

# Linux Administrator

Alejandro Campos

October, 2023

# Contents

<b>1</b>	<b>Linux Directory Structure Explained</b>	<b>9</b>
1.1	/ – The Root Directory . . . . .	9
1.2	/bin – Essential User Binaries . . . . .	9
1.3	/sbin – System Administration Binaries . . . . .	9
1.4	/boot – Static Boot Files . . . . .	9
1.5	/dev – Device Files . . . . .	9
1.6	/etc – Configuration Files . . . . .	9
1.7	/home – Home Folders . . . . .	9
1.8	/root – Root Home Directory . . . . .	10
1.9	/lib – Essential Shared Libraries . . . . .	10
1.10	/lost+found – Recovered Files . . . . .	10
1.11	/media – Removable Media . . . . .	10
1.12	/mnt – Temporary Mount Points . . . . .	10
1.13	/opt – Optional Packages . . . . .	10
1.14	/tmp – Temporary Files . . . . .	10
1.15	/usr – User Binaries & Read-Only Data . . . . .	10
1.16	/var – Variable Data Files . . . . .	11
<b>2</b>	<b>Managing Linux Users and Groups</b>	<b>12</b>
2.1	Introduction . . . . .	12
2.2	Managing Linux Users . . . . .	12
2.2.1	Type of Users in Linux . . . . .	12
2.2.2	Understanding the /etc/passwd file . . . . .	12
2.2.3	Understand the /etc/shadow file . . . . .	13
2.2.4	Add User to Linux System . . . . .	14
2.2.5	Switching User in Linux - su command . . . . .	16
2.2.6	Delete User in Linux System . . . . .	16
2.2.7	Manage User from Linux System . . . . .	17
2.2.8	Switch Users in Linux Terminal . . . . .	17
2.2.9	SSH Key-Based Authentication . . . . .	18
2.3	Managing Linux Groups . . . . .	18
2.3.1	Understanding the /etc/group file . . . . .	18
2.3.2	Add a Group in Linux . . . . .	18
2.3.3	Change the group ID . . . . .	19
2.3.4	Rename a group . . . . .	19
2.3.5	How to Assign Users to Groups in Linux . . . . .	19

2.3.6	How to Delete Users from Groups in Linux . . . . .	19
2.3.7	How to delete a group . . . . .	19
2.4	Sudo Command . . . . .	19
2.4.1	Give sudo permissions with password . . . . .	20
2.4.2	Give only some actions sudo permission . . . . .	21
2.4.3	How to enable sudo without entering a password . . . . .	22
2.4.4	How to gain su permissions with sudo . . . . .	23
2.5	Su - root user . . . . .	23
2.5.1	First Approach - Enabling root account temporary . . . . .	23
2.5.2	Enable root user . . . . .	23
2.5.3	Disable root user . . . . .	24
<b>3</b>	<b>Basic Linux Commands</b>	<b>25</b>
3.1	Linux System Information . . . . .	25
3.2	Change Linux hostname . . . . .	25
3.3	Linux System Consumption . . . . .	26
3.4	Linux Services . . . . .	26
3.4.1	ps . . . . .	26
3.4.2	systemctl . . . . .	26
3.5	Service . . . . .	27
3.6	Journalctl . . . . .	28
3.6.1	Usage . . . . .	28
3.7	Moving between directories . . . . .	29
3.7.1	Introduction . . . . .	29
3.7.2	ls -la Analyzing Results . . . . .	29
3.8	chmod . . . . .	30
3.9	chown . . . . .	31
3.10	Ctrl+R . . . . .	31
3.11	Mkdir (create files) . . . . .	32
3.12	Mv . . . . .	32
3.13	Cp . . . . .	32
3.14	Rsync . . . . .	33
3.15	Simbolik Links . . . . .	33
3.15.1	What is a Symbolic Link? . . . . .	33
3.15.2	Why Use Symbolic Links? . . . . .	34
3.15.3	Symbolic Links Management . . . . .	34
3.16	Aliases . . . . .	34
3.16.1	Ephemeral Aliases . . . . .	34
3.16.2	Permanent Aliases . . . . .	35

3.17	Source . . . . .	35
3.18	Shutdown machines or terminals . . . . .	35
3.19	Others . . . . .	36
<b>4</b>	<b>Package Management Systems</b>	<b>37</b>
4.1	Apt - Debian . . . . .	37
4.1.1	Basic apt commands . . . . .	37
4.1.2	Managing Repositories . . . . .	38
4.2	Dnf - RedHat . . . . .	40
4.2.1	Yum or Dnf? . . . . .	40
4.2.2	Basic dnf commands . . . . .	40
<b>5</b>	<b>.bashrc, .bash_profile, /etc/bashrc &amp; path</b>	<b>42</b>
5.1	.bashrc vs .bash_profile . . . . .	42
5.2	PATH . . . . .	42
5.3	Set your PATH . . . . .	42
5.3.1	For current terminal . . . . .	42
5.3.2	Permanently for interactive sessions . . . . .	42
<b>6</b>	<b>Useful Tools</b>	<b>43</b>
6.1	Cat . . . . .	43
6.1.1	EOF . . . . .	43
6.2	Head & Tail . . . . .	45
6.3	Diff (File comparator) . . . . .	45
6.4	Grep (Global search for Regular Expressions and Print out) . . . . .	46
6.4.1	Two flavors of grep usage . . . . .	46
6.4.2	Useful Flags . . . . .	47
6.5	Find . . . . .	48
6.6	Sed (Stream Editor) . . . . .	49
6.7	AWK . . . . .	50
6.8	Tar Command . . . . .	50
6.8.1	Compress . . . . .	50
6.8.2	Extract . . . . .	51
6.8.3	More flags . . . . .	51
6.9	WGET . . . . .	52
6.10	wc . . . . .	52
6.11	Base64 Encoding . . . . .	53
6.12	JQ . . . . .	54
6.13	YQ . . . . .	54

<b>7</b>	<b>Linux Default Text Editors</b>	<b>55</b>
7.1	Vi . . . . .	55
7.2	Vim . . . . .	55
7.2.1	Introduction . . . . .	55
7.2.2	Command Mode Shortcuts . . . . .	55
7.2.3	Vim Config . . . . .	57
7.3	Nano . . . . .	59
<b>8</b>	<b>Network</b>	<b>61</b>
8.1	Introduction, Switches and Routers . . . . .	61
8.1.1	Switching . . . . .	61
8.1.2	Routing . . . . .	62
8.1.3	Private and Public Networks configuration . . . . .	63
8.2	Network Namespaces in Linux . . . . .	64
8.2.1	Introduction to Namespaces in Linux . . . . .	64
8.2.2	ip addr command . . . . .	64
8.2.3	Namespaces for Networking . . . . .	65
8.3	OSI Model . . . . .	68
8.4	MAC Adress vs IP Address . . . . .	68
8.4.1	Introduction . . . . .	68
8.4.2	MAC Address (Media Control Address) . . . . .	69
8.4.3	IP Address . . . . .	70
8.4.4	Why Both MAC and IP Addresses Are Needed . . . . .	70
8.4.5	ARP Table . . . . .	71
8.5	Public IP . . . . .	71
8.5.1	Introduction . . . . .	71
8.5.2	How to get my public ip? . . . . .	72
8.5.3	Who is this IP assigned to me? . . . . .	72
8.6	DHCP . . . . .	72
8.6.1	Introduction . . . . .	72
8.6.2	How IP's are assigned? . . . . .	73
8.7	Private IP . . . . .	74
8.7.1	Introduction . . . . .	74
8.7.2	Advantages of have Private IP addresses . . . . .	74
8.7.3	Can the private subnet be changed? . . . . .	74
8.7.4	Private IP Address Ranges . . . . .	75
8.7.5	Example of Private IP Assignment . . . . .	75
8.8	DNS in Linux . . . . .	75
8.8.1	Introduction to DNS . . . . .	75

8.8.2	DNS server . . . . .	76
8.8.3	FQDN . . . . .	77
8.8.4	Creating own FQDN . . . . .	79
8.8.5	Record Types (Tipos de Registros) . . . . .	79
8.8.6	DNS Resolution Process . . . . .	79
8.8.7	Caching the Result . . . . .	80
8.8.8	DNS check Tools . . . . .	81
8.8.9	Local Cache Services . . . . .	82
8.9	SSH . . . . .	84
8.9.1	What is SSH? . . . . .	84
8.9.2	Authenticate SSH connections using user-password . . . . .	85
8.9.3	Authenticate SSH connections using Asimetric Key authentication method . . . . .	85
8.9.4	SSH Extra Arguments . . . . .	90
8.9.5	SFTP . . . . .	90
8.9.6	Introduction . . . . .	90
8.9.7	Commands inside the SFTP . . . . .	91
8.9.8	Being sudo with SFTP? . . . . .	92
8.10	Localhost Networking . . . . .	93
8.10.1	Ports Exposed . . . . .	93
8.10.2	Localhost Adresses . . . . .	93
<b>9</b>	<b>Network Tools</b>	<b>95</b>
9.1	Ping . . . . .	95
9.2	Curl . . . . .	95
9.3	Nmap . . . . .	98
9.4	Netcat . . . . .	99
9.5	Telnet . . . . .	100
9.6	Traceroute . . . . .	100
9.7	MTR . . . . .	101
9.8	Whois . . . . .	102
9.9	Proxy variables definition for binaries . . . . .	102
<b>10</b>	<b>Terminal</b>	<b>103</b>
10.1	Ctrl+r . . . . .	103
10.2	Bash . . . . .	103
10.2.1	Introduction . . . . .	103
10.2.2	Configuration .bashrc . . . . .	103
10.2.3	Path . . . . .	104
10.3	ZSH, why should it be used instead of bash? . . . . .	105

10.3.1	What is ZSH? . . . . .	105
10.3.2	ZSH vs Bash . . . . .	105
10.3.3	How to install and enable zsh? . . . . .	105
10.3.4	Configure ZSH . . . . .	105
10.4	Tmux . . . . .	106
10.4.1	Tmux Socket . . . . .	107
10.4.2	Arguments . . . . .	107
10.4.3	Commands Inside tmux . . . . .	110
10.5	Ohmyzsh . . . . .	111
10.5.1	Introduction . . . . .	111
10.5.2	How to install it? . . . . .	111
10.5.3	Post-Installation . . . . .	112
<b>11</b>	<b>Linux Secondary Storage</b>	<b>113</b>
11.1	Introduction . . . . .	113
11.2	Drives . . . . .	113
11.3	Partitions . . . . .	113
11.4	Filesystems . . . . .	114
11.5	LVM . . . . .	114
11.5.1	Introduction . . . . .	114
11.5.2	LVM Structure . . . . .	115
11.6	Tools to Manage Secondary Storage . . . . .	115
11.6.1	Lsblk - List block devices . . . . .	115
11.6.2	Fdisk - Managing Partitions . . . . .	116
11.6.3	LVM - Managing Volumes on Linux . . . . .	118
11.6.4	Filesystem . . . . .	120
<b>12</b>	<b>Storage Area Network (SAN)</b>	<b>122</b>
12.1	Introduction . . . . .	122
12.2	What is a SAN? . . . . .	122
12.3	Key Components of SAN Architecture . . . . .	122
12.3.1	Host Layer (Servers) . . . . .	122
12.3.2	Fabric Layer (Network) . . . . .	122
12.3.3	Storage Layer . . . . .	122
12.4	Interconnect Layer . . . . .	122
12.5	SAN Protocols . . . . .	122
12.6	SAN Architecture Topologies . . . . .	123
12.6.1	Point-to-Point . . . . .	123
12.6.2	Fibre Channel Arbitrated Loop (FC-AL) . . . . .	123

12.6.3	Switched Fabric . . . . .	123
12.7	How SAN Works . . . . .	123
12.7.1	Block-Level Access . . . . .	123
12.7.2	Logical Unit Number (LUN) . . . . .	123
12.7.3	Data Paths . . . . .	124
12.8	Proper Use of SAN . . . . .	124
12.8.1	Storage Virtualization . . . . .	124
12.8.2	Multipathing for High Availability . . . . .	124
12.8.3	Zoning and LUN Masking . . . . .	124
12.8.4	Data Replication and Backup . . . . .	124
12.8.5	Snapshot and Cloning . . . . .	124
12.8.6	Storage Tiering . . . . .	124
12.8.7	Quality of Service (QoS) . . . . .	124
12.9	Benefits of SAN . . . . .	124
12.10	Common SAN Use Cases . . . . .	125
<b>13</b>	<b>Logs</b>	<b>126</b>
13.1	Rsyslog . . . . .	126
<b>14</b>	<b>Others</b>	<b>127</b>
14.1	Docker Desktop . . . . .	127
14.2	Helm . . . . .	128
14.3	JSONPath . . . . .	128
14.3.1	Basic Usage . . . . .	128
14.3.2	Filters . . . . .	129
14.3.3	Example . . . . .	130
14.3.4	JSONPath K8s . . . . .	134
14.4	Ubuntu vs RH7 . . . . .	135
14.5	Vagrant . . . . .	135
14.6	Red Hat Package Installer . . . . .	136
14.6.1	RPM . . . . .	136
14.6.2	YUM . . . . .	136
14.7	Timeshift to Backup and Restore Your Linux System . . . . .	136
14.7.1	Introduction . . . . .	136
14.7.2	Timeshift Installation . . . . .	136
14.7.3	Create a Snapshot . . . . .	136



# 1 Linux Directory Structure Explained

## 1.1 / – The Root Directory

Everything on your Linux system is located under the / directory, known as the root directory. You can think of the / directory as being similar to the C:\ directory on Windows. But this isn't strictly true, as Linux doesn't have drive letters. While another partition would be located at D:\ on Windows, this other partition would appear in another folder under / on Linux.

## 1.2 /bin – Essential User Binaries

The /bin directory contains the essential user binaries (programs) that must be present when the system is mounted in single-user mode. Users applications such as Firefox are stored in /usr/bin, while important system programs and utilities such as the bash shell, python, ansible, docker are located in /bin. Placing these files in the /bin directory ensures the system will have these important utilities even if no other file systems are mounted.

## 1.3 /sbin – System Administration Binaries

The /sbin directory is similar to the /bin directory. It contains essential system administration binaries, which are generally intended to be run by the root user for system administration.

## 1.4 /boot – Static Boot Files

The /boot directory contains the files needed to boot the system. For example, the GRUB boot loader's files and your Linux kernels are stored here. The boot loader's configuration files aren't located here, though they're in /etc with the other configuration files.

## 1.5 /dev – Device Files

Linux exposes devices as files, and the /dev directory contains a number of special files that represent devices. This directory also contains pseudo-devices, which are virtual devices that don't actually correspond to hardware. For example, /dev/random produces random numbers. /dev/null is a special device that produces no output and automatically discards all input. When you pipe the output of a command to /dev, you discard it

## 1.6 /etc – Configuration Files

The /etc directory contains configuration files, which can generally be edited by hand in a text editor. Note that the /etc directory contains system-wide configuration files.

### NOTE

user-specific configuration files are located in each user's home directory.

## 1.7 /home – Home Folders

The /home directory contains a home folder **for each user**. For example, if your user name is bob, you have a home folder located at /home/bob. This home folder contains the user's data files and user-specific configuration files. **Each user only has write access to their own home folder** and must obtain elevated permissions (become the root user) to modify other files on the system.

## 1.8 /root – Root Home Directory

The `/root` directory is the home directory of the root user. Instead of being located at `/home/root`, as the rest of the users, it's located at `/root`. This is distinct from `/`, which is the system root directory.

## 1.9 /lib – Essential Shared Libraries

The `/lib` directory contains libraries needed by the essential binaries in the `/bin` and `/sbin` folder.

### NOTE

Libraries needed by the binaries in the `/usr/bin` folder are located in `/usr/lib`.

## 1.10 /lost+found – Recovered Files

Each Linux file system has a `lost+found` directory. If the file system crashes, a file system check will be performed at next boot. Any corrupted files found will be placed in the `lost+found` directory, so you can attempt to recover as much data as possible.

## 1.11 /media – Removable Media

The `/media` directory contains subdirectories where removable media devices inserted into the computer are mounted. For example, when you insert a CD into your Linux system, a directory will automatically be created inside the `/media` directory. You can access the contents of the CD inside this directory.

## 1.12 /mnt – Temporary Mount Points

Historically speaking, the `/mnt` directory is where system administrators mounted temporary file systems while using them. For example, if you're mounting a Windows partition to perform some file recovery operations, you might mount it at `/mnt/windows`. However, you can mount other file systems anywhere on the system.

## 1.13 /opt – Optional Packages

The `/opt` directory contains subdirectories for optional software packages. It's commonly used by proprietary software that doesn't obey the standard file system hierarchy. For example, a proprietary program might dump its files in `/opt/application` when you install it.

## 1.14 /tmp – Temporary Files

Applications store temporary files in the `/tmp` directory. These files are generally deleted whenever your system is restarted and may be deleted at any time by utilities such as `tmpwatch`.

## 1.15 /usr – User Binaries & Read-Only Data

The `/usr` directory contains applications and files used by users, as opposed to applications and files used by the system. For example, non-essential applications are located inside the `/usr/bin` directory instead of the `/bin` directory and non-essential system administration binaries are located in the `/usr/bin` directory instead of the `/sbin` directory.

## 1.16 /var – Variable Data Files

The `/var` directory is the writable counterpart to the `/usr` directory, which must be read-only in normal operation. Log files and everything else that would normally be written to `/usr` during normal operation are written to the `/var` directory. For example, you'll find log files in `/var/log`.

## 2 Managing Linux Users and Groups

### 2.1 Introduction

Linux is a multi-user system, which means that more than one person can interact with the same system at the same time. As a system administrator, you have the responsibility to manage the system's users and groups by creating and removing users and assign them to different groups .

### 2.2 Managing Linux Users

In a Linux system, users refer to individuals or entities that interact with the operating system by logging in and performing various tasks. User management plays a crucial role in ensuring secure access control, resource allocation, and system administration.

A user in Linux is associated with a user account, which consists of several properties defining their identity and privileges within the system. These properties are a username, UID (User ID), GID (Group ID), home directory, default shell, and password.

#### 2.2.1 Type of Users in Linux

Linux supports two types of users: system users and regular users.

- **System users:** are created by the system during installation and are used to run system services and applications.
- **Regular users:** are created by the administrator and can access the system and its resources based on their permissions.

#### 2.2.2 Understanding the `/etc/passwd` file

User account information is stored in the `/etc/passwd` file. This information includes the account name, home directory location, and default shell, among other values. Each field is separated by a ":" character, and not all fields must be populated, but you must delineate them.

```
username:password:UID:GID:comment:home:shell
```

- **UID:** User Identifier
  - **0:** reserved for root user
  - **1-999:** reserved by system for administrative and system users.
- **GID:** Group Identifier
  - **0:** reserved for root group
  - **1-99:** reserved by system for administrative and system groups.

#### NOTE

UID and GUI assignation policies are defined in the file `/etc/login.defs`

```

acamp0s@BCNLT5CG3284PRF:~$ cat /etc/login.defs | grep -i uid
# for private user groups, i. e. the uid is the same as gid, and username is
# Min/max values for automatic uid selection in useradd
UID_MIN                1000
UID_MAX                60000
#SYS_UID_MIN           100
#SYS_UID_MAX           999
# (examples: 022 -> 002, 077 -> 007) for non-root users, if the uid is
acamp0s@BCNLT5CG3284PRF:~$ cat /etc/login.defs | grep -i gid
# If you have a "write" program which is "setgid" to a special group
# In Debian /usr/bin/bsd-write or similar programs are setgid tty
# for private user groups, i. e. the uid is the same as gid, and username is
# Min/max values for automatic gid selection in groupadd
GID_MIN                1000
GID_MAX                60000
#SYS_GID_MIN           100
#SYS_GID_MAX           999
# the same as gid, and username is the same as the primary group name.

```

So you can trim the output using AWK or cut:

```
$ awk -F ":" '{ print $1 }' /etc/passwd
```

```
$ cut -d ":" -f1 /etc/passwd
```

#### NOTE

We will discuss passwords in the next sect, but expect to see an "x" in the password field of this file.

### 2.2.3 Understand the /etc/shadow file

Long ago, password hashes were stored in the /etc/passwd file. This file was world-readable, allowing inquisitive users to pull password hashes for other accounts from the file and run them through password-cracking utilities. Eventually, the password hashes were moved to a file readable only by root: /etc/shadow. Today, the password field in the /etc/passwd file is marked with an x.

#### NOTE

To know more about the passwords, the value we can see in the /etc/passwd is a hashed value, produced by the crypt function in Linux when we set the passphrase for the user using the passwd command. The hashed passphrase follows a specific format:

```
$id$salt$hashedpassword
```

- **id:** is the hashing method used when hashing the passphrase. For example, if the hash value is produced by yescrypt, the ID will be y, and 6 if the sha512crypt method is used. For a complete list of hashing methods and their ID, we can refer to the [Official Documentation](#).
- **salt:** adding random data to the input of a hash function to guarantee a unique output, the hash, even when the inputs are the same.
- **hashedpassword:** the password encrypted

## 2.2.4 Add User to Linux System

### 2.2.4.1 Basic Addition

```
$ sudo useradd [OPTIONS] user_name
```

When invoked, `useradd` creates a new user account according to the options specified on the command line and the default values set in the `/etc/default/useradd` file.

#### NOTE

When executed without any option, `useradd` creates a new user account using the default settings specified in the `/etc/default/useradd` file.

#### WARNING

Only root or users with sudo privileges can use the `useradd` command to create new user accounts.

To check the user has been correctly created

```
$ id username
```

To be able to log in as the newly created user, you need to set the user password

```
$ sudo passwd username
```

You will be prompted to enter and confirm the password. Make sure you use a strong password.

### 2.2.4.2 Addition with Home Directory Creation

On most Linux distributions, when creating a new user account with `useradd`, **by default the user's home directory is not created.**

Use the `-m` (`--create-home`) option to create the user home directory as `/home/username`

```
$ sudo useradd -m username
```

The command above creates the new user's home directory and copies files from `/etc/skel` directory to the user's home directory. If you list the files in the `/home/username` directory, you will see the initialization files:

- `.bash_logout`
- `.bashrc`
- `.profile`

```
acampos@BCNLT5CG3284PRF:~$ ls -la /etc/skel/
total 20
drwxr-xr-x  2 root root 4096 May  1 23:35 .
drwxr-xr-x 73 root root 4096 Oct  5 14:51 ..
-rw-r--r--  1 root root  220 Jan  6  2022 .bash_logout
-rw-r--r--  1 root root 3771 Jan  6  2022 .bashrc
-rw-r--r--  1 root root  807 Jan  6  2022 .profile
```

## NOTE

Within the home directory, the user can write, edit and delete all files and directories.

To create another different home

```
$ sudo useradd -d /custom/home username
```

### 2.2.4.3 Custom Shell

```
$ sudo useradd -s /custom/shell username
```

### 2.2.4.4 Addition Specifying the UID

By default, when a new user is created, the system assigns the next available UID from the range of user IDs specified in the `/etc/login.defs` file.

```
acampos@BCNLT5CG3284PRF:~$ cat /etc/login.defs | grep -i uid
# for private user groups, i. e. the uid is the same as gid, and username is
# Min/max values for automatic uid selection in useradd
UID_MIN                1000
UID_MAX                60000
#SYS_UID_MIN           100
#SYS_UID_MAX           999
# (examples: 022 -> 002, 077 -> 007) for non-root users, if the uid is
```

Invoke `useradd` with the `-u` (`--uid`) option to create a user with a specific UID. For example to create a new user named `username` with UID of 1500 you would type:

```
$ sudo useradd -u 1500 username
```

## WARNING

If the specified UID is already allocated to another user, you're alerted that the UID is unavailable and the operation aborts. Rerun it with a different UID number.

## NOTE

An easy way to look your user UID:

```
$ id -u username
```

### 2.2.4.5 Addition Specifying the GID

When creating a new user, the default behavior of the `useradd` command is to create a group with the same name as the username, and same GID as UID.

The `-g` (`--gid`) option allows you to create a user with a specific initial login group. You can specify either the group name or the GID number. The group name or GID must already exist.

```
$ sudo useradd -g group username
```

#### WARNING

If the specified group ID is already allocated to another group, you're alerted that the GID is unavailable and the operation aborts. Rerun it with a different group ID number.

#### NOTE

An easy way to look your user primary group:

```
$ id -gn username
```

### 2.2.4.6 Addition Assigning Multiple Groups

There are two types of groups in Linux operating systems Primary group and Secondary (or supplementary) group. Each user can belong to **exactly one primary** group and **zero or more secondary groups**.

You to specify a list of supplementary groups which the user will be a member of with the `-G (--groups)` option.

```
$ sudo useradd -g primary_group -G sec_group1, sec_group2, sec_group3 username
```

#### NOTE

An easy way to look all your user configuration:

```
$ id username
```

### 2.2.4.7 Addition with more Configurations

To create users with more different configurations, you can check the [documentation](#).

### 2.2.5 Switching User in Linux - su command

Su allows you to change the existing user to some other user. Use the `-l username` method to define a user account if you need to execute a command as someone other than root.

### 2.2.6 Delete User in Linux System

Deleting a user requires deleting both the user account as well as the files that are connected with the user account. With this command you do both:

```
$ userdel -r username
```



### WARNING

The above command deletes the user whose username is provided. Make sure that the user is not part of a group. If the user is part of a group then it will not be deleted directly, hence we will have to first remove him from the group and then we can delete him.

## 2.2.7 Manage User from Linux System

### Change the home directory

```
$ usermod -d new_home_directory_path username
```

### Change login name

```
$ sudo usermod -l new_login_name old_login_name
```

### Change the user password

```
$ passwd
```

### WARNING

It will require authentication with the old password, so you should know the old password to change it this way.

If you do not know the user password, you can use the sudo user to do it: **Change the user password**

```
$ sudo passwd username
```

### NOTE

Even further, you can force Linux user to change password at their next login, check the [documentation](#)

### Modify the UID of a user

```
$ usermod -u new_uid username
```

### Modify the group GID of a user

```
$ usermod -g new_gid username
```

## 2.2.8 Switch Users in Linux Terminal

Switch to sudo user:

```
$ sudo su
```

Switch to another user:

```
$ su - <user_name>
```

### 2.2.9 SSH Key-Based Authentication

Secure Shell (SSH) key-based authentication provides a more secure alternative to password-based authentication. Users generate a public-private key pair, where the public key is stored on the server and the private key is kept securely on the user's device.

With SSH key-based authentication, users like Lisa, a system administrator at CTechCo, can authenticate without entering a password. Instead, the server verifies the user's identity based on the possession of the private key.

To configure SSH key-based authentication for Lisa, the following steps can be taken:

- Generate an SSH key pair on Lisa's machine using the `ssh-keygen` command.
- Copy the public key to the server's `/home/lisasmith/.ssh/authorized_keys` file.
- Configure the server to allow SSH key-based authentication.

## 2.3 Managing Linux Groups

In Linux, groups are collections of users. Creating and managing groups is one of the simplest ways to deal with multiple users simultaneously, especially when dealing with permissions. It's more efficient to group user accounts with similar access requirements than to manage permissions on a user-by-user basis.

### 2.3.1 Understanding the `/etc/group` file

Similar to the `/etc/passwd` file above, the `/etc/group` file contains group account information. This information can be essential for troubleshooting, security audits, and ensuring users can access the resources they need.

The fields in the `/etc/group` file are:

```
groupname:password:GID:group members
```

### 2.3.2 Add a Group in Linux

To create a new group:

```
$ sudo groupadd group_name
```

To view the group you just added, run the command below:

```
$ cat /etc/group | grep -i group_name
```

#### NOTE

When a group is created, a unique group ID gets assigned to that group. You can verify that the group appears (and see its group ID) by looking in the `/etc/group` file.

If you want to create a group with a specific group ID (GID), use the `--gid` or `-g` option:

```
$ sudo groupadd -g 1009 group_name
```

### WARNING

If the specified group ID is already allocated to another group, you're alerted that the GID is unavailable and the operation aborts. Rerun it with a different group ID number.

#### 2.3.3 Change the group ID

You can change the group ID of any group with the `groupmod` command and the `--gid` or `-g` option:

```
$ sudo groupmod -g 1011 demo1
```

#### 2.3.4 Rename a group

You can rename a group using `groupmod` with the `--new-name` or `-n` option:

```
$ sudo groupmod -n test demo1
```

#### 2.3.5 How to Assign Users to Groups in Linux

Once a group is created, users can be added to it in the following way:

```
$ sudo usermod -aG group_name username
```

To check the user has been successfully added:

```
$ id username
```

#### 2.3.6 How to Delete Users from Groups in Linux

To remove a specific user from a group, you can use the `gpasswd` command to modify group information:

```
$ sudo gpasswd --delete username group_name
```

#### 2.3.7 How to delete a group

When a group is no longer needed, you delete it by using the `groupdel` command:

```
$ sudo groupdel demo
```

### 2.4 Sudo Command

The `sudo` command allows you to run programs as the root user. Using `sudo` instead of login in as root is more secure because you can grant limited administrative privileges to individual users without them

knowing the root password. **That's why the root user account in Ubuntu is disabled by default for security reasons, and users are encouraged to perform system administrative tasks using sudo.** The initial user created by the Ubuntu installer is already a member of the sudo group, so if you are running Ubuntu, chances are that the user you are logged in as is already granted with sudo privileges.

#### 2.4.1 Give sudo permissions with password

By default, on most Linux distributions granting sudo access is as simple as adding the user to the sudo group defined in the sudoers file. Members of this group will be able to run any command as root. The name of the **group** may differ from distribution to distribution.

- **sudo**: Debian, Ubuntu and derivatives groups for sudo permissions.

```
$ usermod -aG sudo username
```

- **wheel**: RedHat, CentOS and Fedora group for sudo permissions

```
$ usermod -aG wheel username
```

We can see the permission groups allowed in the `/etc/sudoers` file:

```
$ sudo cat /etc/sudoers
```

```
acampos@BCNLT5CG3284PRF:/home$ sudo tail -n 13 /etc/sudoers

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin   ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
```

So in my Linux System, any user added to the groups **sudo** or **admin** will have all sudo permissions enabled.

**What would happen with a user that is not in these groups?**

```
$ whoami
jiminy
$ id jiminy
uid=1001(jiminy) gid=1002(jiminy) groups=1002(jiminy)
$ sudo whoami
[sudo] password for jiminy:
jiminy is not in the sudoers file. This incident will be reported.
$ |
```

But if we add him to the admin group... Voilà!

```
acampos@BCNLT5CG3284PRF:/home$ sudo usermod -aG admin jiminy
acampos@BCNLT5CG3284PRF:/home$ su jiminy
Password:
$ whoami
jiminy
$ id jiminy
uid=1001(jiminy) gid=1002(jiminy) groups=1002(jiminy),115(admin)
$ sudo whoami
[sudo] password for jiminy:
root
$ |
```

#### NOTE

The password you have to introduce to access to sudo permissions is the user password, in this case, the jiminy password.

### 2.4.2 Give only some actions sudo permission

To allow a specific user to run only certain programs as sudo, instead of adding the user to the sudo group, add the users to the `/etc/sudoers` file. For example, to allow the user `linuxize` to run only the `mkdir` command:

#### WARNING

Modifying `/etc/sudoers` file can be very dangerous, so do it carefully. **Se puede liar muy parda!**

Advices:

- Use **sudo visudo**, because visudo will warn you that there's a syntax error and asks to undo the changes.
- It is hardly recommended to make a copy of the file before

```
$ sudo cp /etc/sudoers sudoerscopy
```

Just if you know what you're doing, keep going:

```
$ sudo visudo
```

```
/etc/sudoers
```

```
linuxize ALL=/bin/mkdir
```

#### NOTE

The file you are opening with `sudo visudo` is the `/etc/sudoers` file with the Nano editor! If you are not used to Nano, you can open it with vim. But vim does not notify you if there are some syntax error in the file, nano does.

```
$ sudo vim /etc/sudoers
```

### 2.4.3 How to enable sudo without entering a password

Modifying the `/etc/sudoers` adding the user with the following:

#### WARNING

Modifying `/etc/sudoers` file can be very dangerous, so do it carefully. **Se puede liar muy parda!**

Advices:

- Use **sudo visudo**, because visudo will warn you that there's a syntax error and asks to undo the changes.
- It is hardly recommended to make a copy of the file before

```
$ sudo cp /etc/sudoers sudoerscopy
```

```
$ sudo visudo
```

```
/etc/sudoers
```

```
jiminy ALL=(ALL:ALL) NOPASSWD: ALL
```

```
$ whoami
jiminy
$ sudo tail -2 /etc/sudoers
@includedir /etc/sudoers.d
jiminy ALL=(ALL:ALL) NOPASSWD: ALL
$ sudo ls
acampos jiminy
$ |
```

Also you can modify it in the group configuration:

```
/etc/sudoers
```

```
# Members of the admin group may gain root privileges without pwd
%admin  ALL=(ALL) NOPASSWD: ALL
```

#### 2.4.4 How to gain su permissions with sudo

If you try to redirect the output of a command to a file that your user has no write permissions, you will get a "Permission denied" error.

```
$ sudo echo "test" > /root/file.txt
# Output: bash: /root/file.txt: Permission denied
```

This happens because the redirection ">" of the output is performed under the user you are logged in, not the user specified with sudo. The redirection happens before the sudo command is invoked.

One solution is to invoke a new shell as root by using `sudo sh -c`:

```
$ sudo sh -c 'echo "test" > /root/file.txt'
```

Another option is to pipe the output as a regular user to the tee command , as shown below:

```
$ sudo echo "test" | sudo tee /root/file.txt
```

## 2.5 Su - root user

As we comment in the previous section, Using sudo instead of login in as root is more secure because you can grant limited administrative privileges to individual users without them knowing the root password. **That's why the root user account in Ubuntu is disabled by default for security reasons, and users are encouraged to perform system administrative tasks using sudo.**

### 2.5.1 First Approach - Enabling root account temporary

If you only need the root account for a particular task or job, run the following command and supply the super user password to authenticate the action with sudo.

```
$ sudo -i
```

What behind the scenes this command makes is to enable the root user only in the current shell (associated Linux process), and when this shell is shot down (for example typing exit) and you come back to you user, the root will be disabled again.

### 2.5.2 Enable root user

```
$ sudo -i passwd root
```

You will have to enter the new password for the root user and go ahead.

```
acampos@BCNLT5CG3284PRF:~$ users
acampos root
```

### 2.5.3 Disable root user

```
$ sudo passwd -dl root
```

```
acampos@BCNLT5CG3284PRF:/home$ sudo cat /etc/shadow
root:!:19635:0:99999:7:::
daemon:!:19478:0:99999:7:::
bin:!:19478:0:99999:7:::
root:!:19478:0:99999:7:::
```



## 3 Basic Linux Commands

### 3.1 Linux System Information

**Prints information about your machine's kernel, name, and hardware**

**The order is:** kernel name, network node hostname, kernel release, kernel version, machine hardware name, processor type, hardware platform and os.

```
$ uname -a
```

**To check Linux Distro**

```
$ uname -or
```

**To know more about Linux Distro**

```
$ cat /etc/os-release
```

```
$ cat /etc/lsb-release
```

```
$ lsb_release -a
```

```
$ hostnamectl
```

### 3.2 Change Linux hostname

Check the current hostname:

```
$ hostname
```

```
$ hostnamectl
```

```
$ less /etc/hostname
```

Change it:

```
$ hostnamectl set-hostname <new_hostname>
```

#### NOTE

You can do it manually:

- `hostname <new_hostname>`
- Replace all old occurrences by new hostname: `vim /etc/hostname`
- Replace all old occurrences by new hostname: `vim /etc/hosts`

### 3.3 Linux System Consumption

Displays running processes and the system's resource usage

```
$ top
```

Do the same but with some colorfull graphics

```
$ htop
```

Displays the system's overall disk space usage

```
$ df -h
```

Displays a folder/file disk space usage

```
$ df -h folder_name
```

Checks a file or directory's storage consumption

```
$ du -h directory_name
```

#### NOTE

-h flag shows an humanize output, it specifies K, M, G, etc. If you want to have the output in Kibibyte do not use the flag.

### 3.4 Linux Services

#### 3.4.1 ps

ps displays information about a selection of the active processes. If you want a repetitive update of the selection and the displayed information, use top instead. **Display a snapshot of the currently running processes**

```
$ ps waux
```

- **w:** display the full command line of each process
- **a:** list processes from all users
- **u:** detailed information about each process
- **x:** list processes without controlling terminals

#### 3.4.2 systemctl

**Systemctl** manages systemd services and other units such as sockets, devices, mount points, and timers in Linux systems. It is a powerful command-line tool used to query or send control commands to the system manager.

**Systemd** is a system and service manager for Linux operating systems

#### Start a Service

```
$ systemctl start <service_name>
```

#### Stop a Service

```
$ systemctl stop <service_name>
```

#### Check the status of a Service

```
$ systemctl status <service_name>
```

**Restart a Service:** if the service is very critical, check `systemctl reload`

```
$ systemctl restart <service_name>
```

**Reload configuration of a Service:** very useful for critical services that cannot be restarted, it is like a soft restart.

```
$ systemctl reload <service_name>
```

#### Enable a service to start on Boot

```
$ systemctl enable <service_name>
```

#### Disable a service to start on Boot

```
$ systemctl disable <service_name>
```

#### Show the running services

```
$ systemctl --type=service --state=running
```

### 3.5 Service

The **service** command is commonly used in older versions of Linux distributions or those that haven't fully adopted systemd yet. It provides a simpler interface compared to `systemctl`, but it offers similar functionality for starting, stopping, restarting, and checking the status of services.

It is very similar to **systemctl**, with almost same functionalities, just reversing arguments

#### Start a Service

```
$ service <service_name> start
```

#### Check the status of a Service

```
$ service <service_name> status
```

## 3.6 Journalctl

journalctl and systemctl are command-line utilities in Linux systems that are commonly used for managing and monitoring system services and logs. They are part of the systemd suite of tools, which is a system and service manager for Linux operating systems.

**journalctl** is a command-line tool used to query and display logs from the systemd journal. The systemd journal is a component of systemd that collects and stores log messages from the kernel, services, and other system components in a structured binary format. journalctl allows you to view and filter logs collected by the systemd journal. This includes logs from various services, system messages, kernel logs, and more.

### 3.6.1 Usage

Display all available logs:

```
$ journalctl
```

Display logs from current boot:

```
$ journalctl -b
```

Display logs from previous boot:

```
$ journalctl -b -1
```

Display logs associated to a specific service:

```
$ journalctl -u <service_name>
```

Display latest logs:

```
$ journalctl -e
```

Detailed logs:

```
$ journalctl -x
```

Follow the logs in real time:

```
$ journalctl -f
```

Show logs since:

```
$ journalctl --since "YYYY-MM-DD HH:MM:SS"
```

Show logs until:

```
$ journalctl --until "YYYY-MM-DD HH:MM:SS"
```

Filter logs based on priority

```
$ journalctl -p <priority>
```

```
$ journalctl -p err
```

#### Note

Most used:

```
$ journalctl -xeu <service_name>
```

But remember that most relevant logs **are in the specific log file of the service.**

## 3.7 Moving between directories

### 3.7.1 Introduction

- `pwd`: which directory we are
- `cd == cd ~` : go to users `$HOME`
- `cd /` : go to `/`
- `cd .` : does not do anything
- `cd ..` : go one directory back
- `ls -l` : list all directory content (not hidden)
- `ls -la` : list all directory content (hidden or not)

### 3.7.2 `ls -la` Analyzing Results

The first character on the left represents the type, possible values for this position are as follows:

- `-`: file
- `d`: directory
- `l`: symbolic link
- `b`: binary (executable file)

The following 9 represent the file permissions and should be viewed in groups of 3.

- Los **first 3** owner permissions
- Los **3 in the middle** group permissions
- Los **last 3** rest of the world permissions
- `r`: read
- `w`: write
- `x`: execute

## Note

The `/etc/group` is a text file which defines the groups to which users belong under Linux and UNIX operating system. Under Unix / Linux multiple users can be categorized into groups. Unix file system permissions are organized into three classes, user, group, and others. The use of groups allows additional abilities to be delegated in an organized fashion, such as access to disks, printers, and other peripherals

### 3.8 chmod

You may need to know how to change permissions in numeric code in Linux, so to do this you use numbers instead of "r", "w", or "x". Cause Basically, you add up the numbers depending on the level of permission you want to give.

- **0:** No Permission
- **1:** Execute
- **2:** Write
- **4:** Read

Examples:

- To give read, write, and execute permissions for everyone:

```
$ chmod 777 folder/file_name
```

- To give read, write, and execute permissions for the owner user only.

```
$ chmod 700 folder/file_name
```

- To give write and execute (3) permission for the owner user, w (2) for the group, and read, write, and execute for the rest of users.

```
$ chmod 321 foldername
```

Permiso	Valor	Descripción
rw- -- -- --	600	El propietario tiene permisos de lectura y escritura
rw- --x --x	711	El propietario lectura, escritura y ejecución, el grupo y otros solo ejecución
rw- r-x r- --x	755	El propietario lectura, escritura y ejecución, el grupo y otros pueden leer y ejecutar el archivo
rw- rw- --x	777	El archivo puede ser leído, escrito y ejecutado por quien sea
r- -- -- --	400	Solo el propietario puede leer el archivo, pero ni el mismo puede modificarlo o ejecutarlo y por supuesto ni el grupo ni otros pueden hacer nada en el
rw- r- --	640	El usuario propietario puede leer y escribir, el grupo puede leer el archivo y otros no pueden hacer nada

Figure 1: Permissions

### 3.9 chown

The **chown** command allows you to change the user and/or group ownership of a given file, directory, or symbolic link.

In Linux, all files are associated with an owner and a group and assigned with permission access rights for the file owner, the group members, and others.

```
$ chown [OPTIONS] USER[:GROUP] FILE(s)
```

**USER** is the user name or the user ID (UID) of the new owner. **GROUP** is the name of the new group or the group ID (GID). **FILE(s)** is the name of one or more files, directories or links.

#### NOTE

Numeric IDs should be prefixed with the **+** symbol.

For example, the following command will change the ownership of a file named `file1` to a new owner named `linuxize`:

```
$ chown linuxize file1
```

To change recursively

```
$ chown -R linuxize /var/www/
```

#### NOTE

If the directory contains symbolic links you should add the flag `-h`

```
$ chown -hR linuxize /var/www/
```

**Using a Reference File** The `--reference=ref_file` option allows you to change the user and group ownership of given files to be same as those of the specified reference file (**ref\_file**).

```
$ chown --reference=REF_FILE FILE
```

#### WARNING

If the reference file is a symbolic link `chown` will use the user and group of the target file.

### 3.10 Ctrl+R

With this command we can search in the history the past commands we have run on the machine.

Be carefull using `Ctrl+R`, because for example if you click on `tab`, you will crash it:

- **Intro:** to execute the command it is showing you
- **Ctrl+R:** to go to the command behind that matches the pattern

### 3.11 Mkdir (create files)

- `mkdir -p`: create a directory recursively, with different subfolders (`mkdir -p first/second/third...`)
- `mkdir -m a=rwx`: create a directory which files are going to have the permissions specified
- `rm -rf <file>`: remove recursively the file
- `rm *.txt`: remove all the files which contains the string "txt" on the current directory

### 3.12 Mv

Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

```
$ mv path/to/source path/to/dest
```

- `-t`: Moving different files to DEST

```
$ mv -t path/to/dest path/to/source_1 path/to/source_2 ... path/to/source_n
```

- `-i`: prompts the user before overwriting an existing file.
- `-f`: forces the move operation without prompting for confirmation, even if it means overwriting files.
- `-n`: prevents overwriting existing files.
- `-v`: verbose.
- **Move content from one folder to another**

```
$ mv path/to/source/* /path/dest/folder/
```

### 3.13 Cp

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

```
$ cp path/to/source path/to/dest
```

- `-r / -R / --recursively`: Recursively.

```
$ cp -r path/to/source path/to/dest
```

- `-t`: Copying different files to DEST.

```
$ cp -t path/to/dest path/to/source_1 path/to/source_2 ... path/to/source_n
```

- `-p`: preserves the original file's attributes, such as the timestamp, ownership, and permissions.
- `-f`: forces the copy operation without prompting for confirmation, even if it means overwriting files.
- `-n`: prevents overwriting existing files.



- `-v`: verbose.
- **Copy all the content from one folder to another:**

```
$ cp -r path/to/source/* /path/dest/folder/
```

### 3.14 Rsync

Rsync is a fast and extraordinarily versatile file copying tool. It can copy locally, to/from another host over any remote shell, or to/from a remote rsync daemon. It offers a large number of options that control every aspect of its behavior and permit very flexible specification of the set of files to be copied. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

```
$ rsync -av /path/to/source/ /path/to/destination/
```

- `-a`: enables archive mode, which preserves symbolic links, file permissions, user and group ownerships, and timestamps.
- `-v`: verbose.
- `-P`: displays the progress of the transfer, including the percentage of the transfer completed, transfer speed, and estimated time remaining. Also keep partially transferred files if the transfer is interrupted.
- `--partial`: keep partially transferred files if the transfer is interrupted.
- `--progress`: displays the progress of the transfer, including the percentage of the transfer completed, transfer speed, and estimated time remaining.
- `--delete`: deletes files in the destination directory that are not present in the source directory. This keeps the destination directory in exact sync with the source.
- **From localhost to remote host:**

```
$ rsync -avzP /path/to/local/source \
  remote_user@remote_ip:/path/to/remote/destination
```

- **From remote host to localhost:**

```
$ rsync -avzP remote_user@remote_ip:/path/to/remote/source \
  /path/to/local/destination
```

#### Note

You can specify the key to use: `-e "ssh -i /path/to/private_key"`

### 3.15 Simbolik Links

#### 3.15.1 What is a Symbolic Link?

A symbolic link (also known as a symlink or soft link) is a type of file that points to another file or directory in the Linux filesystem. Unlike hard links, which reference the data on the disk directly, a symbolic link contains a path that points to the target file or directory.

### 3.15.2 Why Use Symbolic Links?

Convenience: They allow you to create shortcuts to files and directories. Flexibility: If the target file is moved, you can update the symlink without modifying the files that use it. Resource Efficiency: They can link to directories and files across different filesystems.

### 3.15.3 Symbolic Links Management

The `ln` command is used to create links between files. The `-s` option creates a symbolic link. The basic syntax is:

```
$ ln -s [target] [link_name]
```

The `unlink` command is used to remove symbolic links. It only works with single files, not directories.

```
$ unlink [link_name]
```

#### Note

Alternatively, you can use the `rm` command to remove symbolic links. This can be used for both files and directories.

```
$ rm -rf [link_name]
```

## 3.16 Aliases

To know all the aliases we have in our system

```
$ alias
```

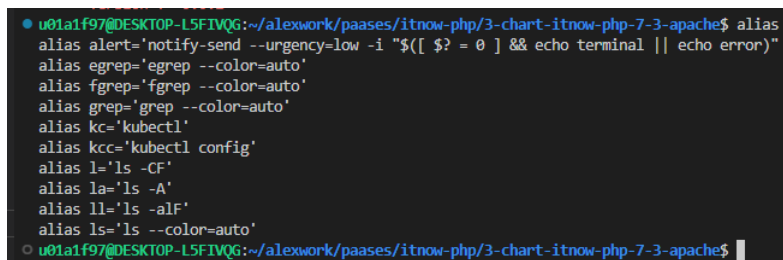
A terminal window screenshot showing a list of default aliases. The prompt is 'u01a1f97@DESKTOP-L5FIWQ6:~/alexwork/paases/itnow-php/3-chart-itnow-php-7-3-apache\$'. The aliases listed are: alert='notify-send --urgency=low -i "\${[ \$? = 0 ]} && echo terminal || echo error"', egrep='egrep --color=auto', fgrep='fgrep --color=auto', grep='grep --color=auto', kcc='kubectl', kcc='kubectl config', l='ls -CF', la='ls -A', ll='ls -aLF', and ls='ls --color=auto'. The prompt at the bottom is 'u01a1f97@DESKTOP-L5FIWQ6:~/alexwork/paases/itnow-php/3-chart-itnow-php-7-3-apache\$'.

Figure 2: Default aliases

### 3.16.1 Ephemeral Aliases

To generate an ephemeral alias:

```
$ alias <alias_name>="command arg1 arg2 ..."
```

#### Examples

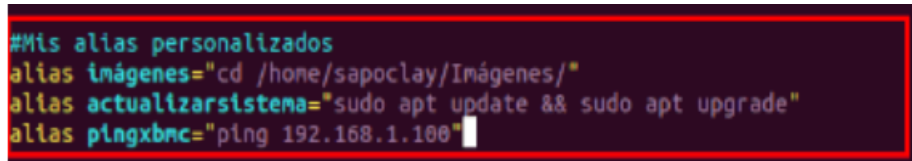
```
$ alias docker=podman
```

```
$ alias kcc="kubectl config"
```

### 3.16.2 Permanent Aliases

To maintain alias in our system and do not lose them when we change of terminal we need to configure them on the shell config file we use:

- Bash → ~/.bashrc
- ZSH → ~/.zshrc
- Fish → ~/.config/fish/config.fish

A screenshot of a terminal window with a dark background and a red border. It shows a list of custom aliases in Spanish. The first line is a comment: "#Mis alias personalizados". The following lines are: "alias imágenes="cd /home/sapoclay/Imágenes/"", "alias actualizar sistema="sudo apt update && sudo apt upgrade"", and "alias pingxbmc="ping 192.168.1.100"". A cursor is visible at the end of the last line.

```
#Mis alias personalizados
alias imágenes="cd /home/sapoclay/Imágenes/"
alias actualizar sistema="sudo apt update && sudo apt upgrade"
alias pingxbmc="ping 192.168.1.100"
```

Figure 3: Alias in permanent files

## 3.17 Source

```
$ source filename [arguments]
```

When you execute a command or a script from a shell, a subprocess (child process) of the shell is created to execute the command or script (parent process).

If the script that executes the child process creates or modifies any environment variable, those changes or variables disappear when the command or script finishes.

If we want those changes to remain, we can use the Bash command source. This command causes the process or command to execute without creating any child process, so that changes made to environment variables and others are maintained when the file finishes.

More information: [Source in Bash](#)

## 3.18 Shutdown machines or terminals

- Shutdown the linux machine now

```
$ sudo shutdown -r now
```

```
$ sudo reboot
```

- Just shutdown the terminal

```
$ exec "$SHELL"
```

### 3.19 Others

To create a new empty file

```
$ touch main.py
```

Checks a file's type

```
$ file main.py
```

Prints command outputs in Terminal and a file

```
$ echo "Hello" | tee output.txt
```

[More basic interesting linux commands](#)

## 4 Package Management Systems

Package management systems in Linux handles the installation, updating, configuration, and removal of software packages. Each distribution has its own package management system, but they generally follow similar principles.

Key components of all package management systems:

- **Repositories:** Central locations where software packages are stored and maintained
- **Packages:** Precompiled binaries or source code bundled with metadata, dependencies, and scripts necessary for installation.
- **Package Managers:** Tools that automate the process of handling packages.

### 4.1 Apt - Debian

Apt (Advanced Package Tool) package management system is built on top of the dpkg system, which is the underlying package management system for Debian-based distributions. It uses **APT Repositories**, servers that store software packages and provide metadata about those packages. Ubuntu repositories are organized into different sections like main, universe, restricted, and multiverse.

#### 4.1.1 Basic apt commands

Update the local package index with the latest changes in the repositories:

```
$ apt update
```

Upgrades all installed packages to the latest versions available in the repositories:

```
$ apt upgrade
```

Simulates the upgrade:

```
$ apt upgrade --dry-run
```

Upgrade just a specific package:

```
$ apt install --only-upgrade <package-name>
```

Install the latest available version of a package:

```
$ apt install <package_name>
```

Install a specific version of a package:

```
$ apt install <package_name>=<version>
```

```
$ apt install vim=2:8.1.2269-1ubuntu5
```

List all installed packages and their versions on the system:

```
$ apt list --installed
```

Look for specific package installed and its version:

```
$ apt list --installed | grep <package_name>
```

Remove and purge package:

```
$ apt remove <package_name>
```

```
$ apt purge <package_name>
```

Remove unnecessary packages:

```
$ apt clean
```

```
$ apt autoremove
```

Look for the available versions of a package:

```
$ apt update
```

```
$ apt-cache madison <package_name>
```

Search for packages that match the provided keyword:

```
$ apt-cache search <package_name>
```

Show package details:

```
$ apt-cache show <package_name>
```

Check for broken dependencies:

```
$ apt check
```

### 4.1.2 Managing Repositories

The operating system creates two files to manage repositories:

- `/etc/apt/sources.list`: main configuration file for APT repositories. It contains a list of repository URLs where APT can find packages and updates. Mainly it includes official Ubuntu repositories that match your Ubuntu version and security repositories.
- `/etc/apt/sources.list.d/`: repository updated by some packages when installed or updated to ensure their software is updated from their repositories.

```

• X acampos@BCNLT5C63284PRF ~/personal/latex/latex-linux_administrator updates ± ls -la /etc/apt/sources.list.d/
total 28
drwxr-xr-x 2 root root 4096 Jun 11 12:48 .
drwxr-xr-x 8 root root 4096 Jul 10 12:51 ..
-rw-r--r-- 1 root root 150 Jun 11 12:48 ansible-ubuntu-ansible-jammy.list
-rw-r--r-- 1 root root 63 Jun 11 12:48 apt_releases_hashicorp_com.list
-rw-r--r-- 1 root root 76 Jun 11 12:48 azure-cli.list
-rw-r--r-- 1 root root 71 Jun 11 12:48 docker.list
-rw-r--r-- 1 root root 71 Jun 11 12:48 download_docker_com_linux_ubuntu.list
• acampos@BCNLT5C63284PRF ~/personal/latex/latex-linux_administrator updates ± cat /etc/apt/sources.list.d/docker.list
deb [arch=amd64] https://download.docker.com/linux/ubuntu jammy stable

```

```

• acampos@BCNLT5C63284PRF ~/personal/latex/latex-linux_administrator updates ± cat /etc/apt/sources.list
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://archive.ubuntu.com/ubuntu/ jammy main restricted
# deb-src http://archive.ubuntu.com/ubuntu/ jammy main restricted

# # Major bug fix updates produced after the final release of the
# # distribution.
deb http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted
# deb-src http://archive.ubuntu.com/ubuntu/ jammy-updates main restricted

# # N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
# # team. Also, please note that software in universe WILL NOT receive any
# # review or updates from the Ubuntu security team.
deb http://archive.ubuntu.com/ubuntu/ jammy universe
# deb-src http://archive.ubuntu.com/ubuntu/ jammy universe
deb http://archive.ubuntu.com/ubuntu/ jammy-updates universe
# deb-src http://archive.ubuntu.com/ubuntu/ jammy-updates universe

# # N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
# # team, and may not be under a free licence. Please satisfy yourself as to
# # your rights to use the software. Also, please note that software in
# # multiverse WILL NOT receive any review or updates from the Ubuntu
# # security team.
deb http://archive.ubuntu.com/ubuntu/ jammy multiverse
# deb-src http://archive.ubuntu.com/ubuntu/ jammy multiverse
deb http://archive.ubuntu.com/ubuntu/ jammy-updates multiverse

```

When new repositories are added, their GPG keys are fetched and added to the keyring `/etc/apt/trusted.gpg.d/`. This ensures that APT can verify the integrity and authenticity of the packages from these repositories.

By using `/etc/apt/sources.list` and `/etc/apt/sources.list.d/`, you can efficiently manage where your Ubuntu system looks for software packages and updates, ensuring that you have access to the latest and most secure versions of the software you need.

### Warning

However they can fuck up your `apt update` because some of the repositories have been changed or the GPG key is not compatible. So you need to:

- If the package is smart enough, just:
  - `sudo rm /etc/apt/sources.list.d/<package_name>.list`
  - `sudo apt update`
- If not:
  - `sudo rm /etc/apt/sources.list.d/<package_name>.list`
  - Add the new GPG key and install again the repository.
  - `sudo apt update`

Repositories are added automatically, but to remove them:

```
$ sudo rm /etc/apt/sources.list.d/<package_name>.list
```

## 4.2 Dnf - RedHat

### 4.2.1 Yum or Dnf?

**Yum** (Yellowdog Updater, Modified) is an older package manager used for RPM-based distributions. It is known for its less efficient dependency resolution algorithm compared to **dnf**, which can result in slower performance, particularly with large repositories and metadata. While yum provides essential package management functions, it lacks some of the advanced features and flexibility found in dnf. However, it does offer backwards compatibility with older plugins and configurations, making it suitable for legacy systems.

On the other hand, **dnf** (Dandified Yum) is the modern successor to yum, introduced in Fedora 18 and later adopted by RHEL 8. It uses the libsolv library for faster and more efficient dependency resolution, significantly improving performance, especially with large repositories. dnf boasts advanced features, including better transaction handling, enhanced debugging capabilities, and improved support for extensions. It also supports modularity, allowing users to install different versions of software alongside each other. Despite its advancements, dnf maintains a high level of compatibility with yum commands, easing the transition for users.

### 4.2.2 Basic dnf commands

Update the local package index with the latest changes in the repositories:

```
$ dnf check-update
```

Upgrades all installed packages to the latest versions available in the repositories:

```
$ dnf update
```

Or which is the same:

```
$ dnf upgrade
```

Install the latest available version of a package:

```
$ dnf install <package_name>
```

Install a specific version of a package:

```
$ dnf install <package_name>-<version>
```

List all installed packages and their versions on the system:

```
$ dnf list installed
```

Look for specific package installed and its version:

```
$ dnf list installed | grep <package_name>
```

Look for the available versions of a package:



```
$ dnf list --available <package-name>
```

```
$ dnf info <package-name>
```

```
$ dnf --showduplicates list <package-name>
```

Update a specific package:

```
$ dnf update <package_name>
```

Remove a specific package and clean up dependencies:

```
$ dnf remove package_name
```

```
$ dnf autoremove
```

Remove unnecessary packages and dependencies:

```
$ dnf autoremove
```

```
$ dnf clean all
```

Check for broken dependencies:

```
$ dnf check
```

List enabled repositories:

```
$ dnf repolist
```

Enable/Disable Repositories:

```
$ dnf config-manager --set-enabled <repository_id>
```

```
$ dnf config-manager --set-disabled <repository_id>
```

## 5 .bashrc, .bash\_profile, /etc/bashrc & path

### 5.1 .bashrc vs .bash\_profile

- **.bashrc [non interactive login]:** for changes to take effect, we need to open another interactive terminal (changes don't apply to the current one because this file is executed upon starting a new terminal session).
- **.bash\_profile [interactive login]:** it only executes when there's a login process, upon restarting the machine, when using ssh, sudo... But not when opening a new terminal.

### 5.2 PATH

**PATH** is an environmental variable in Linux that **tells the shell which directories to search for executable files** (i.e., ready-to-run programs) in response to commands issued by a user.

When you type a command into the command prompt in Linux, all you're doing is telling it to run a program. Even simple commands, like `ls`, `mkdir`, `rm`, and others are just small programs that usually live inside a directory on your computer called `/usr/bin`. Other places to search for executables: `/usr/local/bin`, `/usr/local/sbin`, and `/usr/sbin`.

When you type a command into your Linux shell, it doesn't look in every directory to see if there's a program by that name. It only looks to the ones you specified in the `$PATH` environment var.

Sometimes, you may wish to install programs into other locations on your computer, but be able to execute them easily without specifying their exact location. You can do this easily by adding a directory to your `$PATH`.

View your PATH

```
$ echo $PATH
```

### 5.3 Set your PATH

#### 5.3.1 For current terminal

```
$ export PATH=$PATH:/place/to/the/binary/file
```

#### 5.3.2 Permanently for interactive sessions

But what happens if you restart your computer or create a new terminal instance? Your addition to the path is gone! This is by design. The variable `$PATH` is set by your shell every time it launches. The exact way to do this depends on which shell you're running.

Add the following line to `~/.bash_profile`, `~/.bashrc`, or `/.profile`

```
$ export PATH=$PATH:/place/with/the/file
```

#### Nota

Be very careful editing these files, so one error in the configuration of these files can make some binaries crash (example: `mkdir`, `ls`, etc.)

## 6 Useful Tools

### 6.1 Cat

Cat will allow us to view the contents of files without opening them, create files if they do not exist or redirect terminal outputs.

- **Basic usage:** show the content of a file

```
$ cat test
```

- Show the content of both files test and test2

```
$ cat test1 test2
```

- Adds the content of test2 to the file test1, if there was content it overwrites, if there was nothing, it creates the new file

```
$ cat test1 > test2
```

- Appends the content of test2 into the file test1, if there is no test2, it will create it

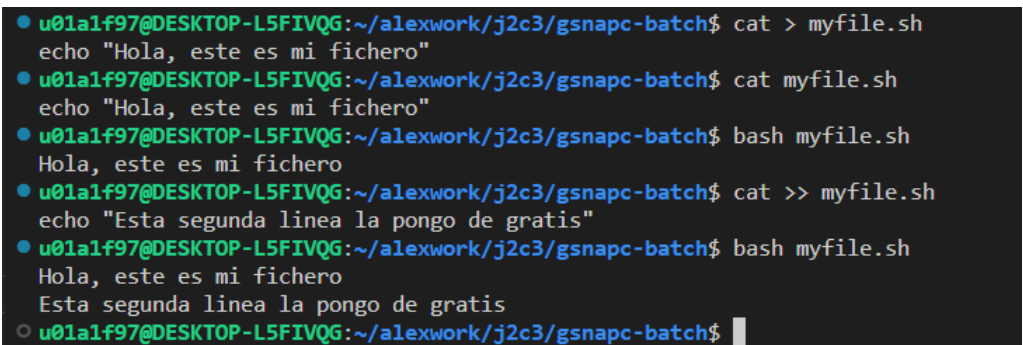
```
$ cat test1 >> test2
```

- Show line number

```
$ cat -n file
```

- Write in a file the content we want, as we were with text editor (Exit: Ctrl+D)

```
$ cat > filename
```

A terminal window screenshot showing a series of commands and their outputs. The user is in a directory ~/alexwork/j2c3/gsnapc-batch. The commands and outputs are: 1. cat > myfile.sh followed by echo "Hola, este es mi fichero". 2. cat myfile.sh followed by echo "Hola, este es mi fichero". 3. bash myfile.sh followed by the output "Hola, este es mi fichero". 4. cat >> myfile.sh followed by echo "Esta segunda linea la pongo de gratis". 5. bash myfile.sh followed by the output "Hola, este es mi fichero" and "Esta segunda linea la pongo de gratis". The prompt is u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch\$.

```
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$ cat > myfile.sh
echo "Hola, este es mi fichero"
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$ cat myfile.sh
echo "Hola, este es mi fichero"
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$ bash myfile.sh
Hola, este es mi fichero
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$ cat >> myfile.sh
echo "Esta segunda linea la pongo de gratis"
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$ bash myfile.sh
Hola, este es mi fichero
Esta segunda linea la pongo de gratis
u01a1f97@DESKTOP-L5FIVQG:~/alexwork/j2c3/gsnapc-batch$
```

Figure 4: amazing cat

#### 6.1.1 EOF

The best way to add multi-line content with cat is using **EOF**:

Directly from the terminal and replace old content:

```
$ cat > filename << EOF
$ heredoc > insert here
$ heredoc > any line of content
$ heredoc > you want
$ heredoc > EOF
```

```
acampos@BCNLT5CG3284PRF ~/work cat > test.txt << EOF
heredoc> Hello
heredoc> This is a test using EOF and cat
heredoc> Thanks
heredoc> EOF
acampos@BCNLT5CG3284PRF ~/work cat test.txt
Hello
This is a test using EOF and cat
Thanks
```

Directly from the terminal and append content:

```
$ cat >> filename << EOF
$ heredoc > insert here
$ heredoc > any line of content
$ heredoc > you want
$ heredoc > EOF
```

From copy paste:

```
$ cat > filename << EOF
insert here
any line of content
you want and copy this
into your terminal
EOF
```

```
acampos@BCNLT5CG3284PRF ~/work cat > filename << EOF
insert here
any line of content
you want and copy this
into your terminal
EOF
acampos@BCNLT5CG3284PRF ~/work cat filename
insert here
any line of content
you want and copy this
into your terminal
```

One good example of usage:

```
export LINK_INTERFACE="enp1s0f1"
export VLAN_ID="2002"
sed -i '$d' /etc/netplan/00-installer-config.yaml # To remove the last line of the file
cat >> /etc/netplan/00-installer-config.yaml << EOF
    ${LINK_INTERFACE}: {}
    vlans:
    vlan${VLAN_ID}:
        dhcp4: yes
        id: ${VLAN_ID}
        link: ${LINK_INTERFACE}
    version: 2
EOF
```

## 6.2 Head & Tail

**Head:** print the first 10 lines of each FILE to standard output

- Print the first NUM lines instead of the first 10

```
$ head -n <number_of_lines> <filename>
```

**Tail:** print the last 10 lines of each FILE to standard output.

- Print the last NUM lines instead of the last 10

```
$ head -n <number_of_lines> <filename>
```

- Attach the terminal following the evolution of the file

```
$ head -f <filename>
```

## 6.3 Diff (File comparator)

Command to compare FILES line by line.

**Basic usage**

```
$ diff /path/to/file1 /path/to/file2
```

**Output side by side (2 columns)**

```
$ diff -y /path/to/file1 /path/to/file2
```

**Recursively**

```
$ diff -r /path/to/directory1 /path/to/directory2
```

**Report only when files differ**

```
$ diff -q /path/to/directory1 /path/to/directory2
```

**Report when two files are the same**

```
$ diff -s /path/to/directory1 /path/to/directory2
```

**Ignore case**

```
$ diff -i /path/to/directory1 /path/to/directory2
```

**Ignore changes in the amount of white space**

```
$ diff -b /path/to/directory1 /path/to/directory2
```

**Ignore all white space**

```
$ diff -w /path/to/directory1 /path/to/directory2
```

**Ignore changes where lines are all blank**

```
$ diff -B /path/to/directory1 /path/to/directory2
```

**Output NUM (default 3) lines of copied context**

```
$ diff -c /path/to/directory1 /path/to/directory2
```

## 6.4 Grep (Global search for Regular Expressions and Print out)

### Nota

grep, egrep, fgrep, rgrep - print lines that match patterns. In addition, the variant programs egrep, fgrep and rgrep are the same as grep -E, grep -F, and grep -r, respectively. These variants are deprecated, but are provided for backward compatibility.

### 6.4.1 Two flavors of grep usage

1. Use grep to search text in a file / files

```
$ grep "texto-buscado" <archivo/archivos>
```

2. Use grep to search text in the Terminal STDOUT

```
$ COMMAND [args...] | grep "texto-buscado"
```

The result of this is the occurrences of the pattern (by the line it is on) in the / stdout files. If there is no match, no output will be printed to the terminal.

### Note

The "" are necessary to convert to String. If what we introduce is already a String, they would not be necessary. Example:

- `grep alias file.txt`: We dont need ""
- `grep "alias pepito" file.txt`: We need ""

#### 6.4.2 Useful Flags

##### line Number

```
$ grep -n ...
```

##### Count the number of coincident lines

```
$ grep -c ...
```

##### IgnoreCase

```
$ grep -i ...
```

##### Execute in silent mode, suppress all normal output

```
$ grep -q ...
```

##### Accept extended regular expressions

```
$ grep -E ...
```

##### Use PATTERNS for matching, you can specify multiple patterns to search for

```
$ grep -e 'pattern1' -e 'pattern2' file.txt
```

##### Match only full words

```
$ grep -w
```

##### Match only full lines

```
$ grep -x
```

##### Print non-matching lines

```
$ grep -v
```

##### Before Context or Aftercontext

```
$ grep "text" file -A NUMBER_A -B NUMBER_B
```

**Recursive Search** By default, grep cannot search for directories. If you try to do so, you will get an error ("It is a directory"). With the -R option, searching for files across directories and subdirectories becomes possible.

```
$ grep -R
```

**Print only names of FILEs with selected lines**

```
$ grep -l
```

**Print only names of FILEs with no selected lines**

```
$ grep -L
```

## 6.5 Find

Command to search for files in a directory hierarchy.

**Find all files and directories from "/" with the name "linux"**

```
$ find / -name "linux"
```

**Find all files and directories from "/" with the name "linux" and list them**

```
$ find / -name "linux" -ls
```

**Find just files"**

```
$ find / -type f -name "linux"
```

**Find just directories"**

```
$ find / -type d -name "linux"
```

**Find from our current directory all the files which name contains the substring "Thr"**

```
$ find . -name "*Thr*"
```



```
[default@iks03-iksphpitnowr-pre-59f9b86968-hlj2k ~]$ find /var/template/ -name "*envvar*" -ls
```

8	4	-rw-r--r--	1	100	1000	423	Jan	4	14:46	/var/template/batch-db3-envvar
7	4	-rw-r--r--	1	100	1000	423	Jan	4	14:46	/var/template/mysql-db3-envvar
6	4	-rw-r--r--	1	100	1000	348	Jan	4	14:46	/var/template/postgresql-envvar
5	4	-rw-r--r--	1	100	1000	600	Jan	4	14:46	/var/template/oracle-envvar
4	4	-rw-r--r--	1	100	1000	179	Jan	4	14:46	/var/template/mysql-envvar
3	4	-rw-r--r--	1	100	1000	179	Jan	4	14:46	/var/template/db2-envvar

```
[default@iks03-iksphpitnowr-pre-59f9b86968-hlj2k ~]$
```

Figure 5: use of find

## NOTE

The "" are optional, because they are just to define strings (process spaces and weird characters as part of the string)

## 6.6 Sed (Stream Editor)

It is a very essential tool for text processing, specifically it is common used for text replacement. Con Sed podemos editar archivos, incluso sin abrirlos, de manera individual o masiva. Dicho sea, que esta forma, es mucho más rápida para encontrar y reemplazar algo en un archivo de manera manual.

Print the content of "fichero.txt" replacing the first occurrence of the string "Microsoft Windows" by "GNU Linux" in the file "fichero.txt"

```
$ sed "s/Microsoft Windows/GNU Linux/" fichero.txt
```

Save the changes in the file:

```
$ sed -i "s/Microsoft Windows/GNU Linux/" fichero.txt
```

Replace all the occurrences of the string "Microsoft Windows" by "GNU Linux" in the file "fichero.txt" and save.

```
$ sed -i "s/Microsoft Windows/GNU Linux/g" fichero.txt
```

Replace just the 3rd occurrence of "Microsoft Windows" by "GNU Linux" in the text and save

```
$ sed "s/Microsoft Windows/GNU Linux/3g" fichero.txt
```

Replace the string "Microsoft Windows" just in line 1:

```
$ sed "1 s/Microsoft Windows/GNU Linux/" fichero.txt
```

## NOTE

By default sed is **case SENSITIVE** but to change it by case insensitive, you can add I at the end of the REGEX:

```
$ sed -i "s/Microsoft Windows/GNU Linux/gI" fichero.txt
```

## WARNING

Sed will replace all the string matches, without differentiating if it is completed or partial, for example:

```
acampos@BCNLT5CG3284PRF:~$ echo "Alex and Alexandra go to Pedraforca" | sed 's/Alex/Helena/g'  
Helena and Helenaandra go to Pedraforca
```

To replace the exact match you should use spaces, in **REGEX** `space=\s`:

```
$ sed "s/Alex\s/Claudia /g"
```

For more information: [Uso del Comando Sed](#)

## 6.7 AWK

Powerful text processing tool in Linux and Unix environments. It's primarily used for pattern scanning and processing. awk reads input line by line and divides each line into fields (by default, using whitespace as the delimiter). You can then specify patterns and actions to perform on those patterns.

**Basic:** Perform an action in the lines that match the pattern in the file

```
$ awk '/pattern/ { action }' <filename>
```

Print the first field of each line in the file (separated by default by space)

```
$ awk '{ print $1 }' <filename>
```

Print the 3rd field of each line in the file (separated by ":")

```
$ awk -F ":" '{ print $3 }' <filename>
```

Print specific fields of each line in the file (separated by ":")

```
$ awk -F ":" '{ print $2, $3 }' <filename>
```

## 6.8 Tar Command

The tar command on Linux is often used to create .tar.gz or .tgz archive files, or extract them into local directories.

### 6.8.1 Compress

#### Basics

Use the following command to compress an entire directory or a single file on Linux. It'll also compress every other directory inside a directory you specify—in other words, it works recursively.

```
$ tar -czvf name-of-archive.tar.gz /path/to/directory-or-file
```

Compress multiple files/directories in a unique archive

```
$ tar -czf FILENAME.tar.gz path/to/dir1 path/to/file1 path/to/dir2 ...
```

### Exclude files or directories

```
$ tar -czf FILENAME.tar.gz path/to/dir1 \  
--exclude=/path/to/dir1/excludeddir \  
path/to/dir2 --exclude=/path/to/dir1/*.mp4
```

## 6.8.2 Extract

### Basics

Once you have an archive, you can extract it with the tar command. The following command will extract the contents of archive.tar.gz to the current directory.

```
$ tar -xzvf archive.tar.gz
```

### Choose the extract directory

You may want to extract the contents of the archive to a specific directory. You can do so by appending the -C switch to the end of the command:

```
$ tar -xzvf archive.tar.gz -C /path/to/directory/to/extract
```

More information in: [How-To Geek tar](#)

Here's what those switches actually mean:

- **-c:** create an archive
- **-z:** compress into gzip
- **-v:** verbose [removable]
- **-f:** allows you to specify the FILENAME filename of archive

### Nota

-f switch must be the last one, because is the flag which specifies the name of the file indicated after.

## 6.8.3 More flags

- **-x:** Extract the archive
- **-t:** displays or lists files in archived file
- **-u:** archives and adds to an existing archive file
- **-A:** concatenates the archive files
- **-j:** filter archive tar file using tbzip
- **-W:** verify a archive file
- **-r:** update or add file or directory in already existed .tar file

## 6.9 WGET

**wget** is a command-line utility used to download files from the web. It stands for "World Wide Web" and "get," indicating its core functionality: retrieving content from the internet. It is highly popular due to its simplicity, robustness, and versatility. One of the main advantages of **wget** is its ability to handle unstable or interrupted network connections by automatically retrying and resuming downloads. It's available on Unix-based systems (like Linux and macOS), as well as Windows.

Basic Usage:

```
$ wget https://example.com/file.zip
```

Resuming an Interrupted Download If a download is interrupted:

```
$ wget -c https://example.com/file.zip
```

Download Files in the Background:

```
$ wget -b https://example.com/file.zip
```

Download multiple files at once by specifying URLs in a text file :

```
$ wget -i file_with_urls.txt
```

Download Entire Website:

```
$ wget -r https://example.com
```

Disable Verbose Mode:

```
$ wget -q --no-verbose https://example.com
```

Verbose Mode:

```
$ wget --debug https://example.com
```

Print the output into a file:

```
$ wget -o file_name.txt https://example.com
```

**Download and untar a file in the same command.** **-O-**: Outputs the downloaded content directly to stdout (the hyphen - represents stdout).:

```
$ wget -qO- https://example.com/file.tar.gz | tar -xz
```

## 6.10 wc

**wc** (short for "word count") is a command-line utility in Unix/Linux systems used to count lines, words, and characters (or bytes) in files or input streams. It's often used for quick text statistics and file size information.

Basic usage, will count: Lines, Words, and Characters:

```
$ wc <filename>
```

Count lines:

```
$ wc -l <filename>
```

Count words:

```
$ wc -w <filename>
```

Count characters:

```
$ wc -m <filename>
```

Count bytes:

```
$ wc -c <filename>
```

Find the length of the longest line:

```
$ wc -L <filename>
```

You can pipe the output of other commands into wc. For example, to count the number of lines in a command's output:

```
$ ls -la | wc -l
```

## 6.11 Base64 Encoding

Base64 encode or decode FILE, or standard input, to standard output.

### Code

```
$ echo -n $PASSWORD_CLEAR | base64
```

### Decode

```
$ echo -n $PASSWORD_CODED | base64 -d
```

```
titocampis@DESKTOP-FMGU: X + v
titocampis@DESKTOP-FMGU3N7:~$ echo "Hola" | base64
SG9sYQo=
titocampis@DESKTOP-FMGU3N7:~$ echo "SG9sYQo=" | base64 -d
Hola
titocampis@DESKTOP-FMGU3N7:~$ |
```

## 6.12 JQ

To parse into a beautiful json format:

```
$ cat file.json | jq "."
```

To show the content inside field in a beautiful json format:

```
$ cat file.json | jq ".field"
```

## 6.13 YQ

Is the same as **jq** but to parse YAML files.

For more information, check <https://github.com/mikefarah/yq>

## 7 Linux Default Text Editors

### 7.1 Vi

Vi (ViSual editor) is the default editor that comes with the UNIX operating system. It is a full screen editor and has two modes of operation:

- **Command mode:** mode to write shortcuts which cause action to be taken on the file, as remove line, save, undo, redo, etc.
- **Insert mode:** mode which entered text is directly inserted into the file.

We are not going to dedicate more time to **vi**, because it is actually not used. Instead of **vi**, **vim** is used (Vi IMproved). Which we are going to analyze in the next section.

### 7.2 Vim

#### 7.2.1 Introduction

Vim (Vi IMproved) is an enhanced version of the original Vi editor, it builds upon the features of Vi while adding numerous enhancements and improvements. These enhancements include syntax highlighting, built-in support for scripting, extensive customizability, plugin support, portability and many additional commands and options.

While Vi is still used and appreciated for its simplicity and efficiency, Vim has become the de facto standard for many developers and system administrators due to its enhanced features and flexibility.

As well as vi, it has two modes of operation:

- **Command mode:** mode to write shortcuts which cause action to be taken on the file, as remove line, save, undo, redo, etc.
- **Insert mode:** mode which entered text is directly inserted into the file.

When you enter vim, you enter in the **command mode**, so to change to the **insert mode**:

```
i
```

To go from **insert mode** to the **command mode**:

```
Esc
```

#### 7.2.2 Command Mode Shortcuts

Exit:

- Without save the changes

```
:q!
```

- Saving the changes

```
:q
```

- Forcing to save the changes

```
:wq
```

Undo

```
u
```

Redo

```
r
```

```
Ctrl+r # On INSERT mode
```

Remove a full line

```
dd
```

To remove the current word

- With the spaces:

```
daw
```

- Without the spaces:

```
diw
```

Find occurrences

```
/<words>
```

- To start finding the occurrences: Intro
- To go to the next occurrence: n
- To go to the previous occurrence: N
- **By default it is case sensitive, to make it insensitive:** before search, execute.

```
:set ignorecase
```

#### NOTE

If you want to make it again case sensitive:

```
:set noignorecase
```

- To replace the first occurrence of a word

```
:s/search/replace/
```



- To replace all the occurrences of a word in the full file

```
:%s/search/replace/g
```

item To replace all the occurrences of a word in the full file (with confirmation before each replacement)

```
:%s/search/replace/gc
```

Go to the start of the current line

```
0
```

Go to the end of the current line

```
$
```

Go to the beginning of the current line

```
gg
```

Go to the end of the current line

```
G
```

To configure the colors of the terminal

```
:set termguicolors  
:set background=dark  
:set colorscheme <name_of_scheme>
```

### 7.2.3 Vim Config

```
$ sudo vim ~/.vimrc
```

```

" Disable compatibility with vi which can cause unexpected issues.
set nocompatible

" Enable type file detection. Vim will be able to try to detect the type of file in use.
filetype on

" Enable plugins and load plugin for the detected file type.
filetype plugin on

" Load an indent file for the detected file type.
filetype indent on

" Set authomatic indentation
" set autoindent

" Turn syntax highlightin
syntax on

" Add numbers to each line on the left-hand side.
set number

" Highlight cursor line underneath the cursor horizontally.
set cursorline

" Highlight cursor line underneath the cursor vertically.
set cursorcolumn

" Set shift width to 4 spaces.
set shiftwidth=4

" Set tab width to 4 columns.
set tabstop=4

" Use space characters instead of tabs.
set expandtab

" Colorful (),[],{}
set showmatch

" While searching though a file incrementally highlight matching characters as you type.
set incsearch

" Ignore capital letters during search unless you use it to search.
set ignorecase
set smartcase

" Use highlighting when doing a search.
set hlsearch

" Enable auto completion menu after pressing TAB.
set wildmenu

" Make wildmenu behave like similar to Bash completion.
set wildmode=list:longest

" Enable black theme
" set background=dark

" Tab helper
" set smarttab

```

If you want to persist this configuration when using `sudo vim` you will need to do something a little bit tricky:

- Create an alias in your `~/.bashrc` file:

```
alias svim="sudo -E vim"
```

- Use always

```
$ svim <file>
```

## 7.3 Nano

Nano is a simple and easy-to-use text editor for Unix-like operating systems. It's especially popular among beginners due to its straightforward interface and command set. Unlike more complex editors like Vim or Emacs, Nano provides an intuitive environment where users can create and edit files without a steep learning curve. It features on-screen command reminders, which make it accessible for those who are new to command-line text editing.

Save the file

```
Ctrl + O
```

Exit Nano

```
Ctrl + X
```

Undo the last operation

```
Alt + U
```

Redo the last operation

```
Alt + E
```

Search within the file

```
Ctrl + W
```

Search and replace within the file

```
Ctrl + \
```

Cut the current line

```
Ctrl + K
```

Paste the current line

```
Ctrl + U
```

Move to the beginning of the current line

`Ctrl + A`

Move to the end of the current line

`Ctrl + E`

Move to the beginning of the file

`Alt + ^`

Move to the end of the file

`Alt + $`

Delete the current word

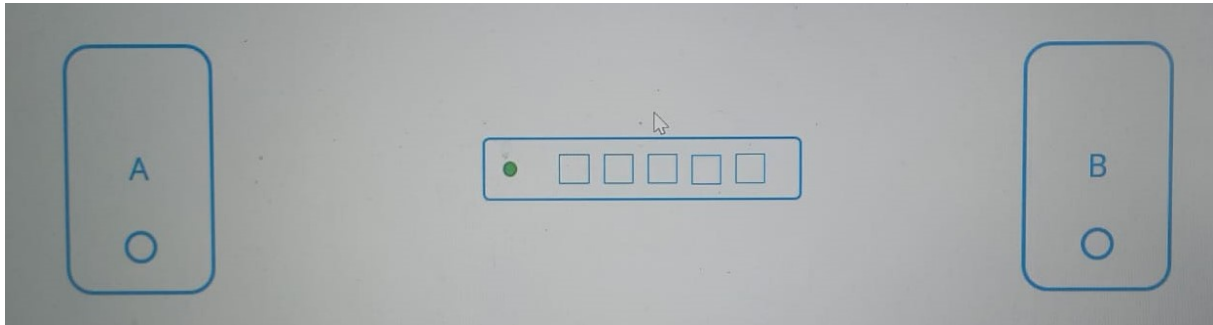
`Alt + D`

## 8 Network

### 8.1 Introduction, Switches and Routers

#### 8.1.1 Switching

What is a network? We have 2 Linux Machines, A and B, it can be: laptops, desktops, VMs on the Cloud, wherever. How does system A reach system B? So we connect them to a **switch** and it creates a network containing the two systems.



To connect the 2 Linux machines to a switch, we need an interface on each host (physical or virtual depending on the Host). And to see the interfaces from the Host, we use the following command:

- View and managing link-layer (MAC-level) settings.

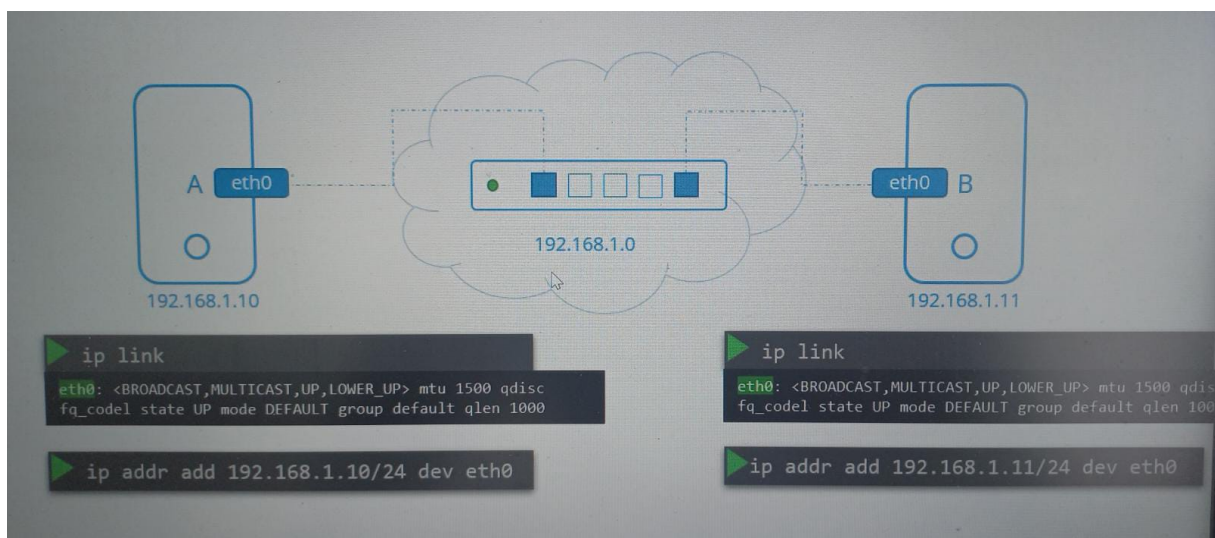
```
$ ip link
```

- Check network-layer (IP-level) details along with link-layer information.

```
$ ip a
```

We only want the information of **eth0** interface, which it will be used to connect the machine to the switch.

Let's assume it's a Network with the address 192.168.1.0, so we then assign the system with IP addresses on the same Network:



To add an IP address (192.168.1.10/24) to the network interface eth0

```
$ ip addr add 192.168.1.10/24 dev eth0
```

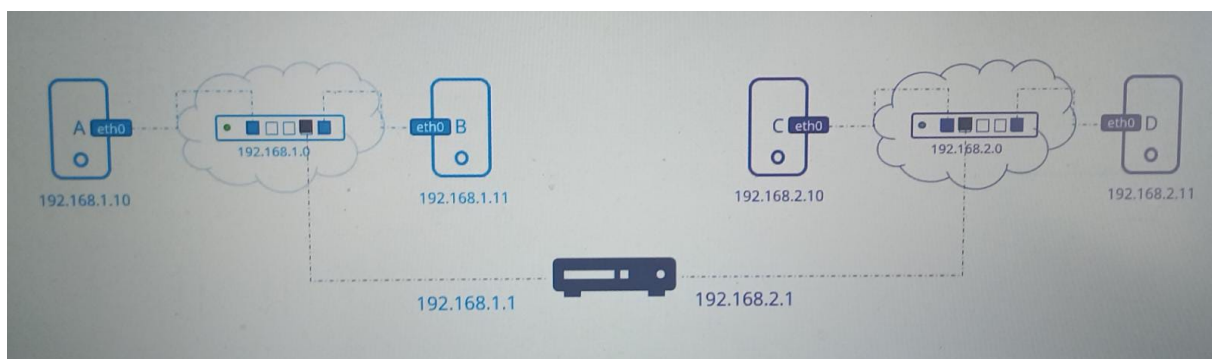
Once the links are up and the IP addresses are assigned, the computers can now communicate with each other through the switch. The switch can only enable communication within a Network, which means it can receive packets from a Host on the Network and deliver it to other systems within the same Network.

### 8.1.2 Routing

Imagine we have another 2 machines in another Network at address 192.168.2.0 and system address according to this address. How does a system in one Network reach a system in another Network?

Using **Routers**, a **Router** is an intelligent device which can connect two networks together. Is like another server with a lot of **Network Ports**. Since it connects to the two separate Networks it gets 2 IPs assigned, one on each Network, for example:

- For the first Network, it receives the IP: 192.168.1.1
- For the second Network, it receives the IP: 192.168.2.1



Imagine that system A wants to communicate with system C. How does the system A know where the Router is configured on the Network to send the packages through? The router is just another device set on the Network with an associated IP. This is why we configure the systems with a **gateway (route)**. If the Network was a room, the **gateway** would be the door to the external world (other Networks). The systems need to know where that door is.

**See existing routing configuration on a system:** it displays the kernel's routing table

```
$ route
```

If nothing is configured, system A cannot go outside its Network, so it cannot communicate with system C or D, only with B.

**To configure a gateway in system A:**

```
$ ip route add 192.168.2.0/24 via 192.168.1.1
```

Specifying that to reach 192.168.2.0/24 Network, it should pass through 192.168.1.1 (the router).

As you can imagine, we have to do the same for system B, to reach system C and system D. And the same for systems C and D to reach systems A and B. Configuring a route on the Network and configure routes on the systems.

Now suppose these systems need access to the internet, for example to google at 172.217.194.0. You only need to connect the router to the internet and add a new route to the systems, to pass through router to go to Google (172.217.194.0). But there are so many different sites on different Networks on the internet, so instead of adding a routing table entry for the same router's IP address, for each of those Networks, **we can simply say for any Network that you do not know where to route, use this router as the default gateway**. And that's the key.

```
$ ip route add default via 192.168.2.1
```

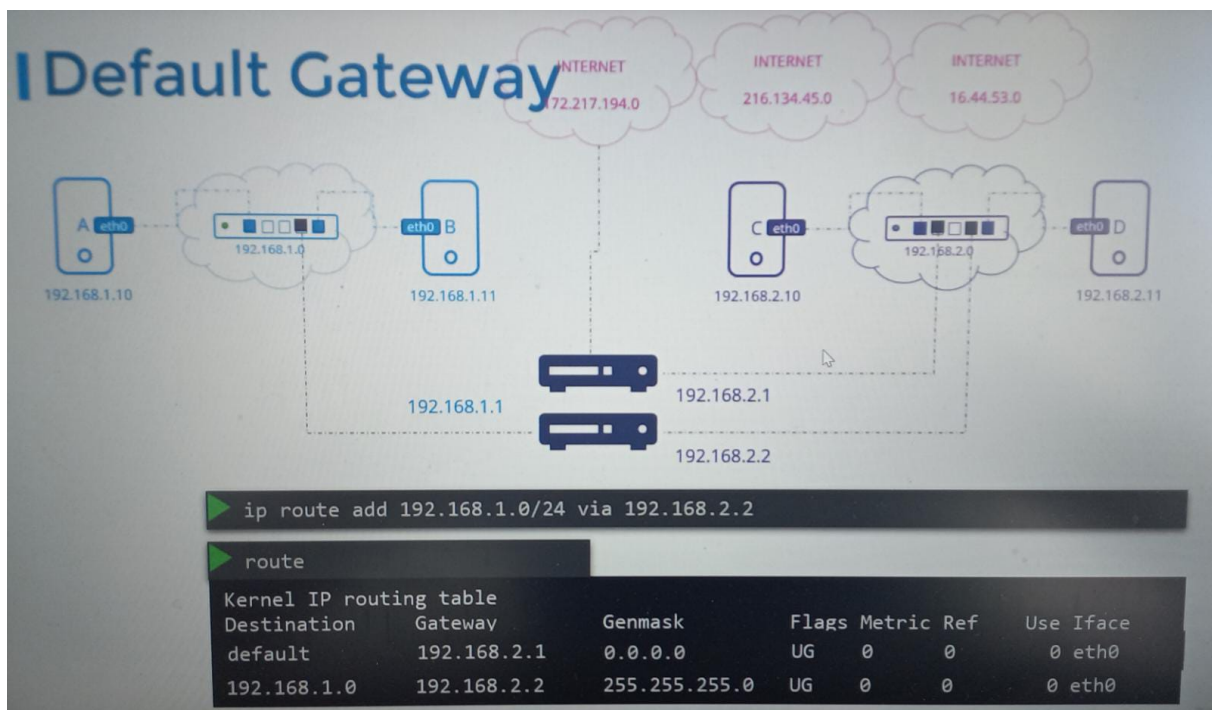
## NOTE

Instead of the word "default" we can use 0.0.0.0, which means any IP destination.

This way, any request to an IP outside of your existing Network (not defined in route table) goes to this particular router.

### 8.1.3 Private and Public Networks configuration

So in a simple model like this, all you need is a single routing table entry with the default gateway set to the router's IP address. But if we want to manage connections with another private Network at the same time than Public Network (as internet), it gets complicated. We should have 2 routers with 2 route rules in the configured system tables, to use different routers to send the traffic depending on the Network systems want to reach.



- One rule to connect to internal private Network on router 1

```
$ ip route add 192.168.1.0/24 via 192.168.2.2
```

- Another rule to connect to all other Networks (including internet)

```
$ ip route add default via 192.168.2.1
```

## 8.2 Network Namespaces in Linux

### 8.2.1 Introduction to Namespaces in Linux

Network Namespaces in Linux are used by containers like Docker to implement Network isolation, as we know **containers are separated from their underlying host using Namespaces**, but what are Namespaces? You can imagine Host is a house, and the Namespaces are the different rooms you assign to each of your child, to have certain privacy and independence, so the kids can only see their own room, they cannot see what happens in other room or in the rest of the house (they are isolated). However, as a parent, you have visibility into all the rooms in the house as well as other areas of the house. if you wish, you can establish connectivity between two rooms in the house.

When you create a container you want to make sure that it is isolated, that it does not see any other processes on the host or any other containers, so we create a special room for it on our host using a Namespace, so the container only sees the processes run by it and thinks that it is on its own host XD. Nevertheless, the underlying host has visibility into all of the processes including those running inside containers. This can be seen listing system processes.

- If you list the system processes on a **running container**, you will see only the processes executed by the container and the container process itself

*Inside Docker Container*

```
$ ps waux
```

- If you list the **system processes** on the underlying host you will see all the other processes running in the host (a lot)

*On the host directly*

```
$ ps waux
```

### 8.2.2 ip addr command

The ip addr command is a part of the ip suite of tools used in Linux-based systems to manage and display network configurations. It is used to display and modify the IP addresses, interface information, and associated network configuration parameters of a system.

ip addr (or ip address) shows information about the network interfaces and the IP addresses assigned to them. It is often used for network diagnostics and configuration purposes.

Basic usage:

```
$ ip a
```

Show details for a specific interface:

```
$ ip a show dev <interface_name>
```

Show list and human-readable network information:

```
$ ip -br a list
```

```
$ ip -br a list <interface_name>
```



Bring an interface up:

```
$ sudo ip link set <interface_name> up
```

Bring an interface down:

```
$ sudo ip link set <interface_name> down
```

Display only IPv4 information:

```
$ ip -4 addr
```

Display only IPv6 information:

```
$ ip -6 addr
```

Add an IP address to an interface:

```
$ sudo ip addr add <IP_address>/<subnet_mask> dev <interface_name>
```

Remove an IP address from an interface:

```
$ sudo ip addr del <IP_address>/<subnet_mask> dev <interface_name>
```

### 8.2.3 Namespaces for Networking

When it comes to networking, our host has its own interfaces that connect to the LAN (Local Area Network), as well as the routing and ARP tables with information about rest of the Network. We want to seal all of those details from the container, so when the container is created, we create a Network Namespace for it, that way it has no visibility to any network-related information on the host. Within its Namespace, the container can have its own virtual interfaces, routing and ARP tables.

If you want to create a new Network Namespace on a Linux host:

```
$ ip netns add <net_namespace_name>
```

To list the Network Namespaces:

```
$ ip netns
```

If you want to execute a command inside a Network Interface:

```
$ ip netns exec <net_namespace_name> <command>
```

For example to check the isolation of the network namespace:

- To see the ARP table inside the network namespace:

```
$ ip netns exec <net_namespace_name> arp
```

- To check the network interfaces inside the network namespace:

```
$ ip netns exec <net_namespace_name> ip link
```

- To check the routing network inside the network namespace:

```
$ ip netns exec <net_namespace_name> route
```

To see the ARP table:

```
$ arp
```

## NOTE

The ARP (Address Resolution Protocol) table is a crucial component of network communication, specifically within local networks. It functions as a mapping between IP addresses (logical addresses used in Layer 3 of the OSI model) and MAC addresses (physical hardware addresses used in Layer 2).

When devices on a network communicate, data is transmitted using MAC addresses at the hardware level. However, applications and services use IP addresses. The ARP table helps bridge this gap by maintaining a lookup table that translates an IP address into the corresponding MAC address of a device within the same local network.

So as they are networking isolated, they have no network connectivity, no networks on their own. The first thing we should do is establish connectivity between namespaces using a **virtual internet pair (virtual ethernet veth)**: So create the veth:

```
$ ip link add veth-<net_namespace_name1> type veth peer
    name veth-<net_namespace_name2>
```

Associate it to the network namespaces:

```
$ ip link set veth-<net_namespace_name1> netns <net_namespace_name1>
```

```
$ ip link set veth-<net_namespace_name2> netns <net_namespace_name2>
```

Assign IP address to each namespace

```
$ ip -n <net_namespace_name1> addr add <IP> dev veth-<net_namespace_name1>
```

```
$ ip -n <net_namespace_name2> addr add <IP> dev veth-<net_namespace_name2>
```

Bring up the interfaces:

```
$ ip -n <net_namespace_name1> set veth-<net_namespace_name1> up
```

```
$ ip -n <net_namespace_name2> set veth-<net_namespace_name2> up
```

The namespace will be able to see each other.

But what you would do when we have more than 2 namespaces and we have a lot of namespaces? We will need to create a **Virtual Network** inside the host. To create a Virtual Network:

- At first you need to create a **Virtual Switch**, which for our host will be just another interface.

```
$ ip link add v-net-0 type bridge
```

```
$ ip link set v-net-0 up
```

- Then connect the namespaces to this Virtual Network Switch.

*Create the veth*

```
$ ip link add veth-<net_namespace_name> type veth peer  
name veth-<net_namespace_name>-bridge
```

*Attach the veth to namespace*

```
$ ip link set veth-<net_namespace_name> netns veth-<net_namespace_name>
```

*Attach the veth to the bridge*

```
$ ip link set veth-<net_namespace_name>-bridge master v-net-0
```

- Assign IP addresses and turn them up:

```
$ ip -n <net_namespace_name> add 192.168.15.1 dev veth-<net_namespace_name>
```

```
$ ip -n <net_namespace_name> link set veth-<net_namespace_name> up
```

If I want to establish connectivity between my host and this network interfaces:

```
$ ip addr add 192.168.15.5/24 dev v-net-0
```

Anyway from within the namespaces you cannot reach the outside world, nor can anyone from the outside world reach the services. If we want to access the internet we will need to open the door, so we need to add a gateway.

```
$ ip netns exec <net_namespace_name> ip route add 192.168.1.0/24 via 192.168.15.5
```

```
$ ip netns exec <net_namespace_name> ip route add default via 192.168.15.5
```

As well we need to add a new rule in the NAT IP table for the outside world to reply:

```
$ iptables -t nat -A POSTROUTING -s 192.168.15.0/24 -j MASQUERADE
```

But the response always will go to the localhost, because for the entire world this is the only IP reachable. So if you want that the outside world access the application you will need to define Port-Forward rules (NAT).

## 8.3 OSI Model

The Open Systems Interconnection (OSI) model is a reference model from the International Organization for Standardization (ISO) that "provides a common basis for the coordination of standards development for the purpose of systems interconnection. In the OSI reference model, the communications between systems are split into seven different abstraction layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application.

7	Application Layer	Human-computer interaction layer, where applications can access the network services
6	Presentation Layer	Ensures that data is in a usable format and is where data encryption occurs
5	Session Layer	Maintains connections and is responsible for controlling ports and sessions
4	Transport Layer	Transmits data using transmission protocols including TCP and UDP
3	Network Layer	Decides which physical path the data will take
2	Data Link Layer	Defines the format of data on the network
1	Physical Layer	Transmits raw bit stream over the physical medium

## 8.4 MAC Address vs IP Address

### 8.4.1 Introduction

Suppose someone has to deliver a package to someone else. To effectively send the courier, the sender must provide two details about the recipient. The two items are the receiver's name and address, which may include a house number, street, city, state, and pin code (to identify the right person to deliver the courier specifically). If we apply this example to networking, the MAC address will be the address of the particular node where we want to send the data, and the IP address will be the address of the network connection where several devices may be present.

A MAC address and an IP address both help identify a device specifically on the internet. The IP address is generated by the ISP (Internet Service Provider), whereas the MAC address is provided by the Network Interface Controller (NIC) manufacturer. There are significant differences between a MAC address and an IP address. The IP address of a device is crucial for determining a network's connection (using which the device is connecting to the network). On the other hand, the MAC address guarantees the computer device's precise location. It enables us to uniquely identify a certain device on the access network.

Every device connected to a network has a Media Access Control (MAC) address that uniquely identifies it, much like every house has a unique postal address. Meanwhile IP addresses are just as significant as a person's identification number or card.

### 8.4.2 MAC Address (Media Control Address)

A MAC address, which is also known as a hardware or physical address, is a unique alphanumeric identifier consisting of 12 characters. MAC address is allocated to a network interface controller (NIC) and functions as a network address when communications occur within a network segment. Ethernet, Wi-Fi, and Bluetooth are just a few of the IEEE 802 networking technologies that frequently employ this application. Six groups of two hexadecimal digits between 00 and FF (48 bits in length) constitute MAC addresses. They can be separated by ":", "-", " ", " " or not separated at all. An example of MAC address:

- 02:AB:6D:9C:EF:42
- 02-AB-6D-9C-EF-42
- 02 AB 6D 9C EF 42

The first 24 bits of the MAC address, the first 3 octets (first 3 hexadecimal groups) are the ID number of the adapter manufacturer, here we have examples of OUIs (Organisationally Unique Identifier):

- **Cisco:** CC:46:D
- **Google:** 3C:5A:B4
- **HP:** 3C:D9:2B
- **Intel:** 00:A0:C9

The serial number that the manufacturer assigned to the adapter is represented by the second half of a MAC address (24 more bits)



MAC address operate at the Data Link Layer (Layer 2) of the OSI model. They are used to identify devices within a local network (LAN). It is unique: every network interface card (NIC) is assigned a unique MAC address by the manufacturer, ensuring that no two devices on the same network have the same MAC address.

Within a local network, devices communicate with each other using MAC addresses. For instance, when a device wants to send data to another device on the same network, it uses the destination device's MAC address. As it is operating at OSI Layer 2, data is encapsulated into Ethernet frames, which include the source and destination MAC addresses.

#### How to know my MAC address?

Windows (powershell): execute the following command and look for the "Physical Address" under the network adapter you're using.

```
$ ipconfig /all
```

Linux: execute the following command and check for ethX for Ethernet or wlanX for Wi-Fi

```
$ ip addr
```

Router, Mobile, Windows (without terminal), Linux (without terminal): check the device configuration.

### 8.4.3 IP Address

The term "Internet Protocol", for which the abbreviation IP stands, is a set of guidelines that control the format of data exchanged via a local or public network.

An IP address is an identification tool for network connections. A link in a network receives what is known as a "logical address" from the network. IP addresses define how internet routers behave and allow you to regulate how devices communicate over the Internet.

An internet protocol address (IP address) is a numerical representation of a network interface that serves as its sole means of identification such as 192.97.23.57.

The two IP versions that are now in use are IPv4 and IPv6. The original IPv4 protocol is still in use today on the internet and in many corporate networks. On the other hand, the IPv4 protocol was limited to  $2^{32}$  addresses. Due to the way addresses were assigned, there would be a situation where there wouldn't be enough unique addresses for all internet-connected devices. That's why the most recent version is Internet Protocol version 6 (IPv6). To accommodate the demand for more Internet addresses, this new IP address version is being used. It was developed to overcome IPv4 issues. With a 128-bit address space, it supports 340 billion distinct address spaces. IPng (Internet Protocol next generation) is another name for IPv6.

An IP address is used to manage the connection between gadgets that send and receive data over a network. Without an IP address, it is impossible to communicate with any device connected to the internet. IP addresses allow computing devices (such PCs and tablets) to communicate with websites and streaming services, as well as information websites of the identity of the connecting user.

IP Address works operate at the Network Layer (Layer 3) of the OSI model. They are used for logical addressing and routing of packets across different networks, including the global internet, so data is encapsulated into IP packets, which include the source and destination IP addresses.

### 8.4.4 Why Both MAC and IP Addresses Are Needed

Having both MAC and IP addresses allows for the separation of physical hardware identification (MAC) from logical network addressing (IP). This separation simplifies network management, mobility, and security.

Devices can move between networks and retain their unique MAC address, while their IP address can change based on the network they are connected to. This flexibility supports mobile devices and scalable network architectures.

#### Local vs Global Scope:

- **MAC Address:** Used for communication within a local network segment (Layer 2). It is essential for Ethernet communication and ensures that data reaches the correct device on the same network.
- **IP Address:** Used for communication between different networks (Layer 3). It enables data to be routed across the internet and large networks.

#### Routing and Address Resolution:

- **Routing:** IP addresses are used by routers to make forwarding decisions and route data across multiple networks to its final destination.
- **Address Resolution Protocol (ARP):** When a device wants to communicate with another device on the same local network, it uses ARP to map the destination IP address to the corresponding MAC address. This mapping is necessary because data frames on Ethernet networks require MAC addresses.

### 8.4.5 ARP Table

The **ARP (Address Resolution Protocol) Table** is a data structure used by a network device to store mappings between IP addresses and MAC (Media Access Control) addresses. In local area networks, when a device wants to communicate with another device, it uses the ARP protocol to resolve the MAC address associated with the target device's IP address.

The ARP table stores these IP-to-MAC mappings temporarily, allowing the device to communicate efficiently without constantly querying for the MAC address. Each entry in the table contains:

- The IP address.
- The corresponding MAC address.
- A timer to track how long the mapping has been valid.

Displays the current ARP table entries. You can see a list of IP addresses and their corresponding MAC addresses:

```
$ arp -a
```

A modern command to display the ARP cache on Linux systems:

```
$ ip neigh
```

Deletes an entry from the ARP table manually:

```
$ arp -d <IP\_address>
```

Clears the entire ARP cache:

```
$ ip neigh flush all
```

The output of `arp -a` typically includes the following columns:

- **IP Address:** The IP address of the device.
- **MAC Address:** The MAC address corresponding to the IP address.
- **Type:** Indicates whether the ARP entry is *dynamic* (learned via ARP requests) or *static* (manually entered).

This table is essential for ensuring proper communication between devices in the same local network.

## 8.5 Public IP

### 8.5.1 Introduction

A public IP address uniquely identifies your network on the internet. It allows other networks and devices on the internet to locate and communicate with your network. When you send data over the internet (e.g., browsing websites, streaming videos), it needs a destination and return path. The public IP address serves as the return address for the data requested by devices within your local network.

The Public IP is set at router level, not at device level, and any time you access a website, your request is sent from your device to your router, which then uses the public IP address to send the request out to

the internet. The response from the website is sent back to your router's public IP address, which then routes the response back to your device and vice versa.

The router uses NAT to translate private IP addresses (used within your local network) into the public IP address. This allows multiple devices on your local network to share a single public IP address for accessing the internet.

So if you host a service (e.g., a web server) on a device on your network, you will need to define a NAT route in your router in order to other devices on the internet use your public IP to connect to your server.

### 8.5.2 How to get my public ip?

I can do it with different methods:

```
$ curl ifconfig.me
```

```
$ dig +short myip.opendns.com @resolver1.opendns.com
```

Each of these commands contacts an external service that echoes back your public IP address. This IP address is what others on the internet see as your point of contact.

### 8.5.3 Who is this IP assigned to me?

Your router's public IP address is assigned by your Internet Service Provider (ISP). Usually, if your IP is not for business or point to point specific connection, in which case the assignment of the IP is static. The assignment of the IP is dynamic using DHCP.

The dynamic IP Address Assignment using DHCP works like following:

1. Your ISP uses Dynamic Host Configuration Protocol (DHCP) to assign IP addresses. This means that your router is given a dynamic public IP address from a pool of available addresses.
2. The IP address is typically leased to your router for a certain period. Once the lease expires, the ISP can assign the same IP address or a different one.

## 8.6 DHCP

### 8.6.1 Introduction

DHCP (Dynamic Host Configuration Protocol) is a network management protocol used to dynamically assign an IP address to any device, or node, on a network so it can communicate using IP. DHCP automates and centrally manages these configurations rather than requiring network administrators to manually assign IP addresses to all network devices. DHCP can be implemented on small local networks, as well as large enterprise networks.

DHCP assigns new IP addresses in each location when devices are moved from place to place, which means network administrators do not have to manually configure each device with a valid IP address or reconfigure the device with a new IP address if it moves to a new location on the network.

Versions of DHCP are available for use in IP version 4 (IPv4) and IP version 6 (IPv6).

It is a client-server protocol in which servers manage a pool of unique IP addresses, as well as information about client configuration parameters. It dynamically assigns IP addresses to DHCP clients and allocates TCP/IP configuration information to DHCP clients. This information includes subnet mask information, default gateway IP addresses and domain name system (DNS) addresses. DHCP-enabled clients send a request to the DHCP server whenever they connect to a network.



### 8.6.2 How IP's are assigned?

1. **DHCP Request:** when a device (e.g., a computer or smartphone) connects to your network, it sends out a DHCP request asking for an IP address.
2. **DHCP Offer:** the DHCP server responds with a DHCP offer, which includes an available IP address, subnet mask, gateway address, and DNS server information.
3. **DHCP Acknowledgment:** the device acknowledges the offer, and the DHCP server finalizes the lease, assigning the IP address to the device for a specified period.

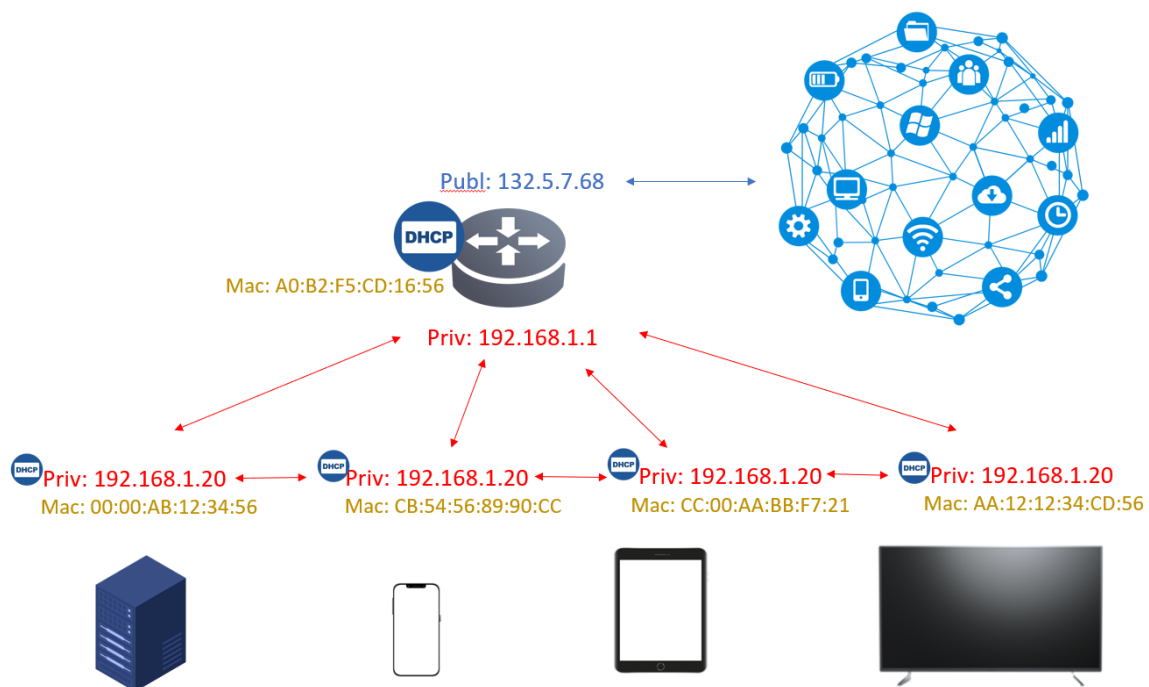
## 8.7 Private IP

### 8.7.1 Introduction

We have seen that the router is your device gateway to the internet, but how your router knows to which device redirect the requests? This is because all your devices: phone, tablet, smart tv, laptop... Have private IP addresses inside your local network, assigned directly by your router.

Your router is the responsible for assigning private IP addresses to devices on your local network. It does this using the DHCP service, which is typically built into your router making it the DHCP server on your private network. When a device connects to the network, it requests an IP address from the DHCP server (the router), which then assigns an available address from a predefined range.

The private network (subnet) created by your router is typically not random. Most routers come with a default subnet configuration, often something like 192.168.0.0/24 or 192.168.1.0/24. These default subnets are predefined by the manufacturer to ensure ease of setup and compatibility with most home networking needs.



### 8.7.2 Advantages of have Private IP addresses

- **Address Space Conservation:** private IP addresses allow multiple devices to share a single public IP address (the one in the router), conserving the limited number of available public IP addresses.
- **Network Segmentation:** private IP addresses enable you to create a local network that can communicate internally without directly exposing each device to the internet.
- **Security:** : by using private IP addresses, devices within your local network are protected from direct access by external users on the internet.

### 8.7.3 Can the private subnet be changed?

Yes, you can change the subnet configuration of your router. This is useful if you have specific networking requirements or to avoid conflicts with other networks (e.g., when setting up a VPN or connecting to

another network with the same default subnet).

### How to Change It

1. **Access Router Settings:** open a web browser and enter the router's IP address (commonly 192.168.1.1 or 192.168.0.1) to access the router's administration interface.
2. **Login:** enter the admin username and password (default credentials are often found on the router or in the manual).
3. **Navigate to Network Settings:** look for sections like "LAN Setup," "Network Configuration," or similar.
4. **Change Subnet:** you can change the IP address range, such as changing from 192.168.1.0/24 to 192.168.2.0/24 or any other valid private IP range.
5. **Save and Reboot:** save the changes and reboot the router for the new settings to take effect.

#### NOTE

You can change a lot of things in the router configuration following these instructions, the IP's assigned to some devices, static IP assignment instead of using DHCP, NAT rules, etc.

### 8.7.4 Private IP Address Ranges

Private IP addresses are defined by the Internet Engineering Task Force (IETF) in RFC 1918. They are reserved for use within private networks and are not routable on the public internet. The commonly used private IP address ranges are:

- 10.0.0.0 to 10.255.255.255
- 172.16.0.0 to 172.31.255.255
- 192.168.0.0 to 192.168.255.255

### 8.7.5 Example of Private IP Assignment

1. **Router Configuration:** Your router might be configured to use the range 192.168.1.2 to 192.168.1.254 for DHCP.
2. **Device Connection:** when your smartphone, laptop, tablet connects to the Wi-Fi, it sends a DHCP request asking for an IP address.
3. Then the router's DHCP server assigns it 192.168.1.10 (for example) along with other network information.

## 8.8 DNS in Linux

### 8.8.1 Introduction to DNS

We have 2 Hosts in the same Network, Host A and Host B. And they have been assigned a Network with IP: 192.168.1.0. So they are 192.168.1.10 and 192.168.1.11. So as we saw in the last section ???. As they are in the same Network, they can reach each other using their IP's. But it can be so tediously for Hosts to know all the IP's, so instead of having to remember the IP address of system B, Host A can give it a name, for example db, that when you `ping db` it works like if it was using IP.

Basically we want to tell Host A that system B at IP address at 192.165.1.11 has a name db. So when db name is used, for Host A must be the same as 192.165.1.11. It can be done by adding an entry in the file `/etc/hosts`:

```
$ echo "192.168.1.11      db" >> /etc/hosts
```

Or vim and add at the end of the file the relationship.

```
$ sudo vim /etc/hosts
```

So now Host A translates db as 192.165.1.11, this is called **name resolution**, associate a hostname to an IP. Whatever we put in the `/etc/hosts` is the source of true for the Host. But that may not be the truth, maybe Host B is called host-2, but Host A does not care, it will follow what is in the `/etc/hosts` file.

You can make a test, associate `www.google.com` hostname to IP of Host B and test to ping, the ping will be answered by Host B. You can have any names as you want for any servers as you want in the `/etc/hosts` file. Commands look into this file:

- ping
- ssh
- curl
- nc
- nmap
- mtr
- ...

### 8.8.2 DNS server

In a system with few systems, you can get away with the entries of `/etc/hosts` file, and this is how it was done in the past. However, when the environment grows and these files got filled with many entries you can imagine this is unmanageable. If one of the Hosts changes its IP, you are done, goodbye. This is the reason why it was decided to **move all this configuration into a single server**, who will manage all this centrally, called the **DNS server**. And then we point all hosts to look up that server if they need to resolve a hostname to an IP address instead of its own `/etc/hosts` files.

How do we do that? How do we point our host to a DNS server? Imagine our DNS server has the IP 192.168.1.100, every host has a DNS resolution configuration file at `/etc/resolv.conf`. So you have to add an entry into it specifying the address of the DNS server.

```
$ echo "nameserver      192.168.1.100" >> /etc/resolv.conf
```

Or vim and add it at the end of the file.

```
$ sudo vim /etc/resolv.conf
```

Once this is configured into a Host, everytime it comes up across a hostname that it does not know about, it looks it up from the DNS server. With this configuration, if the IP of any of the host was to change, you only need to change it in one file in one Host.

## WARNING

You can still have entries in your `/etc/hosts` file, but it is not hardly recommended because it will override the configuration in `/etc/resolv.conf` nameserver. Only use it when you are sure of what you are doing.

But this is default configuration, this order can be changed in the file `/etc/nsswitch.conf`

```
$ cat /etc/nsswitch.conf
```

**OUTPUT:** hosts: files dns

To change this, you should modify the content on the field to invert it.

What happens if the hostname is not found in `/etc/hosts` neither in `/etc/resolv.conf`, so it will fail. Unless you configure in the file `/etc/hosts` of the **DNS server** to forward all unknown hostnames to the public name server on the internet (ex: 8.8.8.8).

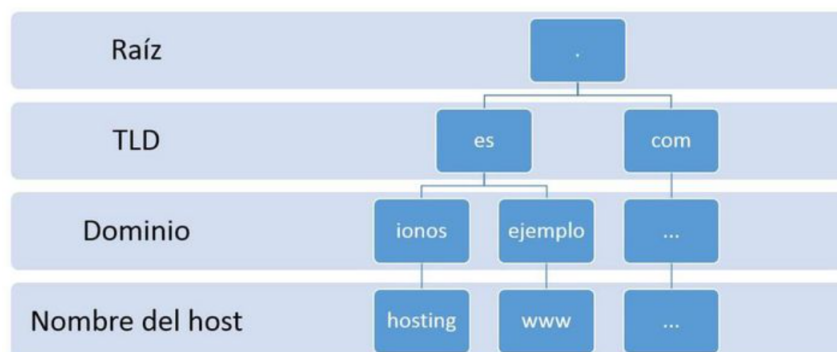
```
$ echo "Forward All to 8.8.8.8" >> /etc/hosts
```

### 8.8.3 FQDN

So far, we are naming with names like db, host-1, host-2... But the real services are called: `www.google.com`, `www.github.com`, etc. They are using **FQDN's**.

**FQDN** (Fully Qualified Domain Name) or **Domain Name**, which are a complete and unique direction needed to have presence on the internet. It is composed by the name of the host and the domain, and it is used to request to specific hosts on the internet using the name resolution instead of their IP.

Let's see a FQDN structure in detail:



1. **TLD (Top Level Domains):** they represent the intent or the geographical situation of the website:
  - **.com:** for commercial or general purpose
  - **.net:** for network
  - **.edu:** for educational organizations
  - **.es / .cat / .fr:** for geographical situation
  - **.org:** for non-profit organizations
2. **Domain:** it is the hostname assigned to Google
3. **Name of the host:** is a subdomain, which helps in further grouping things together under the Domain, for example:

- **maps.google.com:** the maps of google
- **drive.google.com:** the drive of google
- **www.google.com:** google search
- **mail.google.com:** google email

But how does this magical works in all Hosts in the world, for exampe the PC we are using. This is because when your system tries to reach any of these **FQDN's** your system first check at your **/etc/hosts** or **internal DNS server**, and it normally does not know who apps or google is, so it forwards your request to **your internet provider DNS server (you can change this on your machine configuration)**. Then the DNS Server looks at your request and forwards it to the correct host.

#### Orange DNS (France)

- Primary DNS: 8.8.8.8
- Secondary DNS: 8.8.4.4

#### Google Public DNS

- Primary DNS: 8.8.8.8
- Secondary DNS: 8.8.4.4

#### Cloudflare DNS

- Primary DNS: 1.1.1.1
- Secondary DNS: 1.0.0.1

#### OpenDNS

- Primary DNS: 208.67.222.222
- Secondary DNS: 208.67.220.220

#### Quad9 DNS

- Primary DNS: 9.9.9.9
- Secondary DNS: 149.112.112.112

#### Comodo Secure DNS

- Primary DNS: 8.26.56.26
- Secondary DNS: 8.20.247.20

But as this is so tedious, your **local DNS server (router)** may choose to cache this IP for a period of time, typically few seconds up to few minutes. That way, it does not have to go throught the whole process again each time.

### 8.8.4 Creating own FQDN

Imagine we want to expose a service to the internet with different sub-services, like a pay service, a consult service, a book service, a message service, etc. And do you want to access them on the internet by FQDN names and not IP's, so common. You must have an **internal DNS server**, and you must configure the hostname-IP relationship in its `/etc/hosts` file.

So when you want to access your service from inside the Network from any host, the host at first will look in its local configuration file `/etc/hosts`, then if nothing finds it will look in the **DNS server** configured in the `/etc/resolv.conf`, which will tell you that `pay.myapp.com` is the IP 192.68....

But we can one step further, inside my private network, maybe I want to use the service without the FQDN, because it is very long, maybe I want to use it by web, it is easier. Is this possible? The answer is yes, you must make an entry into your host's `/etc/resolv.conf`:

```
search mycompany.com
```

Next time you try to ping web, you will see it actually tries `web.mycompany.com`. As if you try pay, it will try `pay.company.com`.

### 8.8.5 Record Types (Tipos de Registros)

How are the records stored in the DNS server?

- **hostname to IP:** A records
- **hostnames to IPv6:** AAAA records
- **Mapping one name to another name:** CNAME records. For example, you may have multiple aliases for the same application (`maps.google.com` = `car.google.com`, `walk.google.com`, `cycle.google.com`, etc.) and that's why a CNAME record is used, name to name mapping.

### 8.8.6 DNS Resolution Process

#### 8.8.6.1 Checking the Local Cache

When you try to access an fqdn (like `www.google.com`), your machine first checks its local DNS cache to see if it already knows the IP address. This cache is typically managed by a service such as `systemd-resolved`, `dnsmasq`, or `nsd`.

```
acampos@BCNLT5C63284PRF ~/personal/latex/latex-linux-administrator [main] ps waux | grep -i 'resolv'
systemd+ 154 0.0 0.1 25540 12612 ? Ss 11:26 0:00 /lib/systemd/systemd-resolved
```

#### 8.8.6.2 Hosts File

If the address is not found in the cache, it then checks the `/etc/hosts` file, which can have static mappings of hostnames to IP addresses.

```
/etc/hosts
127.0.0.1    localhost
8.8.8.8      www.dns-google.com
8.8.4.4      www.dns-google-2.com
9.9.9.9      quad9-dns.net
89.163.78.34 my.webpage.com
```

#### Note

In windows this file is in C:\Windows\System32\drivers\etc\hosts

### 8.8.6.3 DNS Resolver Configuration

If the fqdn is not found in the `/etc/hosts` file, the system uses DNS servers defined in the `/etc/resolv.conf` file. This file contains the IP addresses of the DNS servers that the system will query to resolve the domain name.

A typical `/etc/resolv.conf` might look like this:

```
/etc/resolv.conf  
  
nameserver 9.9.9.9  
nameserver 8.8.8.8  
nameserver 8.8.4.4
```

So the system sends a DNS query to one of the DNS servers listed in `/etc/resolv.conf`. The query asks for the IP address associated to the fqdn.

#### Note

The IP addresses 8.8.8.8, 8.8.4.4, and 9.9.9.9 are addresses for public DNS servers provided by different organizations. 8.8.8.8 and 8.8.4.4 are public DNS servers operated by Google. They are part of the Google Public DNS service, which was launched in December 2009. The service aims to make the web faster and more secure by providing a reliable and fast DNS resolution service.

- 8.8.8.8: Primary DNS server
- 8.8.4.4: Secondary DNS server

9.9.9.9 is a public DNS server operated by Quad9, a nonprofit organization. Quad9 was launched in 2017 with the goal of improving internet security and privacy. You also have the CloudFlare one in 1.1.1.1.

### 8.8.6.4 Recursive DNS Resolution inside the DNS Server

The DNS server might not know the answer immediately and may perform a recursive resolution process:

- **Root DNS Servers:** The DNS server first contacts a root DNS server (one of the root name servers).
- **Top-Level Domain (TLD) Servers:** The root server responds with a referral to a TLD server (for .com domains, it would be a .com TLD server).
- **Authoritative DNS Servers:** The TLD server then provides a referral to the authoritative DNS server for google.com.
- **Final Resolution:** The authoritative DNS server for the fqdn responds with the IP address for the fqdn

### 8.8.7 Caching the Result

Once the IP address is retrieved, it is sent back to your machine, which can now connect to the fqdn. This result is also cached locally to speed up future requests.



### 8.8.8 DNS check Tools

#### 8.8.8.1 nslookup

nslookup (Name Server LOOKUP) is a network utility used for querying the Domain Name System (DNS) to obtain domain name or IP address mapping information. It is a command-line tool available on many operating systems, including Windows, macOS, and Linux. nslookup helps troubleshoot DNS-related issues by allowing users to query DNS servers and view the details of DNS records.

Basic usage

```
$ nslookup example.com
```

Specify a DNS Server to resolve the fqdn

```
$ nslookup example.com 9.9.9.9
```

Reverse Lookup: find the domain name associated with an IP address

```
$ nslookup 8.8.4.4
```

#### 8.8.8.2 dig

dig (Domain Information Groper) is a network administration command-line tool for querying the Domain Name System (DNS). It is commonly used to retrieve information about DNS records, troubleshoot DNS issues, and verify DNS configurations. dig is widely used on Unix-like operating systems such as Linux and macOS, and it can also be installed on Windows.

Basic usage

```
$ dig example.com
```

Specify a DNS Server to resolve the fqdn

```
$ dig @9.9.9.9 example.com
```

Reverse Lookup: find the domain name associated with an IP address

```
$ dig -x 8.8.4.4
```

#### Analysing the dig output

- **QUESTION SECTION:** The query you made.
- **ANSWER SECTION:** The resolved IP address(es) for www.google.com.
- **SERVER:** The DNS servers that are authoritative for the domain.
- **ADDITIONAL SECTION:** Any additional information provided by the DNS server.

Short answer

```
$ dig +short example.com
```

Verbose answer

```
$ dig +trace example.com
```

## 8.8.9 Local Cache Services

### 8.8.9.1 systemd-resolved

**systemd-resolved** is a system service provided by the **systemd** suite of system and service management tools for Linux operating systems. It is responsible for network name resolution, which means converting human-friendly domain names (like "example.com") into IP addresses that computers can use to communicate over a network. It is in charge of Caching and split DNS support for faster DNS resolution and optimized network usage.

The configuration files for **systemd-resolved**:

- **/etc/systemd/resolved.conf**: Main configuration file for **systemd-resolved**. It contains settings for DNS servers, domains, and other options.
- **/etc/resolv.conf**: typically, if you are using **systemd-resolved**, it is a symbolic link pointing to **/run/systemd/resolve/resolv.conf** or **/run/systemd/resolve/stub-resolv.conf**. This file is what applications traditionally read to find DNS settings when they are not cached.

Check if **systemd-resolved** is running

```
$ ps waux | grep -i 'systemd-resolved'
```

```
$ sudo systemctl status systemd-resolved
```

### 8.8.9.2 dnsmasq

**dnsmasq** is a lightweight, easy-to-configure DNS forwarder and DHCP server. It is designed to provide DNS, DHCP, TFTP, and PXE services for small networks. Here's a detailed look at **dnsmasq**:

Functions and features:

- **DNS Forwarding**: forwards DNS queries to upstream DNS servers and caches the responses locally to speed up subsequent queries.
- **DNS Caching**: Caches DNS responses to improve performance and reduce load on upstream DNS servers.
- **DNS Masquerading**: Can provide custom DNS responses for local network addresses, which is useful for internal name resolution.
- **DHCP Server**: acts as a DHCP server, dynamically assigning IP addresses to devices on a local network. It supports DHCPv4 and DHCPv6.
- **TFTP Server**: provides Trivial File Transfer Protocol (TFTP) services, which are often used for network booting.
- **PXE Boot**: Supports network booting using PXE (Preboot Execution Environment), which is useful for diskless workstations or automated OS installations.

Main configuration file from `dnsmasq`, it includes settings for DNS servers, DHCP ranges, TFTP options, and more: `/etc/dnsmasq.conf`

Check if `systemd-resolved` is running

```
$ ps waux | grep -i 'dnsmasq'
```

```
$ sudo systemctl status dnsmasq
```

## 8.9 SSH

### 8.9.1 What is SSH?

SSH, also known as Secure Shell or Secure Socket Shell, is a network protocol that gives users, particularly system administrators, a secure way to access a computer over an unsecured network. Secure Shell provides strong password authentication and public key authentication, as well as encrypted data communications between two computers connecting over an open network, such as the internet.

In addition to providing strong encryption, SSH is widely used by network administrators to manage systems and applications remotely, enabling them to log in to another computer over a network, execute commands and move files from one computer to another.

The most basic use of SSH is to connect to a remote host for a terminal session. The form of that command is the following:

- IP:

```
$ ssh user_name@host_IP
```

- Hostname:

```
$ ssh user_name@SSHserve.example.com
```

The SSH key command instructs your system that you want to open an encrypted Secure Shell Connection. `user_name` represents the account you want to access. For example, you may want to access the root user or `acampos` user.

#### Note

To connect via ssh using a user, it is needed to have this user created in the host server.

- **host server:** refers to the remote server you are trying to access.
- **client server:** refers to the server you are using to access the host.

For the first time negotiating a connection between the client server and the host server, the user will be prompted with the remote host's public key fingerprint and prompted to connect.

```
The authenticity of host 'sample.ssh.com' cannot be established.  
DSA key fingerprint is 01:23:45:67:89:ab:cd:ef:ff:fe:dc:ba:98:76:54:32:10.  
Are you sure you want to continue connecting (yes/no)?
```

Answering yes to the prompt will cause the session to continue, and the host key is stored in the client server local system's `known_hosts` file. This is a hidden file, stored by default in a hidden directory, called `/.ssh/known_hosts`, in the user's home directory. Once the host key has been stored in the `known_hosts` file, the client system can connect directly to that server again without need for any approvals; the host key authenticates the connection.

```
acampos@CNL15CG3284PRF:~/work/opticlimb$ cat ~/.ssh/known_hosts  
[1]/9rmhuubZ4qgAtyc+oET4qSiQBY=|WcLLPZ50ha/xbtLEywd3oX5e94U= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIEFFvzDJXiUu2iuu8j4cEVQb0H9cSXS4YWevJ4IaPE0I  
[1]Bzdkkx08QRMefu4wv/rvtd0gW4=|ygfyp0pkHAScl5bNv7KAHfx108U= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI0Ix2nnszRk5QD3SCVA+agB7Vx6HRRKZoLwRITweV3eUc  
[1]xQeGrSv4REETw69a7Y010z0X/NQ=|4sJm12fVfJYujzcUJkplEbU0GA8= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI6CTK0N0wT0yGH16KL U16em6TYEeLbEbl.OMxSWrLkmI  
[1]8KcWwKTTB0d4FTR+A0WXPQxtK3A=|IDLy4NnsUJ4chEQUEmv/NZ86jd0= ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDUaRNe39+gxSvGigeN5otQYTeMswkaexBYqHYDRkm1xT488c8Se9pWVD  
J/zokG9vUQIz597RAIMRIKHRgdnEUpigJR3+4627Fa1NY+V0hjS765JXkuTI3VD08drZ0Kv2KoFLh27TdPHVBw73G67eVsC3+Zhqo9VMGgfk8bpUCswYct5NI7o0VC3UPYLL6NmQ2PcdhLDAi6Esvqjay6L  
Gr92PD5JfV0+zUd2t0n5UK40Lc07Yt5h04b6ytOKNydmh2VaRYJCqQ/IdCxSg1j5B2HmZJImJFdemuiGLym6PJm1Wx9JnPKtLAnaWUQ3/2+/zid8W5wiXyC+BfKfs/G69t7AJVDJ0b7PeKw6E7KETabbwGz  
T1fLXiePpS8UkGSyqzS3urStqyzQ88QDVtXpQkwxKsM2pJ483fbVn/e5dfx/0mNuHxJtAzNnsQxPJ+Z053jz0ji8B1g27rm7cvGjVa7SruRpULt69p9gnjKjZGBXWNJf2i/50KrrbZIIjsAc=  
[1]B21LUO/qmD5TLV9HgE55a4Bwo=|CQixy4AXDHMFouFMBST5MgBBges= ecDSA-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHdhYNTYAAAAIbmlzdHdhYNTYAAABBBP/mJkG+VEIPYxRfZA/1WW  
noiMvVvAeyawseNXyhr3++kpNmHGEbyEu0KcJBDGTmQj00rMbHhFvPipjbeDppFW4=  
[1]qEPdg8lgCj14tW2Ut0BoT8HRXZ0=|wEcMAGcXcUILLH9Qszjcv7B/cmg= ecDSA-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHdhYNTYAAAAIbmlzdHdhYNTYAAABBBLJXQm24i4gw0lfMNHJjtYA  
iiyM3dsN0cqfRymYcJr2LE2HoThierLJF0unoCVvbL0qU7eukvHMD/euVjZ1q/A0=  
[1]Qn5XbJFL0m+B+BB68K3FYLfNqF0=|9Xtv+hhmS6XNuSG0spzdV3fLzz= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPcsdudljzZEFQMTcN/t1ug/ePhi+PvEnDNf6mZcStde  
[1]LxrxehRUjYopfqzNPv/4ZfBFq0=|2dgVAaRBD0LLy+AgGGR+1yFouhA= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIKGDU4gq4CXHYzBjn5R3RjPvpwph9ykv27j03wf/QQJsm  
[1]soAYHGrrkoPjFTHth/jmUMAtLpc=|2A+I7E0UtKpotZMqfMAV4IcTpII= ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIIIsLePk3hZW8YPLHkn7H8HVaXkD+Own6S6mD0V2nZF  
[1]S9ACTDViQCRw7jJ/+2/cWfQe=|KS/xJmQ7UVi9JuIQmzSBZsUL580= ecDSA-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHdhYNTYAAAAIbmlzdHdhYNTYAAABBBK+WvgdSx6WIM/vpolfGTh  
d4KVsuxXaQ8k9KraYfXP8StkUEzPGFmpT8YgclLqEkcnjxwb+et7eXJ2VnQe/d3k=
```

Present in all data centers, SSH ships by default with every Unix, Linux and Mac server. SSH connections have been used to secure many different types of communications between a local machine and a remote host, including secure remote access to resources, remote execution of commands, delivery of software patches, and updates and other administrative or management tasks, manage routers, server hardware, virtualization platforms, operating systems (OSes), and inside systems management and file transfer applications.

Secure Shell is used to connect to servers, make changes, perform uploads and exit, either using tools or directly through the terminal. SSH keys can be employed to automate access to servers and often are used in scripts, backup systems and configuration management tools.

#### Note

SSH operates on **TCP port 22** by default (though SSH port can be changed if needed). The host server listens on port 22 (or any other SSH assigned port) for incoming connections. It organizes the secure connection by authenticating the client and opening the correct shell environment if the verification is successful.

It is hardly recommended to change the port in which the host server listens, because to let port 22 is more easily to enter the machine without authorization.

### 8.9.2 Authenticate SSH connections using user-password

By default SSH protocol doesn't use public key cryptography for authenticating hosts and users. It still uses the **Linux user-password authentication method**, so you only have to pass the **username** of the Linux user via the ssh command and introduce the **password** of this user:

```
$ ssh -p custom_port linux_username@hostname_or_ip
```

#### Note

If you run the command without specifying user, ssh will take the client server Linux current **username**.

### 8.9.3 Authenticate SSH connections using Asimetric Key authentication method

Generally, the SSH service is configured to use Asimetric Key cryptography for authenticating hosts and users. SSH introduced Asimetric Key authentication as a more secure alternative to the older **.rhosts** authentication. It improved security by avoiding the need to have password stored in files, and eliminated the possibility of a compromised server stealing the user's password.

However, SSH keys are authentication credentials just like passwords. Thus, they must be managed somewhat analogously to user names and passwords. They should have a proper termination process so that keys are removed when no longer needed.

#### 8.9.3.1 Generate public and private keys on the client server side

For that reason, we should generate SSH public and private key on our client and authorize them into the host.

Depending on the technology of the host you want to connect, you need to create the keys:

- **rsa**: an old algorithm based on the difficulty of factoring large numbers. A key size of at least 2048 bits is recommended for RSA; 4096 bits is better. All SSH clients support this algorithm.

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/key_name -C "your_email@example.com"
```

- **ed25519:** this is a new algorithm added in OpenSSH. Support for it in clients is not yet universal.

```
$ ssh-keygen -t ed25519 -f ~/.ssh/key_name -C "your_email@example.com"
```

### Note

Main differences between RSA and Ed25519:

- **Algorithm Type:** RSA is based on the difficulty of factoring large numbers, while Ed25519 uses elliptic curve cryptography.
- **Key Size:** RSA typically uses 2048 or 4096-bit keys; Ed25519 uses a fixed 256-bit key.
- **Security Level:** Ed25519 offers similar security to a 3072-bit RSA key with a smaller key size.
- **Performance:** Ed25519 is much faster in signing, verification, and key generation compared to RSA.
- **Adoption:** RSA is widely adopted and compatible with many systems; Ed25519 is gaining popularity, especially in modern protocols.
- **Use Cases:** RSA is common in SSL/TLS and email encryption, while Ed25519 is favored in SSH, DNSSEC, and other performance-critical applications.

They will ask you for a passphrase, depending on your security needs you can put one or not. If you introduce one, it will ask for them each time the keys are used.

### WARNING

For SSH to recognize the SSH keys they should be stored in:

```
~/.ssh/
```

```
acamp0s@BCNLT5CG3284PRF:~/work/opticlimb$ ls -la ~/.ssh/
total 24
drwxr-xr-x  2 acamp0s acamp0s 4096 Oct  4 14:02 .
drwxr-x--- 11 acamp0s acamp0s 4096 Oct  4 11:49 ..
-rw-----  1 acamp0s acamp0s 3369 Oct  4 10:51 id_rsa
-rw-r--r--  1 acamp0s acamp0s  733 Oct  4 10:51 id_rsa.pub
-rw-----  1 acamp0s acamp0s 2798 Oct  4 13:49 known_hosts
-rw-----  1 acamp0s acamp0s 2576 Oct  4 13:49 known_hosts.old
```

Also, **rsa keys** should have the **permissions** you see in the image above, if not, it will not be valid to authenticate in any server. If you have generated the keys, they are generated with the correct permissions, but if you have copied, it can be wrong.

### Note

If you have multiple rsa keys, it is hardly recommended to create a **known\_hosts** file for each account you have because it makes diagnosing issues easier when you have multiple keys. Ideally the name of this file is similar enough to the key name that you aren't confused later.

```
$ touch known_hosts_keyname
```

For more information, check the documentation of [how to do if for github servers](#).

### 8.9.3.2 Optional - Set up the ssh config file

The `~/.ssh/config` can contain specific configuration for each host. For example:

```
/.ssh/config

# Github
Host github.com
  Hostname github.com
  User git
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/github_key
  UserKnownHostsFile ~/.ssh/known_hosts_github
  IdentitiesOnly yes

# Bitbucket
Host bitbucket.org
  HostName bitbucket.org
  User git
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/bitbucket_key
  UserKnownHostsFiles ~/.ssh/known_hosts_bitbucket
  IdentitiesOnly yes
```

Here is the breakdown of what each line means:

- **Host** is a pattern matcher that is used to differentiate between these sets of configurations. Keep it the same as the **HostName** so it matches hosts in connections correctly without additional specification.
- **Host** The URL on the **HostName** line is the base URL where the repository resides. For example, if you have a personal account on github with personal projects, the URL will be github.com.
- **User** for git based systems will be git. The value of User will be different if you connect to something else (i.e. ec2-user for connecting to an Amazon AWS EC2 instance)
- **IdentityFile** asks for the location of the identity key we made. Type in the respective path here.
- **AddKeysToAgent** allows a private key that is used during authentication to be added to ssh-agent if it is running
- **UseKeychain** (macOS only) allows the computer to remember the password each time it restarts
- **UserKnownHostsFile** specifies an exact location to store all hosts you connect to when you're using that profile. Provide the respective paths here and choose a unique known hosts file name (see step 2 above) so that troubleshooting and key maintenance over time is easier.
- **IdentitiesOnly** specifies that only the keys provided must be used to connect to a host, even if another service like the ssh-agent offers a key for use.

### 8.9.3.3 Upload the Public Key into the authorized\_keys of the host server

To use public key authentication, the public key must be **installed in the** `authorized_keys` file of the server. This can be conveniently done using the `ssh-copy-id` tool. Like this:

```
$ ssh-copy-id -i ~/.ssh/my_ssh_key.pub user_name@hostname
```

#### WARNING

With the command above, we will append the host **pubkey** in the `authorized_keys` file of the server.

The authorized keys to enable an specific user accessing the server through ssh are stored in:

```
~/.ssh/authorized_keys
```

So you can copy your public key directly to this directory inside the server.

#### Note

For git you should do it different, just adding your public ssh key in the GUI, accessing to your profile settings. You can follow the [Official Documentation](#)

### 8.9.3.4 Configure the ssh service into the host server

To enable the SSH connection, the host server we want to connect should have the ssh service running and configured.

We can check if the `sshd` (ssh daemon) service is running. SSH Daemon listens for incoming SSH connections and handles authentication, encryption, and communications:

```
$ systemctl status sshd
```

#### WARNING

So not confuse it with the ssh service, which is the one used on the client-side to establish a secure connection to an SSH server.

The configuration file for the SSH Daemon is:

```
/etc/ssh/sshd_config
```

As we said before, by default SSH protocol doesn't use public key cryptography for authenticating hosts and users. So to enable it, we will need to modify this file in order to decomment the line:

```
/etc/ssh/sshd_config  
PubkeyAuthentication yes
```

And then restart the `sshd` service:

```
$ systemctl restart sshd
```



## NOTE

As well you can disable the password authentication:

```
/etc/ssh/sshd_config  
  
PasswordAuthentication no
```

To see all the configurable parameters in the `/etc/ssh/sshd_config` you can check

- [All configurable parameters in sshd file](#)
- Your template in ansible

### 8.9.3.5 Use the Asimetric Key authentication method using -i

By default, when you try to stablish ssh connection, if the host server enables PubkeyAuthentication, your ssh service will try to authenticate with all this pub keys:

- `~/.ssh/id_ecdsa.pub`
- `~/.ssh/id_ecdsa_sk.pub`
- `~/.ssh/id_ed25519.pub`
- `~/.ssh/id_ed25519_sk.pub`
- `~/.ssh/id_xmss.pub`
- `~/.ssh/id_xmss_sk.pub`
- `~/.ssh/id_dsa.pub`
- `~/.ssh/id_rsa.pub`

If you want to enforce the client server ssh service use just one, or a custom one different from those shown, you should:

```
$ ssh -p <Port_Number> -i /path/to/the/priv_key
```

## NOTE

You should have both keys on the same folder, public and private, if not, it will fail.

### 8.9.3.6 Use the ssh-agent for Asimetric Key

SSH agent is program which runs in the background and stores your decrypted private keys for SSH authentication. It helps you manage your SSH keys securely without having to enter their passphrase every time you use them. It is very useful as well when your keys have passphrase.

So in order to use to store your ssh private keys you should:

1. Check if it is running (it is not enabled when Boot by default so if you dont do it manually when boot the terminal it won't be running)

```
$ ps waux | grep -e "ssh-agent"
```

2. If it is not running (you don't have any process with ssh-agent), initiate it:

```
$ eval "$(ssh-agent -s)"
```

3. Add your private keys to the agent

```
$ ssh-add /path/to/the/priv/key
```

4. Run your ssh normally (without specifying private key -i)

#### 8.9.4 SSH Extra Arguments

Custom Port (by default is 22)

```
$ ssh -p <Port_Number>
```

Use specific pub\_key to authenticate

```
$ ssh -i /path/to/the/priv_key ...
```

Disable Check on the hosts file for this host

```
$ ssh -o StrictHostKeyChecking=no ...
```

Verbose Mode

```
$ ssh -v ...
```

Set timeout

```
$ ssh -o ConnectTimeout=30 ...
```

Using Proxy to establish the connection

```
$ ssh -o ProxyCommand="ssh -p <Proxy_Port_Number> -W %h:%p \  
-q <username>@<proxy_hostname_or_ip>"
```

#### 8.9.5 SFTP

##### 8.9.6 Introduction

FTP, or "File Transfer Protocol" was a popular unencrypted method of transferring files between two remote systems.

SFTP, which stands for SSH File Transfer Protocol, or Secure File Transfer Protocol, is a separate protocol packaged with SSH that works in a similar way but over a secure connection. The advantage is the ability to leverage a secure connection to transfer files and traverse the filesystem on both the local and remote system.

In almost all cases, SFTP is preferable to FTP because of its underlying security features and ability to piggy-back on an SSH connection. FTP is an insecure protocol that should only be used in limited cases or on networks you trust.

## Basic Usage

```
$ sftp -P <Port_Number> <username>@<hostname_or_ip>
```

### Set timeout

```
$ sftp -o ConnectTimeout=30 ...
```

You will be asked to insert the password if the sshd service is configured with password, so there is a way to insert it without be asked.

### Send te password without interaction

```
$ sshpass -p "your_password" sftp -P ...
```

## 8.9.7 Commands inside the SFTP

Once you are inside the sftp you can execute the follogin commands:

### 8.9.7.1 Navigate

#### Ask for help

```
sftp> help
```

#### Navigate through directories

```
sftp> pwd
```

```
sftp> ls -la
```

```
sftp> cd /the/path/you/want
```

Create a directory

```
sftp> mkdir folder
```

#### NOTE

You wont be able to remove it from sftp, you need to access the server via ssh and remove it

#### Displays a folder/file disk space usage inside host server

```
sftp> df -h folder_name
```

#### Execute commands in your local machine (client server)

If you include "!" before your command, you will execute this command on your local linux machine

```
sftp> !ls
```

#### 8.9.7.2 Get

Get file from host server to client server

```
sftp> get /path/to/the/remote/file /path/to/the/local/file
```

Get directory from host server to client server

```
sftp> get -Pr /path/to/the/remote/directory /path/to/the/local/directory
```

#### NOTE

The -P is to maintain the appropriate permissions and access times

#### 8.9.7.3 Put

Upload client server file to host server

```
sftp> put /path/to/the/local/file /path/to/the/remote/file
```

Upload directory from client server to host server

```
sftp> put -Pr /path/to/the/local/directory /path/to/the/remote/directory
```

#### NOTE

The -P is to maintain the appropriate permissions and access times

#### 8.9.7.4 Exit

```
sftp> bye
```

```
sftp> exit
```

```
sftp> !
```

### 8.9.8 Being sudo with SFTP?

In general, the sudo command is not directly applicable to SFTP (SSH File Transfer Protocol) because SFTP is a protocol specifically designed for transferring files securely over SSH (Secure Shell). However, if you want to perform administrative tasks on a remote server using SFTP, you typically need to have the appropriate permissions granted to your SSH user account.

If you need to perform administrative tasks on files through SFTP, you would typically ensure that the user account you're logging in with has the necessary permissions to read, write, and modify those files.

```
$ ssh ... username@remote_server
```

```
$ sudo command_to_execute
```

```
$ sftp ... username@remote_server
```

## 8.10 Localhost Networking

### 8.10.1 Ports Exposed

Para saber que puertos tiene escuchando nuestro localhost:

#### 1. Using `lsof`

```
$ sudo lsof -i -P -n
```

```
$ sudo lsof -i -P -n | grep -i listen
```

#### 2. Using `netstat`

```
$ netstat -putona | grep numero-de-puerto
```

- **p:** muestra las conexiones para el protocolo especificado que puede ser TCP o UDP
- **u:** lista todos los puertos UDP
- **t:** lista todos los puertos TCP
- **o:** muestra los timers
- **n:** muestra el numero de puerto
- **a:** visualiza todas las conexiones activas del sistema

#### NOTE

`netstat` command shows the system's network information, like routing and sockets.

### 8.10.2 Localhost Adresses

De puertas para adentro

#### 1. Ipv4:

- localhost
- 127.0.0.1

#### 2. Ipv6

- 0:0:0:0:0:0:1

- ::1

## De puertas para fuera

- `ifconfig`: displays the system's network interfaces and their configurations. In this case our localhost direction to outside is 172.21.220.125/20 (172.21.220.125 with mask 255.255.240.0)

```
$ ifconfig
```

```
titocampis@DESKTOP-FMGU3N7:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.21.220.125 netmask 255.255.240.0 broadcast 172.21.223.255
    inet6 fe80::215:5dff:feab:2d48 prefixlen 64 scopeid 0x20<link>
    ether 00:15:5d:ab:2d:48 txqueuelen 1000 (Ethernet)
    RX packets 63 bytes 8021 (8.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7 bytes 586 (586.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 6: `ifconfig`

- `ip -br addr list eth0`

```
titocampis@DESKTOP-FMGU3N7:~$ ip -br addr
lo                UNKNOWN    127.0.0.1/8 ::1/128
bond0             DOWN
dummy0            DOWN
tunl0@NONE        DOWN
sit0@NONE         DOWN
eth0              UP          172.21.220.125/20 fe80::215:5dff:feab:2d48/64
titocampis@DESKTOP-FMGU3N7:~$
```

Figure 7: `ip -br`

## Nota

### *Netmask*

24 (number of 1's):

255.255.255.0

11111111 11111111 11111111 00000000

20 (number of 1's):

255.255.240.0

11111111 11111111 11110000

## 9 Network Tools

### 9.1 Ping

Ping is a network utility used to test the reachability of a host on an Internet Protocol (IP) network and measure the round-trip time for packets sent from the source to the destination host and back. It works by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for ICMP echo reply packets in response.

#### Basic Usage

```
$ ping <hostname_or_ip>
```

#### Send specific number of packages

```
$ ping -c <number_of_packages> ...
```

#### Set interval between sending ping requests in seconds

```
$ ping -i <interval_seconds> ...
```

#### Set timeout for packages

```
$ ping -t <timeout_seconds> ...
```

#### Set timeout for ping (deadline in seconds)

```
$ ping -W <timeout_seconds> ...
```

#### Stopping ping

```
$ Ctrl+C
```

#### WARNING

Use ping to "ping" an IP address doesn't necessarily mean that you can access services like SSH, HTTP, database, HTTPS, or others on that IP. Ping operates at the ICMP (Internet Control Message Protocol) level, which is a lower level than the protocols used by services like SSH (Secure Shell), HTTP or database (TCP), HTTPS (HTTP over SSL/TLS), or others.

So ping is not a good method to check the connection to a specific service running on a host. You can use it to check if you can resolve / reach a host, but not to check if you can reach a database inside this host. Because the network and firewall rules are usually defined by protocol, they use to allow ICMP echo requests (which is what ping uses) for diagnostic purposes, but blocks other types of traffic such as SSH, TCP, HTTP, HTTPS... for security reasons.

### 9.2 Curl

Curl (Client for URL's) is a command-line tool and library for transferring data with URLs. It is mainly intended for testing HTTP and HTTPS connections, but it supports various protocols more: FTP, FTPS, SCP, SFTP, LDAP, TELNET, and more.

Curl is widely used for testing web services, downloading / uploading files, and automating data transfers in scripts and applications.

It is a powerful tool for accessing and transferring data over networks, with support for multiple protocols and versatile capabilities. But for network connectivity, it is not so suitable tool, because there are many protocols not supported by curl.

It can happen that with curl we get a connection error and it is not because the connection itself is not open, but because we are not using the correct protocol.

### Basic Usage

```
$ curl <url>
```

### Using proxy

```
$ curl -x <proxy_url> <url>
```

### Forcing no proxy

```
$ curl --noproxy "*" <url>
```

### With specific timeout in seconds

```
$ curl --connect-timeout 5 ...
```

### Not show content, only info

```
$ curl -I ...
```

### Follow redirects

To have more context, when you make an HTTP request to a server, it might respond with a redirect status code (such as 301 Moved Permanently or 302 Found) along with the new URL to which the request should be redirected. By default, tools like curl won't automatically follow these redirects; instead, they'll stop and return the redirect response to the user.

When you use the -L option with curl, it instructs curl to follow HTTP redirects automatically. So, when curl encounters a redirect response (e.g., a 301 or 302 status code), it will automatically make a new request to the URL specified in the redirect response.

```
$ curl -L ...
```

### Force the protocol

```
$ -O,      --http1.0      Use HTTP 1.0
           --http1.1      Use HTTP 1.1
           --http2        Use HTTP 2
           --http2-prior-knowledge Use HTTP 2 without HTTP/1.1 Upgrade
           --http3        Use HTTP v3
```

### Download a file via https

```
$ curl -O https://url/path/to/file
```



### Download a file to specific path

```
$ curl -o /path/to/the/filename https://url/paht/to/file
```

### Upload a file using https protocol

```
$ curl -F "file=@/path/to/local/file.txt" https://example.com/upload
```

### Upload a file using ftp protocol

```
$ curl -T /path/to/local/file.txt ftp://ftp.example.com/upload/
```

### Target different port than 443 or 8080

```
$ curl domain:port
```

### Verbose

```
$ curl -v
```

### HTTPS Protocol (port 443)

```
$ curl https://hostname
```

```
$ curl hostname:443
```

### Allow insecure connections, skip certificate validation

```
$ curl -k hostname
```

### Authentication

```
$ curl -u user_name:pwd hostname
```

### Pass Headers

```
$ curl -H "Content-Type:application/json" hostname
```

### GET HTTP Requests

```
$ curl -XGET -H hostname
```

### PUT HTTP Requests

```
$ curl -XPUT hostname
```

### POST HTTP Requests

```
$ curl -XPOST hostname
```

## POST HTTP Requests

```
$ curl -XDELETE hostname
```

## 9.3 Nmap

Nmap, short for Network Mapper, is a powerful open-source tool primarily used for network discovery and security auditing. It's designed to provide a comprehensive overview of the devices and services running on a network.

It helps in identifying hosts, devices, and services running on a network. It provides essential information such as IP addresses, MAC addresses, and open ports. One of its core functionalities is port scanning, which involves probing a target host to determine which ports are open, closed, or filtered. This information is crucial for understanding the network's security posture and potential vulnerabilities.

Nmap can detect the version and type of services running on open ports. This capability allows administrators to assess the software stack of networked devices and identify outdated or vulnerable services. Nmap can often determine the operating system (OS) running on a target host based on subtle differences in network stack implementations and responses to specific probes. This feature aids in network mapping and security assessment.

### Basic TCP port scan on a specific host

```
$ nmap <hostname_or_ip>
```

### Discover a host on a network

```
$ nmap -sn <hostname_or_ip>
```

### Scan specific ports

```
$ nmap -p <port_name> <hostname_or_ip>
```

```
$ nmap -p 80,443,... <hostname_or_ip>
```

### Scan range of ports

```
$ nmap -p 80-443 <hostname_or_ip>
```

### Scan all ports

```
$ nmap -p- <hostname_or_ip>
```

### Scan the 100 more used ports

```
$ nmap -F <hostname_or_ip>
```

### Detect the operating system of the target host

```
$ nmap -O <hostname_or_ip>
```

**Do not check using ping if the host is up, just try the connection to ports**

It is really useful when the host blocking our pings

```
$ nmap -Pn <hostname_or_ip>
```

**Scan all the ports opened of the target host**

```
$ nmap -sT -T Aggressive <hostname_or_ip>
```

#### NOTE

The services shown in the output of nmap are not exact, it is what is commonly exposed on the port, but it cannot know what is exactly exposed on the ports.

## 9.4 Netcat

Netcat, often referred to as the "Swiss Army Knife" of networking tools, is a versatile utility for reading from and writing to network connections using TCP/IP protocols. It can act both as a client and a server. It can establish connections, listen for connections, and transfer data in various ways.

**Check if a port is open on a remote server**

```
$ nc -zv <hostname_or_ip> <port_number>
```

```
$ nc -zv <hostname_or_ip> 80,443,...
```

**Check range of ports**

```
$ nc -zv <hostname_or_ip> 80-443
```

**Send a file**

```
$ nc <hostname_or_ip> <port> > filename
```

**Receive a file**

```
$ nc <hostname_or_ip> <port> < filename
```

**Starts a netcat process that listens for incoming connections on a port**

Listen mode (-l)

```
$ nc -l <port>
```

**Attempt to establish a TCP connection to the specified hostname and port**

If successful, it will provide a channel for communication between the two machines

```
$ nc <hostname_or_ip> <port>
```

**Starts a netcat listener that listens for incoming connections and executes `/bin/bash` (a Bash shell) upon connection, providing the connecting client with an interactive shell.**

Listen mode (-l). It can pose serious security risks if not properly secured, as it allows arbitrary command execution on the system.

```
$ nc -lp <port> -e /bin/bash
```

## 9.5 Telnet

Telnet stands for "teletype network," and it's a network protocol used to access and manage devices remotely over a network. It's one of the oldest protocols used on the internet, dating back to the late 1960s.

Telnet operates by establishing a connection between a local computer (the client) and a remote computer (the server). Once the connection is established, users can interact with the remote system as if they were physically present at the terminal of that system.

**Check port on a host**

```
$ telnet <hostname_or_ip> <port>
```

If the server is running and accepting connections, Telnet will establish a connection, and you'll see a blank screen. You can then manually send an HTTP request to the server and observe the response:

```
GET / HTTP/1.1  
Host: example.com
```

After sending the request, the server should respond with the appropriate HTTP response, indicating whether the server is working correctly.

### WARNING

Remember that Telnet transmits data in plain text, so it's not secure for transmitting sensitive information like passwords. For secure remote access, it's recommended to use protocols like SSH instead.

## 9.6 Traceroute

**traceroute** is a network diagnostic tool used to trace the path that packets take from your local machine to a specified destination, showing all intermediate routers (hops) between the source and destination. By revealing the path, traceroute helps you understand where potential bottlenecks, delays, or network issues might be occurring.

Each router (or hop) along the way is queried, and traceroute reports the time it takes for a packet to travel from your machine to that router. It is an essential tool for troubleshooting routing issues or slow network connections.

**traceroute** sends a series of ICMP (Internet Control Message Protocol) or UDP (User Datagram Protocol) packets to the target with varying TTL (Time-to-Live) values. The TTL field limits the number of hops a packet can pass through before it gets discarded. Each time a router decreases the TTL by one and sends back a message, traceroute records the delay and the router's address.

Basic Usage:

```
$ traceroute <ip_or_hostname>
```

Limits the maximum number of hops. By default, traceroute will trace up to 30 hops:

```
$ traceroute -m <max_hops> <ip_or_hostname>
```

Use ICMP Instead of UDP:

```
$ traceroute -I <ip_or_hostname>
```

Display IP Addresses Only:

```
$ traceroute -n <hostname/IP>
```

Set Timeout for Each Response:

```
$ traceroute -w <wait_time> <hostname/IP>
```

## 9.7 MTR

**mtr** (My Traceroute) is a powerful network diagnostic tool that combines the functionality of two essential network utilities: ping and traceroute. It provides real-time network performance statistics, helping users identify potential network issues like packet loss, latency, and routing problems between the local machine and a remote host.

Unlike traditional ping or traceroute which give static outputs, mtr runs continuously, providing updated information and live network analysis. It is especially useful for troubleshooting network connectivity issues or analyzing the performance of network routes over time.

When you run mtr, it sends ICMP (Internet Control Message Protocol) echo requests to the target host (like ping) and tracks the path that packets take to reach the destination (like traceroute). It collects and displays real-time statistics on packet loss, response times, and the number of hops (routers) the packets pass through.

Basic usage:

```
$ mtr <hostname_or_ip>
```

Limit the number of pings:

```
$ mtr -c <number> <hostname_or_ip>
```

Run mtr as Root for ICMP (Privileged Mode). Running mtr as root enables the program to send ICMP packets for more accurate results. Without root privileges, mtr sends UDP packets, which may behave differently.

```
$ sudo mtr <hostname_or_ip>
```

Verbose:

```
$ mtr -rw <hostname_or_ip>
```

Show IP address only:

```
$ mtr -n <hostname_or_ip>
```

Specify max number of hops:

```
$ mtr -m <number_hops> <hostname_or_ip>
```

## 9.8 Whois

WHOIS is a protocol used to query databases that store information about **domain names**, **IP addresses**, and **autonomous system numbers**. It helps to retrieve **registration details** of a domain name or IP address, such as the **owner**, **registrar**, **creation/expiration dates**, and more. WHOIS is widely used by **network administrators**, **security professionals**, and even **regular users** to check the **legitimacy** of websites, or to identify who owns a particular domain name.

WHOIS queries are made over a **network** to a WHOIS server, which returns the requested data. Many **domain name registrars** maintain WHOIS servers to provide this information to the public.

Basic usage:

```
$ whois <hostname_or_ip>
```

Specify a particular WHOIS server to query:

```
$ whois -h whois.verisign-grs.com <hostname_or_ip>
```

Short output:

```
$ whois -s <hostname_or_ip>
```

WHOIS data without any formatting:

```
$ whois -r <hostname_or_ip>
```

## 9.9 Proxy variables definition for binaries

Configuring the proxy in the system environment variables with a specific naming convention will cause many system binaries to use them to establish their connections. For example: docker, curl, nslookup...

Include the following variables in the file `/etc/bash.bashrc`:

```
# Configure Default Proxy
export http_proxy=http://yourproxyserver.com
export https_proxy=http://yourproxyserver.com
export HTTP_PROXY=http://yourproxyserver.com
export HTTPS_PROXY=http://yourproxyserver.com
export NO_PROXY=localhost
export no_proxy=localhost
```

## 10 Terminal

### 10.1 Ctrl+r

**Ctrl+r** initiates a reverse search through your command history. This allows you to quickly find and execute a previously used command without having to retype it completely.

Press **Ctrl+r** repeatedly to cycle backward through the matching commands.

If you overshoot and want to go forward, you can use **Ctrl+s**

To edit the command: **right arrow**

To execute the command:

Enter

Ctrl+j

To exit the search:

Esc

Ctrl+g

### 10.2 Bash

#### 10.2.1 Introduction

When a computer boots up, the kernel recognizes all the physical hardware and enables each component to talk with one another and be orchestrated by some basic software. A computer's most basic set of instructions simply keeps it powered on and in a safe state.

Computer scientists developed a shell for Unix computers that operates outside of the kernel (or around the kernel, like a shell in nature) and allows humans to interact with the computer whenever they want to. It was an exciting development at a time when people were feeding punchcards into computers to tell them what to do. Of all the shells available, Bash is one of the most popular, the most powerful, and the most friendly.

When you start a terminal running the shell, you're greeted with a prompt. A prompt is a symbol, usually a dollar sign (\$), indicating that the shell is waiting for your input. Bash (Bourne Again SHell) is the default shell on many Linux distributions, it's a powerful and flexible shell that provides users and administrators with the tools to interact with the operating system, automate tasks, and customize their working environment.

Bash is also a command, and it's usually the default command executed when you open a terminal window or log into a text console.

#### 10.2.2 Configuration .bashrc

The **.bashrc** file is a script that Bash runs whenever a new terminal session is started in interactive mode (non-login shells). It is typically located in a user's home directory (e.g., `/home/username/.bashrc`). The file itself contains a series of configurations for the terminal session. This includes setting up or enabling: coloring, completion, shell history, command aliases, and more.

### 10.2.3 Path

On Linux and Unix, most commands are stored by default in system directories like `/usr/bin` and `/bin`. By nature, Bash doesn't know these commands any more than you naturally know Klingonese, but just as you can look up Klingon words, Bash can look up commands. When you issue a command to Bash, it searches specific directories on your system to see whether such a command exists. If the command does exist, then Bash executes it.

`PATH` is an environment variable that specifies a list of directories where the shell looks for executable files when you type a command. This allows users to run programs and scripts without needing to provide the full path to their location.

The `PATH` variable can be defined and modified in several places, depending on the scope and persistence you desire:

- **System-wide:** `/etc/profile`, `/etc/environment`, `/etc/bash.bashrc`.
- **User-specific:** `~/.profile`, `~/.bash_profile`, `~/.bashrc`, `~/.zshrc` (if using Zsh shell).

#### Warning

`PATH` environment variable can potentially be modified by some other configuration files, in order to ensure for some binaries that the necessary executables are available for the services or jobs they manage. Any changes made will be scoped to the environment of the scripts or services they are associated with. Here are a list of just some of files where `PATH` can be modified.

- `/etc/cron.d/certbot`
- `/etc/cron.weekly/man-db`
- `/etc/init.d/plymouth`
- `/etc/init.d/docker`
- `/etc/init.d/kmod`
- `/etc/init.d/rsync`
- `/etc/init.d/uuid`

Just to clarify, it is always extended, they do not remove the previous content.

Check the current value of `PATH`:

```
$ echo $PATH
```

**Temporary** extend `PATH` variable for the **current session**:

```
$ export PATH=$PATH:/new/directory/path
```

**Permanently** extend `PATH` variable for the **current user**, add the following lines to `~/.bashrc` or `~/.zshrc` (if using Zsh shell):

```
./bashrc  
  
$ export PATH=$PATH:/new/directory/path
```

**Permanently** extend `PATH` variable for **all users**, add the following lines to `/etc/profile` or `/etc/zsh/*` (if using Zsh shell):



```
/etc/profile
```

```
$ export PATH=$PATH:/new/directory/path
```

## 10.3 ZSH, why should it be used instead of bash?

### 10.3.1 What is ZSH?

**ZSH** (Z SHell), is an extended version of the Bourne Shell (sh), with new features and support for plugins and themes. Since it's based on the same shell as Bash, ZSH has many of the same features, and switching over is a breeze. It is known for its powerful scripting capabilities, interactive features, and customization options, making it a popular choice for both scripting and daily use.

**Bash** (Bourne Again SHell) is the default shell for many Linux distributions and macOS but has fewer interactive features compared to Zsh.

### 10.3.2 ZSH vs Bash

- Zsh has more advanced scripting capabilities, including better handling of arrays and associative arrays.
- Zsh offers features like spell correction, approximate command correction, and shared command history.
- Advanced array handling, extended globbing, and improved scripting syntax make Zsh more powerful for writing scripts.
- Zsh offers an advanced and programmable completion system that is highly customizable.
- Zsh supports themes and plugins via frameworks like Oh My Zsh, making it easy to extend and customize.

### 10.3.3 How to install and enable zsh?

1. Install zsh:

```
$ sudo apt install zsh
```

2. Make zsh the Default Shell:

```
$ chsh -s $(which zsh)
```

3. Verify the change:

```
$ echo $SHELL
```

### 10.3.4 Configure ZSH

The recommendation is to use an already builtin zsh full configuration, as **ohmyzsh**. You can follow the [documentation](#).

However, if you want to configure yourself zsh, you need to know that there are different shell configurations, which explains why Zsh supports four different configuration files. Here's how the configuration files are used:

- `~/.zshrc`: Loaded only for interactive shell sessions. It is loaded whenever you open a new terminal window or launch a subshell from a terminal window. This is the configuration file that most developers use.
- `~/.zshenv`: This is loaded universally for all types of shell sessions (interactive or non-interactive, login or non-login). It is the only configuration file that gets loaded for non-interactive and non-login scripts like cron jobs. However, macOS overrides this for PATH settings for interactive shells, universally loaded, so you could use it to configure the shell for automated processes like cron jobs. However, it is best to explicitly set up environmental variables for automated processes in scripts and leave nothing to chance.
- `~/.zprofile`: Loaded for login shells (both interactive and the rare non-interactive sessions). macOS uses this to set up the shell for any new terminal window. Subshells that start within the terminal window inherit settings but don't load `~/.zprofile` again.
- `~/.zlogin`: Only used for login shell configurations, loaded after `.zprofile`. This is loaded whenever you open a new terminal window.

```

/.zshrc

# Set the prompt
PROMPT='%n%m:%~%# '

# Enable command auto-correction
setopt correct

# Enable extended globbing
setopt extended_glob

# Set up aliases
alias ll='ls -l'
alias la='ls -A'
alias l='ls -CF'

# Enable syntax highlighting (requires installation of zsh-syntax-highlighting)
# source /path/to/zsh-syntax-highlighting/zsh-syntax-highlighting.zsh

# Enable autosuggestions (requires installation of zsh-autosuggestions)
# source /path/to/zsh-autosuggestions/zsh-autosuggestions.zsh

# Load custom scripts or functions (if any)
# source ~/.zsh_custom

# Add your custom configurations below
export PATH=$HOME/bin:/usr/local/bin:$PATH
\begin{verbatim}

```

## 10.4 Tmux

Tmux, short for "Terminal Multiplexer", is a powerful command-line tool that enhances the capabilities of your terminal sessions. It allows you to organize multiple terminal windows or sessions within a single terminal window, effectively turning your terminal into a multi-pane workspace.

With Tmux, you can create multiple independent terminal sessions within a single window. This means you can work on different tasks or projects simultaneously without cluttering your desktop with multiple terminal windows.

As well tmux enable separate the execution from terminal, so if you lose the connection, what you launched continues executing.

### 10.4.1 Tmux Socket

A socket in tmux is essentially a file that facilitates communication between the tmux server (the backend process) and tmux clients (the user interfaces that you interact with).

By default, tmux uses a socket named default located in a directory like `/tmp/tmux-1000/default` where 1000 is your user ID. When you start tmux without specifying a socket, it uses this default socket.

Start tmux with a specific socket:

```
$ tmux -L <session_name>
```

#### Note

By default, tmux uses a socket named default located in a directory like `/tmp/tmux-1000/default`. When you run `tmux -L session_name` tmux will create a new socket named `<session_name>` located in a directory like `/tmp/tmux-0/<session_name>`.

Without creating a session in a different socket from default, configuring the tmux session it is not possible.

### 10.4.2 Arguments

List the current running sessions

```
$ tmux ls
```

**-L [socket\_name]**

```
$ tmux -L <session_name>
```

List all tmux sessions on a specific socket:

```
$ tmux -L <session_name> ls
```

**-2**

Tells Tmux to start in 256-color mode. Tmux supports both 256-color and 16-color modes for terminal color support. The -2 option enables the 256-color mode.

```
$ tmux -2
```

#### Configuration:

Tmux by default uses two configuration files, and it's able to merge the content of them. By default, user configuration overrides conflicting configurations:

- `/etc/tmux.conf`: system-wide configuration file.
- `~/.tmux.conf`: user-specific configuration file.

So if we want to use a different file:

```
$ tmux -f /path/to/file -2 -L <session_name>
```

### Warning

When you start tmux without `-L`, if there is already a tmux server running, new sessions will join the existing server and will not create a new one unless you specify a different server socket using the `-L` option.

My custom config for tmux is:

*/.tmux.conf*

```
#####
# MAIN OPS
#####

setw -g mode-keys vi

# bind to ctrl-a instead of ctrl-b
set-option -g prefix C-a
unbind C-b
bind-key C-a last-window

setw -g monitor-activity on
set -g visual-activity on

#####
# STATUS BAR
#####
# general
set -g status-interval 10
#set -g display-time 3000
set -g status-bg black
set -g status-fg yellow
# left
set -g status-left-length 30
set -g status-left " #[fg=green]#H "
# right
#set -g status-right-bg green
set -g status-right-length 120
#set -g status-right "#[bg=colour99, fg=colour230]#(free -m|egrep -i "^mem")#(uptime)"
set -g status-right "#[bg=colour99, fg=colour230]#(uptime)"
#set -g status-right "#(uptime | awk -F\: '){print $5}'); ; #(date)"

# Statusbar properties.
#set -g status-right-fg red
#set -g status-right "#(uptime | awk -F\: '){print $5}'); ; #(date)"

#####
# PANES
#####
set-option -g display-panes-active-colour red
set-option -g display-panes-colour colour246
set-option -g display-panes-time 1000

#set-option -g pane-active-border-style "fg=green bold"
#set-option -g pane-border-status top
#set-option -g pane-border-format "Pane #D Index #P"
set -g display-panes-time 800 # slightly longer pane indicators display time
set -g display-time 1000      # slightly longer status messages display time

#####
# MOUSE
#####
#set -g mode-mouse on
#set -g mode-mouse off
```

In tmux, when we want to go to **command mode**, by default we need to use **Ctrl+b**. But we are going to configure it as **screen**, with **Ctrl+a**, it is configured in the config file.

### attach

Attach to tmux session X, it is used when you want to join an already created tmux session:

```
$ tmux -f /path/to/file -2 -L attach <session_name>
```

Deattach other sessions

```
$ tmux -L <session_name> -d
```

If you want to deattach the other sessions and attach yourself:

```
$ tmux -f /path/to/file -2 -L attach <session_name> -d
```

### kill

Kill all tmux sessions:

```
$ tmux kill-server
```

Kill just a custom session:

```
$ tmux -L mycustomsocket kill-session
```

### Warning

Tmux does not synchronize the history with the main process in **bash**, so if the socket is ended you will lose all your history. History is synchronized if you use **zsh**.

## 10.4.3 Commands Inside tmux

### General

Command	What it does?
CTRL-a c	New tmux tab
CTRL-a n	Next tab
CTRL-a p	Prev tab
CTRL-a n	Next tab
CTRL-a repag	Go up into the logs
CTRL-d (or exit)	exit current tab
CTRL-a &	Destroy current tab
CTRL-a x	Destroy current tab
ctrl-a :	Access tmux config mode (with auto-complete)
CTRL-a ,	change current tab name
CRTL-a d	de-attach tmux

## Splitting

Command	What it does?
CTRL-a “	Split horizontal
CTRL-a %	Split vertical
CTRL-a <arrow-keys>	Move between tabs
CTRL-a CTRL-<arrow-keys>	Resize tabs

### In Repag Mode:

- **space**: start selecting.
- **arrows**: to move on the selection.
- **mouse wheel**: to move on the selection.
- **Esc**: stop selection.
- **Enter**: copy the selection.

### Config Mode (Ctrl-a :)

Basically you can configure the same as in the config file

Synchronize stdin on all tmux tabs:

```
setw synchronize-panes on / off
```

## 10.5 Ohmyzsh

### 10.5.1 Introduction

Oh My Zsh is an open-source, community-driven framework for managing your Zsh (Z shell) configuration. Zsh is a powerful and highly customizable shell, and Oh My Zsh simplifies the process of configuring and maintaining it. Oh My Zsh provides a robust set of features including:

- **Themes**: A wide variety of themes to personalize your terminal prompt.
- **Plugins**: An extensive collection of plugins that enhance functionality, such as Git integration, syntax highlighting, auto-suggestions, and more.
- **Customization**: Easy to customize with your own functions, aliases, and plugins to suit your workflow.

### 10.5.2 How to install it?

#### zsh installed

Ensure you have Zsh installed on your system. You can check by running:

```
zsh --version
```

If Zsh is not installed, you can install it using your package manager. For example:

- On Debian-based systems (like Ubuntu)

```
sudo apt install zsh
```

- On Red Hat-based systems (like Fedora)

```
sudo dnf install zsh
```

## Install ohmyzsh

The simplest way to install Oh My Zsh is by using the installation script. Open your terminal and run the following command:

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

The installation script will clone the Oh My Zsh repository into your home directory (`~/.oh-my-zsh`), set Zsh as your default shell, and create a new Zsh configuration file (`~/.zshrc`).

**Follow the on-screen prompts to complete the installation and restart your terminal**

### 10.5.3 Post-Installation

#### Selecting a Theme

You can change your theme by editing the `ZSH_THEME` variable in your `~/.zshrc` file. For example:

```
ZSH_THEME="agnoster"
```

#### Set colors for `LS_COLORS`

```
eval 'dircolors ~/.dircolors'
```

**Plugins** Which plugins would you like to load? Standard plugins can be found in `$ZSH/plugins/`. Custom plugins may be added to `$ZSH_CUSTOM/plugins/`

#### NOTE

Choose wisely, as too many plugins slow down shell startup.

```
plugins=(
  git
  zsh-autosuggestions
  zsh-syntax-highlighting
  history
  sudo
  web-search
  copypath
  copyfile
)
```



## 11 Linux Secondary Storage

### 11.1 Introduction

In the realm of computer storage, there are **two primary types**: the primary storage and the secondary storage.

**Primary storage** usually refers to the volatile memory that temporarily stores data while the system operates, such as random access memory (RAM).

On the other hand, **secondary storage** refers to the non-volatile type that provides long-term storage for digital data.

### 11.2 Drives

A drive, in the context of Linux, refers to a **physical storage device** that can store data. Specifically, it refers to a **single physical unit of digital storage media**, such as a hard disk drive or a solid-state drive (SSD). Unlike partitions and volumes, the word itself doesn't imply any form of logical division or grouping of storage.

#### Note

Etymologically, the word comes from the physical driver that rotates the disks of the storage for seeking purposes. In early days, non-volatile computer storage consisted of spools of magnetic strips mounted onto a tap driver. As the storage technology advanced, the drive mechanism seems to have stuck around as we can see in the floppy disk and hard disk drive.

Although newer storage device such as SSD no longer has a moving driver, the word “drive” is still used. Specifically, people continue to use the word “drive” to refer to the actual physical storage device, be it a spinning hard disk drive or an SSD.

Linux represents the drives as pseudo files with the **prefix sdx under /dev directory**. For example:

- First physical drive we attach to the system: `/dev/sda`
- Second physical drive we attach to the system: `/dev/sdb`
- Third physical drive we attach to the system: `/dev/sdc`
- ...

### 11.3 Partitions

Partition is a **logical division or subdivision** of a physical storage drive. Specifically, given a single physical storage drive, we can **divide it into several smaller independent and isolated units of storage**, known as partitions.

The key distinction between a partition and a drive lies in their relationship and functionality. While a drive represents the physical storage device itself, a partition is an abstraction layer on top of that drive, serving as a logical division within it.

For example, we can divide a single 1TB physical SSD we have in the system into two different partitions. Then, the system can treat the two partitions as separate storage mediums, each with 500GB of storage space, if we partition the drive equally.

One benefit of partitioning a physical storage drive is that the **partitions are independent and isolated**. The isolation of different partitions means that we can install different operating systems onto the same drive under different partitions.

Besides that, with the partitions being separate storage entities, we can also format the partitions with a different file system from one another. This means we can afford to install multiple operating systems onto the same physical storage drive by partitioning.

Linux represents the drives as pseudo files with the prefix `sdx` under `/dev` directory. For example:

- First partition of physical drive a: `/dev/sda1`
- Second partition of physical drive a: `/dev/sda2`
- First partition of physical drive c: `/dev/sdc1`
- ...

## 11.4 Filesystems

A filesystem is a method and data structure that an operating system uses to manage files on a disk or partition. It defines how data is stored, organized, and retrieved. Without a filesystem, an operating system cannot keep track of the files and data stored on a disk, making it impossible to read or write data efficiently.

Why do we need filesystems? When you create a new partition, it's essentially a blank space on the disk with no structure or organization. Formatting the partition with a filesystem prepares it to store data in a structured way that the operating system can understand. Here are the reasons why formatting is necessary:

- **Preparation of Storage Space:** formatting sets up the initial structure on the partition, creating a table of contents for files and directories. This allows the operating system to keep track of where files are located on the disk.
- **Organization of Data:** a filesystem provides a structured way to store and organize data in files and directories. This structure allows easy navigation and management of files.
- **Data Management:** filesystems manage how data is stored and retrieved. They ensure that files do not overlap and that free space on the disk is utilized efficiently.
- **Access Control:** filesystems often include permissions and access control mechanisms to ensure that only authorized users can access or modify specific files.
- **Data Integrity:** filesystems provide mechanisms to detect and correct errors. Some advanced filesystems also offer features like journaling, which helps in recovering from crashes or power failures.
- **Metadata Management:** filesystems store metadata, which includes information about files such as their names, sizes, creation dates, and permissions. This metadata is crucial for the operating system to manage files properly.

## 11.5 LVM

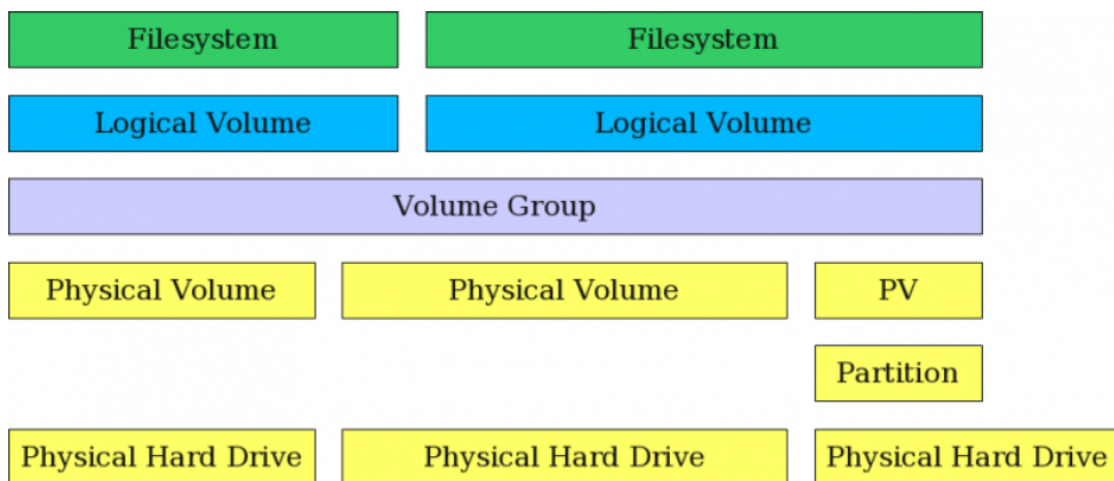
### 11.5.1 Introduction

Logical Volume Management (LVM) is a system for managing disk storage space in a more flexible manner than traditional partitioning methods. It allows system administrators to create, resize, and move storage volumes dynamically without the need for downtime. LVM is particularly useful in environments where storage requirements are constantly changing and flexibility is paramount.

### 11.5.2 LVM Structure

The structure of a Logical Volume Manager disk environment is illustrated below. **Logical Volume Management enables the combining of multiple individual hard drives and/or disk partitions into a single volume group (VG).** That volume group can then be **subdivided into logical volumes (LV)** or used as a single large volume. Regular **file systems**, such as EXT3 or EXT4, can then be created on a logical volume.

In Figure below, two complete physical hard drives and one partition from a third hard drive have been combined into a single volume group. Two logical volumes have been created from the space in the volume group, and a filesystem, such as an EXT3 or EXT4 filesystem has been created on each of the two logical volumes.



- **Physical Volumes (PV's):** basic storage devices that LVM manages. They can be an entire hard drive, a partition on a hard drive, or a RAID array. PVs are initialized for use by LVM using the `pvcreeate` command, which writes metadata to the physical volume so that it can be used by LVM.
- **Volume Groups (VG's):** pool of storage that consists of one or more physical volumes. When you create a VG, you aggregate the storage capacity of all included PVs, making it available for creating logical volumes. The VG acts as a container for logical volumes.
- **Logical Volumes (LV's):** virtual block device that can be used like a regular partition. LVs are created from the storage pool provided by a volume group. You can create, resize, and manage logical volumes without worrying about the underlying physical storage details.
- **Filesystems:** Once a logical volume is created, it must be formatted with a filesystem before it can be used to store files. After formatting, the filesystem can be mounted to a directory in the operating system, making it accessible for reading and writing data.

## 11.6 Tools to Manage Secondary Storage

### 11.6.1 Lsblk - List block devices

`lsblk` is a command-line utility in Linux used to list information about all available or the specified block devices. It shows details about block devices (like hard drives), their partitions, and other types of storage volumes.

`lsblk`: will list all block devices in a tree format showing the hierarchy of devices and their partitions.

- `-a` (`--all`): Show all devices, including empty ones.

- **-d** (**--nodeps**): Don't print device holders or slaves.
- **-f** (**--fs**): Output information about filesystems.
- **-o** (**--output**): Define which columns to output.
- **-l** (**--list**): Use list format instead of tree format.
- **-p** (**--paths**): Print full device paths.
- **-r** (**--raw**): Use the raw output format.
- **-t** (**--topology**): Show topology information.
- **-J** (**--json**): Use JSON output format.

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPPOINTS
sda	8:0	0	250G	0	disk	
└─vg_backups-lv_bamboo_backups	253:4	0	200G	0	lvm	/home/bamboo/shared/backups
sdb	8:16	0	32G	0	disk	
└─sdb1	8:17	0	1G	0	part	/boot
└─sdb2	8:18	0	31G	0	part	
└─almalinux-root	253:0	0	27.8G	0	lvm	/
└─almalinux-swap	253:1	0	3.2G	0	lvm	[SWAP]
sdc	8:32	0	512G	0	disk	
└─vgBambooMaster-lvAllureReports	253:2	0	80G	0	lvm	/opt/allure-reports
└─vgBambooMaster-lv_postgres	253:3	0	60G	0	lvm	/var/lib/pgsql
└─vgBambooMaster-lv_atlassian	253:5	0	60G	0	lvm	/opt/atlassian
└─vgBambooMaster-lv_test	253:6	0	1G	0	lvm	
└─vgBambooMaster-lvBambooHome	253:7	0	200G	0	lvm	/home/bamboo
sr0	11:0	1	1024M	0	rom	

sda	8:0	0	10.9T	0	disk	
└─sda1	8:1	0	1M	0	part	
└─sda2	8:2	0	512M	0	part	/boot
└─sda3	8:3	0	10.9T	0	part	/
└─sda4	8:4	0	1G	0	part	[SWAP]
sdb	8:16	0	10.9T	0	disk	
└─sdb1	8:17	0	5.5T	0	part	/minio/disk1
sdc	8:32	0	10.9T	0	disk	
└─sdc1	8:33	0	5.5T	0	part	/minio/disk2
sdd	8:48	0	10.9T	0	disk	
└─sdd1	8:49	0	5.5T	0	part	/minio/disk3
sde	8:64	0	10.9T	0	disk	
└─sde1	8:65	0	5.5T	0	part	/minio/disk4
sdf	8:80	0	10.9T	0	disk	
└─sdf1	8:81	0	5.5T	0	part	/minio/disk5
sdg	8:96	0	10.9T	0	disk	
└─sdg1	8:97	0	5.5T	0	part	/minio/disk6
sdh	8:112	0	10.9T	0	disk	
└─sdh1	8:113	0	5.5T	0	part	/minio/disk7

### 11.6.2 Fdisk - Managing Partitions

**fdisk** is a command-line utility in Linux used for disk partitioning. It's a powerful tool that allows users to create, delete, resize, and manage disk partitions on hard drives.

**fdisk** stands for "Fixed Disk" and is primarily used to manipulate partition tables on hard drives. It supports various partition table formats, with the most common being the Master Boot Record (MBR) and GUID Partition Table (GPT).

**fdisk -l**: lists the partition tables of all disk drives, showing the details of each disk and its partitions and volumes:

- Disk size
- Sector size
- Partition type
- Partition size

**fdisk -l /dev/sdx**: Show the partition table with details of a specific disk.

**Interactive mode: fdisk /dev/sda**

- **m**: display the help menu.
- **p**: print the partition table.
- **n**: add a new partition.
- **d**: delete an existing partition.
- **t**: change a partition type.
- **w**: write the changes into the disk and exit.
- **q**: quit without saving changes.

### Warning

**g** inside **fdisk** will erase any existing partition table and data on the disk. So it prepares the disk to be partitioned by creating a new empty GPT (GUID Partition Table) partition table.

Usage Example: Install a new partition on drive **/dev/sda** of 5.47TiB:

1. Become sudo: **sudo su**
2. List system disks information: **lsblk**
3. Start interactive disk **/dev/sda** modification: **fdisk /dev/sda**

```
Welcome to fdisk (util-linux 2.34).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): g
Created a new GPT disklabel (GUID: E8DFA8BA-599D-FC40-9B08-553DEA85E03D).

Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-23436722142, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-23436722142, default 23436722142): 11719933918

Created a new partition 1 of type 'Linux filesystem' and of size 5.5 TiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks
```

4. Write the changes: **w**

5. Check the partition has been created: `lsblk` or `fdisk -l /dev/sda`
6. Format the partition with a filesystem: `mkfs.ext4 /dev/sda1`
7. Mount the filesystem to a directory in order can be used: `mount /dev/sda1 /mnt/mydata`
8. Check the filesystems: `df -h`

### 11.6.3 LVM - Managing Volumes on Linux

#### 11.6.3.1 For PVs

Displays information about all physical volumes (PVs):

```
pvdisplay
```

Show detailed information about a physical volume:

```
pvdisplay [options] [PhysicalVolumePath]
```

Initialize a partition or drive as a physical volume, making it ready to be used in a volume group:

```
pvcreate /dev/sda
```

```
pvcreate /dev/sda
```

#### 11.6.3.2 For VGs

Displays information about volume groups. It shows detailed information such as:

- Volume group name
- Volume group Size
- Free space in the volume group
- Number of physical volumes (PVs) in the volume group
- Number of logical volumes (LVs) in the volume group

```
vgdisplay
```

Displays information about volume groups in a brief format:

```
vgs
```

Displays information about a volume groups:

```
vgdisplay <vg_name>
```

Create a new volume group by combining one or more physical volumes:

```
vgcreate <vg_name> <pv_name1> <pv_name2> ... <pv_namen>
```

Adds one or more physical volumes to an existing volume group:

```
vgextend <vg_name> <pv_name>
```

Removes one or more physical volumes from a volume group:

```
vgreduce <vg_name> <pv_name>
```

Removes a volume group from the system:

```
vgremove <vg_name>
```

### Warning

The volume group must be empty (all logical volumes must be removed first).

Renames an existing volume group:

```
vgrename old_volume_group new_volume_group
```

Scans all disks for volume groups and rebuilds the LVM cache:

```
vgscan
```

Checks the consistency of volume group metadata:

```
vgck <vg_name>
```

### 11.6.3.3 For LVs

Display information about logical volumes. It shows details such as:

- Logical volume name
- Volume group name
- Size of the logical volume
- Attributes and status of the logical volume

```
lvdisplay
```

Display information about a LV:

```
lvdisplay /dev/my_volume_group/my_logical_volume
```

Display information about logical volumes in a brief format:

```
lvs
```

Create a new logical volume within a specified volume group:

```
lvcreate -L 10G -n my_logical_volume my_volume_group
```

To extend the size of a lv:

```
lvextend -L +5G /dev/my_volume_group/my_logical_volume
```

To reduce the size of a lv:

```
lvreduce -L -5G /dev/my_volume_group/my_logical_volume
```

Resizes a logical volume, either increasing or decreasing its size. This combines lvextend and lvreduce:

```
lvresize -L +15G /dev/my_volume_group/my_logical_volume
```

```
lvresize -L -15G /dev/my_volume_group/my_logical_volume
```

To remove a logical volume:

```
lvremove /dev/my_volume_group/my_logical_volume
```

To rename a logical volume:

```
lvrename /dev/my_volume_group/my_logical_volume
```

Scan all logical volumes in the system and rebuilds the LVM cache:

```
lvscan
```

Change the attributes of a logical volume, such as making it active or inactive.:

```
lvchange -a y /dev/my_volume_group/my_logical_volume
```

#### 11.6.4 Filesystem

Report the amount of disk space used and available on filesystems:

```
df -h
```

- `-h`: output human-readable by showing sizes in KB, MB, GB, etc.

Estimates file space usage:

```
du -h
```



Create a filesystem on a storage device (e.g., a partition or a logical volume) specifying the type of filesystem:

```
mkfs -t ext4 /dev/sdX1
```

Attach a filesystem to the filesystem hierarchy at a specified mount point:

```
mount /dev/sdxi /path/to/my_mount_point
```

```
mount /dev/vgName/lvName /path/to/my_mount_point
```

Detach a filesystem from the filesystem hierarchy.

```
umount /path/to/my_mount_point
```

**/etc/fstab** file contains a list of entries that define how disk partitions, filesystems, and other block devices should be automatically mounted at boot time.

Example entry in **/etc/fstab** file:

```
/etc/fstab  
# <file system> <mount point> <type> <options> <dump> <pass>  
/dev/vgName/lvName /path/to/my_mount_point ext4 defaults 0 2
```

- **Dump**

- 0: The filesystem will not be backed up by the dump command.
- 1: The filesystem will be backed up by the dump command.

- **Pass:** which filesystems should be checked at boot time.

- 0: The filesystem will not be checked.
- 1: The filesystem will be checked first. This should be used for the root filesystem (/).
- 2: The filesystem will be checked after the filesystems with a pass value of 1. This is typically used for all other filesystems.

To mount all filesystems specified in **verb|/etc/fstab|** without rebooting:

```
mount -a
```

## 12 Storage Area Network (SAN)

### 12.1 Introduction

### 12.2 What is a SAN?

A **Storage Area Network (SAN)** is a high-speed, specialized network that provides block-level storage access to servers. It enables multiple servers to access storage devices (such as disk arrays, tape libraries, etc.) as if they were directly attached to the server. This architecture is primarily used in large-scale data centers where high availability, performance, and scalability of storage are critical.

A SAN is a robust, scalable, and high-performance solution for large enterprises needing centralized, reliable storage. By offering block-level access and integrating advanced features like redundancy, multi-pathing, zoning, and storage virtualization, SANs provide a solution capable of meeting the demands of modern data centers for mission-critical applications.

### 12.3 Key Components of SAN Architecture

SAN architecture comprises several critical components:

#### 12.3.1 Host Layer (Servers)

Servers, or hosts, that require access to storage. Each server has a Host Bus Adapter (HBA), which is a dedicated interface for SAN communication.

#### 12.3.2 Fabric Layer (Network)

The fabric is the communication pathway between servers and storage devices. It consists of SAN switches and cabling (fiber optics or Ethernet, in the case of iSCSI SANs). The SAN fabric includes:

- **SAN Switches:** Direct traffic between storage and servers.
- **Protocols:** Fibre Channel (FC) or iSCSI (Internet Small Computer Systems Interface).

#### 12.3.3 Storage Layer

This layer includes various storage devices, such as RAID arrays, tape libraries, and other block-level storage devices. Storage Controllers manage data between physical storage devices and the network.

### 12.4 Interconnect Layer

SANs are typically connected via **Fibre Channel** or **Ethernet** in the case of iSCSI. Fibre Channel SANs are faster and more reliable but expensive, while iSCSI is more cost-effective, utilizing existing IP networks.

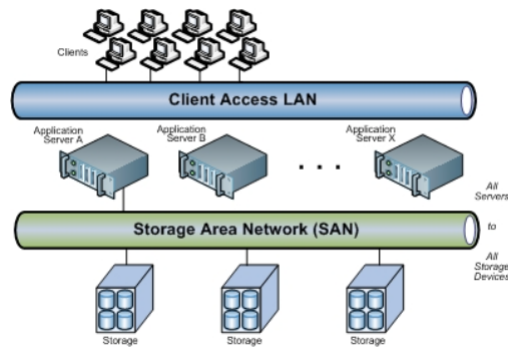
### 12.5 SAN Protocols

- **Fibre Channel (FC):** The most common protocol for SANs, known for low latency, high performance, and long-distance communication capabilities, with speeds up to 128 Gbps.
- **iSCSI:** Uses the existing IP network for data transfer, making it more affordable but slower than Fibre Channel.

- **Fibre Channel over Ethernet (FCoE):** Combines the benefits of FC and Ethernet by encapsulating Fibre Channel commands within Ethernet frames.
- **NVMe over Fabrics (NVMe-oF):** A newer protocol designed for faster, lower-latency access to SSD-based storage over networks.

## 12.6 SAN Architecture Topologies

Several SAN topologies are used depending on performance, availability, and cost requirements:



### 12.6.1 Point-to-Point

A direct connection between a server and a storage device. This topology is not scalable and is rarely used in modern implementations.

### 12.6.2 Fibre Channel Arbitrated Loop (FC-AL)

Devices are connected in a loop, with one device communicating at a time. While simpler, it is less scalable and has been largely phased out.

### 12.6.3 Switched Fabric

The most scalable and robust topology, where devices connect to switches in a fabric. This allows each device to communicate with others without collisions, ensuring high availability and performance. It is the most commonly used architecture in modern SANs.

## 12.7 How SAN Works

SANs provide **block-level access**, which differs from file-level access in that storage appears to the server as if it's a local disk. The key features include:

### 12.7.1 Block-Level Access

SANs offer block-level access, making storage appear as if it's directly attached to servers, ideal for performance-critical applications such as databases.

### 12.7.2 Logical Unit Number (LUN)

Storage devices are divided into smaller units called LUNs (Logical Unit Numbers), and each LUN is assigned to a server. Multiple LUNs can be pooled together for larger storage needs.

### 12.7.3 Data Paths

SANs use multiple paths for redundancy and load balancing, ensuring high availability. Multipathing allows continued access to storage even if one path fails.

## 12.8 Proper Use of SAN

To fully leverage a SAN's capabilities, certain best practices should be followed:

### 12.8.1 Storage Virtualization

Storage virtualization abstracts physical storage resources into pools that can be allocated to servers and applications, improving flexibility and utilization.

### 12.8.2 Multipathing for High Availability

SANs typically implement **Multipath I/O (MPIO)** to provide multiple redundant paths between servers and storage arrays, ensuring continuous access even in case of failures.

### 12.8.3 Zoning and LUN Masking

- **Zoning:** A security mechanism that ensures only certain servers have access to specific storage devices, improving security and traffic efficiency.
- **LUN Masking:** Restricts which servers can see which LUNs, preventing unauthorized access.

### 12.8.4 Data Replication and Backup

SANs should support data replication, both locally and remotely, to ensure backups and disaster recovery.

### 12.8.5 Snapshot and Cloning

Snapshots allow point-in-time copies of data for backup purposes, while cloning creates full copies of data for testing or development.

### 12.8.6 Storage Tiering

Storage tiering automatically moves data between high-performance and low-performance storage tiers based on access patterns, improving cost efficiency.

### 12.8.7 Quality of Service (QoS)

SANs allow setting QoS rules to ensure that high-priority applications always receive the necessary performance levels.

## 12.9 Benefits of SAN

- **High Availability and Redundancy:** SANs ensure no single points of failure, offering high uptime.
- **Scalability:** SANs can grow with your storage needs, providing virtually unlimited scalability.

- **Centralized Storage Management:** SANs allow centralized management of all storage resources, simplifying the allocation, monitoring, and optimization of storage.
- **Improved Performance:** Block-level access provides faster data access compared to traditional file-level systems like NAS.

## 12.10 Common SAN Use Cases

- **Enterprise Databases:** SANs are used to host databases due to their need for high performance and low latency.
- **Virtualization:** Hypervisors such as VMware or Hyper-V often use SANs to store virtual machines (VMs).
- **Backup and Disaster Recovery:** SANs are often used as targets for backups, snapshots, and replication for disaster recovery.
- **Big Data Applications:** SANs are ideal for large-scale data analytics due to their high performance.

## 13 Logs

### 13.1 Rsyslog

## 14 Others

### 14.1 Docker Desktop

- Docker desktop corre en nuestro Ubuntu desde Windows. Por tanto, si queremos que el Docker de nuestro Ubuntu tenga los certificados necesarios para conectarse a los recursos de nuestra empresa, debemos tener los certificados descargados, actualizados e instalados en nuestro sistema Windows nativo.
- Desde un terminal rellenamos el contenido del fichero de configuración de docker `~/.docker/config.json`:

```
~/.docker/config.json

{
  "auths": {
    "docker-registry.cloud.yourcompany.com": {}
  },
  "credsStore": "desktop.exe",
  "proxies": {
    "default": {
      "httpProxy": "http://urltoproxy",
      "httpsProxy": "http://urltoproxy/"
    }
  }
}
```

- En Docker Engine modificar el fichero que ahí introducimos por el siguiente:

```
{
  "builder": {
    "gc": {
      "defaultKeepStorage": "20GB",
      "enabled": true
    }
  },
  "experimental": false,
  "features": {
    "buildkit": false
  },
  "insecure-registries": [
    "docker-registry.cloud.yourcompany.com"
  ]
}
```

- Ejecutar el comando para logearnos contra el registry que hayamos escogido:

```
$ docker login -u USER -p PASSWORD docker-registry.cloud.yourcompany.com
```

- Probar el acceso al registry:

```
$ docker pull docker-registry.cloud.yourcompany.com/catalog/paas/j
```

## 14.2 Helm

### Compilar

Estando en el directorio que contiene el fichero Chart.yaml:

```
$ helm lint .
```

### Prueba de despliegue Resources en Cluster with file

```
$ helm install -f ./values_ssp.yaml . --debug \
--name-template standalone-11 --dry-run > output.yaml
```

### Prueba de despliegue Resources en Cluster with file

```
$ helm install -f ./values_ssp.yaml . --debug \
--name-template standalone-11
```

## 14.3 JSONPath

JSONPath is a query language for JSON, similar to XPath for XML. AlertSite API endpoint monitors let you use JSONPath in assertions to specify the JSON fields that need to be verified.

### 14.3.1 Basic Usage

A JSONPath expression specifies a path to an element (or a set of elements) in a JSON structure.

#### Dot Notation

```
$.store.book[0].title
```

#### Bracket Notation

```
$["store"]["book"][0]["title"]
```

#### Mix

```
["store"].book[0].title
```

#### Nota

Note that dots are only used before property names not in brackets.

#### Nota 2

The leading "\$" represents the root object or array and can be omitted. For example, *\$.foo.bar* and *foo.bar* are the same, and so are *\$/0.status* and *[0].status*.



### Nota 3

Using Bracket Notation be care to put **single quotes**, because double quotes are not accepted:

```
["name"]
```

### Nota 4

JSONPath expressions, including property names and values, are case-sensitive.

## Syntax Elements

- **\$**: The root object or array.
- **.property = ["property"]**: Selects the specified property in a parent object.
- **[n]**: Select the n-th element from an array. Indexes are 0-based.
- **[index1,index2,...]**: Selects array elements with the specified indexes. Returns a list.
- **..property**: Recursive descent: Searches for the specified property name recursively and returns an array of all values with this property name. Always returns a list, even if just one property is found.
- **\***: Wildcard selects all elements in an object or an array, regardless of their names or indexes. For example, address. "\*" means all properties of the address object, and book[\*] means all items of the book array.
- **[:n]**: Selects the first n elements of the array. Returns a list.
- **[-n]**: Selects the last n elements of the array. Returns a list.

### 14.3.2 Filters

We can use some expression to filter the output of the JSON in function of parameters introduced in the syntax.

## Syntax Elements

- **[?(expression)]**: Filter expression. Selects all elements in an object or array that match the specified filter. Returns a list.
- **@**: Used in filter expressions to refer to the current node being processed.

### 14.3.3 Example

#### *JSON Example*

```
{
  "store": {
    "book": [
      {
        "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      {
        "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      {
        "category": "fiction",
        "author": "J.R.R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  },
  "expensive": 10
}
```

#### Expressions

##### Nota

In all these examples, the leading \$. is optional and can be omitted.

```
$.store.* = ["store"][*]
```

- **Meaning:** All direct properties of store (not recursive).
- **Result:**

```
[
  "book": [
    {
      "category": "reference",
      "author": "Nigel Rees",
      "title": "Sayings of the Century",
      ...
    }
  ]
]
```

```

        },
        { ...
      }
    ],
    "bicycle": {
      "color": "red",
      ...
    }
  }
]

```

```
$.store.bicycle.color = ["store"]["bicycle"]["color"]
```

- **Meaning:** The color of the bicycle in the store.
- **Result:** red

```
$.store..price = \$.price = ..["price"]
```

- **Meaning:** The prices of all items in the store.
- **Result:** [8.95, 8.99, 22.99, 19.95]

```
$.store.book[*] = \$.book[*] = ..["book"][*]
```

- **Meaning:** All books in the store.
- **Result:**

```

[
  {
    "category":"reference",
    "author":"Nigel Rees",
    "title":"Sayings of the Century",
    "price":8.95
  },
  {
    "category":"fiction",
    "athor":"J.R.R. Tolkien"
    ...
  }
]

```

```
$.book[*].title = ..["book"][*]["title"]
```

- **Meaning:** The title of all books in the store.
- **Result:**

```

[
  Sayings of the Century,
  Moby Dick,
  The Lord of the Rings
]

```

```
$.book[0]
```

- **Meaning:** The first book.
- **Result:**

```
[
  {
    "category": "reference",
    "author": "Nigel Rees",
    "title": "Sayings of the Century",
    "price": 8.95
  }
]
```

```
$.book[0].title
```

- **Meaning:** The title of the first book.
- **Result:** Sayings of the Century

```
$.book[0,1].title
```

- **Meaning:** The titles of the first two books.
- **Result:** [Sayings of the Century, Moby Dick]

```
$.book[-1:].title
```

- **Meaning:** The title of the last book.
- **Result:** The Lord of the Rings

```
$.book[?(@.author=="J.R.R. Tolkien")].title
```

- **Meaning:** The titles of all books by J.R.R. Tolkien (exact match, case-sensitive).
- **Result:** It is a list because -n always returns a list. [The Lord of the rings]

```
$.book[?(@.isbn)]
```

- **Meaning:** All books that have the isbn property.
- **Result:** [Moby Dick, The Lord of the Rings]

```
$.book[?!@.isbn]
```

- **Meaning:** All books without the isbn property.
- **Result:** [Sayings of the Century]

```
$..book[?(@.price < 10)].title
```

- **Meaning:** All books cheaper than 10 titles.
- **Result:** [Sayings of the Century, Moby Dick]

```
$..book[?(@.price > \$.expensive)]
```

- **Meaning:** All expensive books.
- **Result:**

```
$ [
  {
    "category": "fiction",
    "author": "J.R.R. Tolkien",
    "title": "The Lord of the Rings",
    "isbn": "0-395-19395-8",
    "price": 22.99
  }
]
```

```
$..book[?(@.category == "fiction" ||
@.category == "reference")].title
```

- **Meaning:** All fiction and reference books titles.
- **Result:** [Sayings of the Century, Moby Dick, The Lord of The Rings]

```
$..book[?(@.category != "fiction")]
```

- **Meaning:** All not fiction books.
- **Result:**

```
[
  {
    "category": "reference",
    "author": "Nigel Rees",
    "title": "Sayings of the Century",
    "price": 8.95
  }
]
```

For more information about JSONPath, check the following [page](#)

### 14.3.4 JSONPath K8s

JSONPath template is composed of JSONPath expressions enclosed by curly braces . Kubectl uses JSONPath expressions to filter on specific fields in the JSON object and format the output. In addition to the original JSONPath template syntax, the following functions and syntax are valid:

- `range`, `end`: iterate list. `{range .items[*]}{.metadata.name}`
- `{"\n"}`, `{"\t"}`: to write salto de linea or tab. `{..metadata.name}{"\t"}{..matadata.image}`

#### Example 1

Get all **containers** / **initContainers** running and their images used inside a **Pod** or declared in pod.spec of **Deployment/Statefulset** . It is exactly the same changing little things. At least, it is showed in column view separated by tab.

```
$ kc get RESOURCE_TYPE RESOURCE_NAME \
-o jsonpath="{range ..containers[*]}{.name}{\"\t\"}{.image}{\"\n\"}" \
|sort|column -t
```

1. To choose between **Pod** or **Deploy/Sateful**: modify `RESOURCE_TYPE` & `RESOURCE_NAME`
2. To choose between **containers** or **initContainers**: modify the range.

- containers: `{range ..containers[*]}`
- initContainers: `{range ..initContainers[*]}`

```
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$ kc get po slb03a-s3javastandaloner-tst-88cd46994-kg8gq \
> -o jsonpath='{range ..containers[*]}{.name}{\"\t\"}{.image}{\"\n\"}' \
> |sort|column -t
cloudapphealth      docker-registry.cloud.caixabank.com/catalog/paas/java-runtime-health-standalone-1-8:1.0.0
java-standalone-1-8  docker-registry.cloud.caixabank.com/containers/slb03a/s3javastandaloner:latest
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$
```

Figure 8: Get all containers parsing a Pod JSON

```
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$ kc get deploy slb03a-s3javastandaloner-tst \
> -o jsonpath='{range ..initContainers[*]}{.name}{\"\t\"}{.image}{\"\n\"}' |sort|column -t
init-cacerts  docker-registry.cloud.caixabank.com/catalog/docker-init-setup:2.3.0
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$
```

Figure 9: Get all initContainers parsing a Deployment JSON

#### Example 2

Get all ports exposed by a Pod, Deployment or Statefulset.

```
$ kc get RESOURCE_TYPE RESOURCE_NAME \
-o jsonpath="{range ..containers[*]}{.name}{\"\t\"}{.ports}{\"\n\"}" \
|sort|column -t
```

```
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$ kc get po slb03a-s3javastandaloner-tst-88cd46994-kg8gq \
> -o jsonpath='{range ..containers[*]}{.name}{\"\t\"}{.ports}{\"\n\"}' \
> |sort|column -t
cloudapphealth      [{"containerPort":8180,"name":"health","protocol":"TCP"}]
java-standalone-1-8 [{"containerPort":8080,"name":"http","protocol":"TCP"}, {"containerPort":8090,"name":"metrics","protocol":"TCP"}]
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$
```

Figure 10: Get all ports exposed in a Pod parsing a Pod JSON

```

u01a1f97@DESKTOP-L5FIVQG:~/alexwork$ kc get deploy slb03a-s3javastandaloner-tst \
> -o jsonpath='{range ..containers[*]}.{.name}{"\t"}{.ports}{"\n"}}' \
> |sort|column -t
cloudapphealth [{"containerPort":8180,"name":"health","protocol":"TCP"}]
java-standalone-1-8 [{"containerPort":8080,"name":"http","protocol":"TCP"},{"containerPort":8090,"name":"metrics","protocol":"TCP"}]
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$

```

Figure 11: Get all ports exposed in a Pod parsing a Deployment JSON

### Example 3

Get all Volume Mounts of container called CONTAINER\_NAME inside a Pod, Deployment or Statefulset.

```

$ kc get RESOURCE_NAME -o jsonpath=\
"{range ..containers[?(@.name == "CONTAINER_NAME")]\
.volumeMounts[*]}.{.name}{"\t"}{.mountPath}{"\n"}} \
|sort|column -t

```

```

u01a1f97@DESKTOP-L5FIVQG:~/alexwork$ kc get deploy slb03a-s3javastandaloner-tst \
> -o jsonpath='{range ..containers[?(@.name == "java-standalone-1-8")].volumeMounts[*]}.{.name}{"\t"}{.mountPath}{"\n"}}' \
> |sort|column -t
.
init-cacerts /opt/ca-certs
pvc-mounted-heapdump-cxb-slb03a-s3javastandaloner-tst /storage
volume-from-configmap-testproperty /tmp/properties/testproperty
volume-from-configmap-tsttest /tmp/properties/tsttest
volume-from-secret-certbueno /tmp/secretos/certbueno
volume-from-secret-certmalo /tmp/secretos/certmalo
u01a1f97@DESKTOP-L5FIVQG:~/alexwork$

```

Figure 12: Get all volumeMounts in a container parsing a Deployment JSON

## 14.4 Ubuntu vs RH7

Ubuntu es una distribución de Linux que se basa en un kernel Debian, lo que significa que no funciona exactamente igual que otras distribuciones con otros kernel. RH7 al igual que Centos (Open Source) trabajan con un kernel diferente y por tanto los comandos son diferentes. Por ejemplo el gestor de paquetes de Debian es apt y el de RH7 es yum.

## 14.5 Vagrant

Vagrant is a tool for building and managing virtual machine environments in a single workflow.

- **vagrant global-status:** nos muestra el estado de todas las maquinas vagrant que tenemos corriendo. Ojo! **This data is cached and may not be completely up-to-date (use "vagrant global-status -prune" to prune invalid entries).**
- **vagrant global-status -prune:** same as before, but up-to-date.
- **vagrant status vagrant\_id or vagrant\_name:** shows the status of the vagrant with "vagrant\_id" or "vagrant\_name" (vagrant\_name == cloumlabxxx)
- **vagrant up vagrant\_name:** initialize a new vagrant with the id "vagrant\_id", it can take a lot of time (15 minutes).
- **vagrant halt vagrant\_id or vagrant\_name:** stops the vagrant
- **vagrant destroy -f vagrant\_id or vagrant\_name:** destroy the vagrant with the id "vagrant\_id", you lose all your configurations, but not your data into /vagrant.
- **ps -ef | grep cloumlabxxx:** show the processes running in vagrant.
- **kill -9 process\_id:** kill the process with "process\_id"

## 14.6 Red Hat Package Installer

### 14.6.1 RPM

**RPM (Red Hat Package Manager)** is a free open and **powerful package management system**. The name refers to file format **.rpm** and the package manager program itself. An RPM package can contain an arbitrary set of files. Most RPM files are “binary RPMs” (or BRPMs) containing the compiled version of some software. There are also “source RPMs” (or SRPMs) containing the source code used to build a binary package.

It is capable of:

- **Building computer software from source** into easily distributable packages.
- **Installing, updating and uninstalling** packaged software.
- **Querying detailed information** about the packaged software, whether installed or not.
- **Verifying integrity** of packaged software and resulting software installation.

### 14.6.2 YUM

## 14.7 Timeshift to Backup and Restore Your Linux System

### 14.7.1 Introduction

Linux is susceptible to system failures caused by incorrect commands or system operations. So if you use Linux on your main computer, you may frequently encounter problems. Fortunately, there are system restoration tools that create snapshots of your files and settings, which you can restore on your system to put it back to its previous functioning point in case any of your operations renders it unusable.

Timeshift works by creating a snapshot of your system using either `rsync` or `btrfs` mode, depending on your Linux distro. To do this, what Timeshift essentially does is create a restore point for your system at a time when everything's running smoothly. This backup includes all the system files and settings—and no user files or documents. That way, when you accidentally mess up something on your system while configuring or customizing it, you can restore it back to this restore point and revert all your changes.

### 14.7.2 Timeshift Installation

The first step is install timeshift, depending on your system you will use different commands. For Ubuntu:

```
$ sudo apt install timeshift
```

### 14.7.3 Create a Snapshot

```
$ sudo timeshift --create --comments "A new backup" --tags D
```