtualize applications. It needs an engine to *plug containers* : Docker.

- No simulation of physical components
- Only virtualize the necessary units to run
- Lightweight

To schematically visualize the differences:

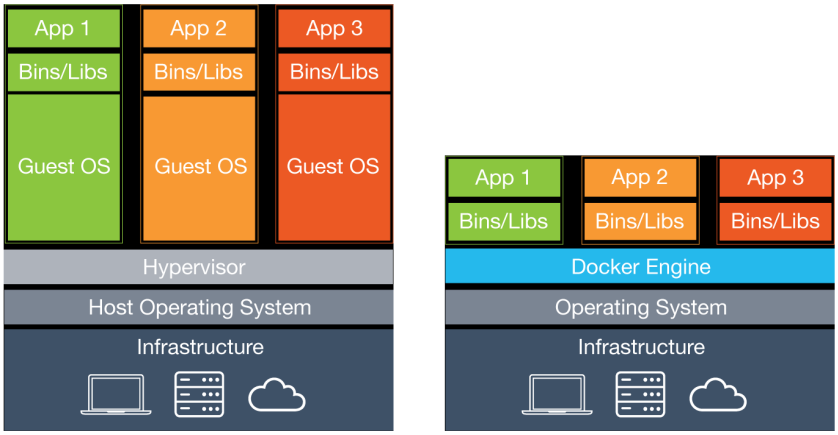


**Fig. 8: VM versus Docker**

> **? Question...**
>
> - From the image and your knowledge (use what we have done before), explain the biggest difference between a VM player and docker, use as many details as possible (for instance, it is obvious that the Guest OS is not present for Docker, but explain in the end what the Guest OS does and why its "absence" in Docker is important).

### Using VMware or Virtual Box

If you are adventurous, you can install VMware player on your distribution. You can download a bundle file at this address:
https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

For Oracle's VirtualBox, the list of distrubtions where you can download specific versions of the software is available here:
https://www.virtualbox.org/wiki/Linux_Downloads

> **ⓘ Information**
>
> Once your software of choice is installed, download a distribution of choice (.iso file) and install it using the virtualization software. If you have difficulties setting up the iso file, call your professor.

### Using Docker

#### Ubuntu

1. Update your system and install required dependancies:

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

2. Add Docker's official GPG key:

```
sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

3. Use the following command to set up the repository:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Install the docker engine:

```
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

**Receiving a GPG error when running apt-get update?**
Your default umask may not be set correctly, causing the public key file for the repo to not be detected. Run the following command and then try to update your repo again: sudo chmod a+r /etc/apt/keyrings/docker.gpg

5. Verify that Docker Engine is installed correctly by running the hello-world image. `sudo docker run hello-world`

### Fedora

Install the dnf-plugins-core package (which provides the commands to manage your DNF repositories) and set up the repository.

`sudo dnf -y install dnf-plugins-core`

`sudo dnf config-manager --add-repo https://download.docker.com/linux/fedora/docker-ce.repo`

Install the latest version of Docker Engine, containerd, and Docker Compose or go to the next step to install a specific version:

`sudo dnf install docker-ce docker-ce-cli containerd.io docker-compose-plugin`

If prompted to accept the GPG key, verify that the fingerprint matches 060A 61C5 1B55 8A7F 742B 77AA C52F EB6B 621E 9F35, and if so, accept it.

This command installs Docker, but it doesn't start Docker. It also creates a docker group, however, it doesn't add any users to the group by default.

Start Docker: `sudo systemctl start docker`

Verify that Docker Engine is installed correctly by running the hello-world image. `sudo docker run hello-world`

This command downloads a test image and runs it in a container. When the container runs, it prints a message and exits.

### Arch

First update your system : `sudo pacman -Syu`

Then the following command should install all the necessary files: `sudo pacman -S docker`

Follow the additional steps below to add your user to the docker group.

Verify that Docker Engine is installed correctly by running the hello-world image. `sudo docker run hello-world`

### Additional steps post-install on Linux

#### To create the docker group and add your user

If you recieve the error `Got permission denied while trying to connect to the Docker daemon socket`. This means that your user is not allowed to run the docker daemon. You will need to add the current user to the docker group.

- First add the docker group if it does not exist: `sudo groupadd docker`
- Then add your user to the docker group: `sudo usermod -aG docker $USER`

Log out and log back in so that your group membership is re-evaluated.

### Configure Docker to start on boot

In order to start the docker daemon on boot, use the following commands:

`sudo systemctl enable docker` and `sudo systemctl enable containerd`

> ⓘ **Information**
>
> For the course and labs on Docker, please refer to the Docker.pdf available in the teams channel.

## Home assignment

In those home assignments, you will create scripts with a menu that asks for specific items. The menu and the scripts need to be robust. For example, if the script asks for a file, and you give it a number, it needs to output an error and ask again. Once a job is done, the script goes back to the menu and the scripts asks again. Create an escape (for instance “quit” or “q” as an answer).

**Assignments : Pick one (or two if you are adventurous!)**

> ? **question**
>
> - Create a script that takes a text file as an input. You will create a menu that asks to cat the text file, grep a file or a sentence, or grep all words that starts with a selected letter

> ? **question**
>
> - Create a script that scans for Wi-Fi networks, a menu will ask:
> - filters by having signal strength over 65
> - connects to a network using ssid or bssid

> ? **question**
>
> - Create a script that asks to update the package manager, searches for a software, installs it or not, cleans the cache of the package manager and removes packages with its depencies

> ? **question**
>
> - Create a script that displays all running services, select one of them and the script asks to restart or stop the selected service

> ? **question**
>
> - Create a script that adds a new user using simple functions (you cannot use useradd for instance)

> ? **question**
>
> - Create a script that rewrite the command `pgrep`, and create a menu that asks to kill, start or restart the process (you have to check if the process is running or not)

**? question**

- Create a script that display information about the internet config, changes the IP address, activates or deactivates the ethernet or wifi network (using `ip`)

**? question**

- Create a script that adds firewall rules using `ufw`, displays all existing rules and removes a rule. Create a rule to allow the subnetwork 172.12.0.0 to be opened to 443 and 80 ports.

**? question**

- Create a script that finds either all files or folders in a specific folder, `cat` the content of the selected file or `cd` in the folder. Use `fzf` to filter the files or folders.

**? question**

- Create a script you can use as a notepad. The script will take as input (first positional parameter `$1`) the name of the note you want. If the note already exists it will update and adds the sentence at the very end, if note it will create it. Then the script asks want you want to write down in the note (usually a sentence).

---

1. Robert Love - Linux Kernel Development : https://cloud.capiod.fr/s/fwCaGHQrLZEGwLN ↩