Linux administration

PID_00290379

Remo Suppi Boldrito
Josep Jorba Esteve

Recommended minimum reading time: 11 hours

Universitat Oberta de Catalunya

Remo Suppi Boldrito

Telecommunications Engineer. Ph.D in Computer Science from the Autonomous University of Barcelona (UAB). Lecturer in the Department of Computer Architecture and Operating Systems at the UAB.

Josep Jorba Esteve

Senior Engineer in Computer Science. Ph.D. in Computer Science Engineering from the UAB. Lecturer in Computer Science, Multimedia and Telecommunications at the UOC, Barcelona.

The assignment and creation of this UOC Learning Resource have been coordinated by the lecturer: Josep Jorba Esteve

First edition: September 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Authorship: Remo Suppi Boldrito, Josep Jorba Esteve
Production: FUOC

Contents

Introduction

One of the first tasks the administrator will have to deal with will be the management of the local resources present in the system to be managed. This module will cover some of these basic administration tasks, concerning network and disks, and security and services.

This will be done by getting an overview of the current state of the system using the different procedures and commands that the administrator can use to evaluate the different parts of the system. This will allow administrative decisions to be made if performance failures, gaps or lack of resources are detected.

As with any operating system, one of the main points of management is the administration of these resources (CPU, memory, processes, disks, users/groups, services, software, etc.), since in any role that the machine acts, their management will be necessary. These topics will be covered in detail in each section.

An important part will also be devoted to network administration, as it is vitally important in a current system and an essential element in its operation. Within this section, the most important network services for information needs (DNS, DHCP, NIS), resource sharing (NFS), and secure connection systems (SSH, VPN) will be analyzed.

Finally, the main issues related to security will be analyzed and aspects of both local and network security will be seen, and it will end with the deployment of services (web and proxies) to analyze the potential of the GNU/Linux systems in these aspects.

Objectives

The main objectives of this module are:

1. To learn the basics of the GNU-Linux operating system management.
2. To analyze how the operating system manages and administers the resources of disks, devices, networks, security, and services.
3. To analyze different scenarios and use cases of common services in operating systems.

Most important concepts:

The students should focus their attention on the following fundamental concepts presented in this module:

- Package management tools
- Generic management tools
- Booting levels and services
- Processes and Memory
- Disks and File System
- Storage redundancy and logical volumes (RAID and LVM)
- Network administration
- Subnets and routing
- Network services configuration (NAT, DHCP, DNS, NFS, Bridging, NIS, SSH, VPN)
- Local security
- Firewalls: iptables and nftables
- Servers: Web and Proxies.

1. Basic tools for the administrator

The GNU/Linux system administrator has to deal with a large number of tasks on a daily basis. In general, in the *Nix (Unix/Linux) philosophy, there is not usually a single tool for each task or a single way of doing things, and it is common for *Nix systems to provide numerous more or less simple tools to deal with the different tasks.

In this section we will see different groups of tools, identify some of their basic functions and see some examples of their uses. As mentioned in previous modules, there are some standards in the GNU/Linux world, which allow defining certain basic common characteristics in any GNU/Linux distribution. These standards, such as the LSB (or Linux Standard Base) and the FHS (Filesystem Hierarchy Standard), define different tools or structures of information, as well as a set of rules that have to be met for a distribution to be considered a GNU/Linux system and that the administrator must keep in mind.

Also as mentioned in the GNU/Linux introductory module, an administrator will automate administration tasks through commands grouped into shell scripts (text files with scripts, control and variables), which will be interpreted by the shell (command interpreter) of the system. Programming these shell scripts, which can be of considerable complexity, allows simple commands to be joined with flow control structures, variables, functions, and other elements to generate new commands but adapted to the tasks a particular administrator wants to perform. This allows for a rapid prototype environment of new tools for task automation, and we encourage the reader to read specialized literature in shell script programming, such as [Qui] [Mik] [Coo]. If this is not sufficient for the actions the administrator wants to perform, high-level language compiling and debugging environments (such as C) can be used. It is unusual, as the large number of commands and shell script programming gives considerable capacity for the automation of management tasks, but these environments add additional capacity to generate new applications or tools, or to incorporate applications into the system that are of open source code.

Below we will analyze a set of tools that are essential for keeping the system up to date, which are the package management tools. The software provided by each GNU/Linux distribution, or subsequently incorporated, is organized into units called packages, which include the files of a given software, plus the steps necessary for installation preparation (dependencies), subsequent configuration, or, if applicable, the upgrade or uninstallation of a given package. Each distribution has a different strategy (depending on the branch from which they are derived) and usually consists of a management software

to maintain the lists of packages installed/or to be installed, as well as the control of existing versions, or various possibilities of updating through different sources (package repositories).

1.1. Package management tools

In any distribution, packages are the basic element for new software installation, upgrade, or removal of unused software.

> Basically, a package is a set of files that form an application or a combination of several related applications, forming a single file (called package), in its own compressed format, which is the one that is distributed, nowadays, through web services from public repositories.

Using packages makes it easy to add or remove software as it works as a unit and the individual files that make it up.

Sometimes, some applications or components are distributed in more than one package, separating essential components from optional or development-related components, or modular components. Packages with different nomenclatures can be found, such as, for example, package as the main, package-common, to offer the associated common files (e.g., if a client or server is installed); package-devel and/or -libs, with the files associated with development and other names for additional files to the application/ system component. For example, if in the Debian packages we search for NFS (Network File System), packages such as *nfs-common* (common files to the client and server), *nfs-ganesha* (NFS server in user space), *nfs-kernel-server* (NFS server), *nfs-modules* (support for NFS filesystem), *nfs4-acl-tools* (complementary tools for ACL) can be seen, among others.

In the content of the distribution (ISO images), the packages are usually grouped by categories such as:

a) Base: essential packages for system operation (utilities, booting programs, system libraries).

b) System: administration tools, utility commands.

c) Development: programming aids such as editors, compilers, debuggers...

d) Graphics: graphical controllers and interfaces, desktops, window managers.

e) Other categories.

These should not be confused with the groupings that are made of the repositories since these are usually more specific, such as Debian, which establishes the following categories (only some of them): Administration Utilities, Databases, Development, Documentation, Editors, Education, Gnome, GNU R, Graphics, Networks, Web Servers, JavaScript, Kernels, etc.

For the installation of a package, a series of steps will be required:

1) Pre-installation: it must be verified that there is the necessary software (and with the correct versions) for its operation (dependencies), whether they are system libraries or other applications that will be used by it.

2) Installation: unpacking the content and copying the files to their final locations, either absolute (they will have a fixed position) or, if allowed, relocated to other directories.

3) Post-installation: adaptation of the necessary files and configuration of the possible software parameters for their correct operation.

Depending on the types of packages, these steps are usually mostly automatic (this is the case for RPM and DEB). A set of steps may also need to be done manually (such as for TGZ packages), depending on the tools the distribution provides.

Next, two of the most classic packages will be discussed; each distribution typically has one as standard but generally includes utilities to work with the other common formats. A description will also be made of a series of proposals for "universal" packaging systems that can be used generically by more than one distribution.

1.2. Red Hat (or derived distributions): RPM packages

RPM packages, by convention, usually use a name such as: `package-version-rev.arch.rpm` where the *package* is the software name, version is the software *version* numbering, *rev is* usually the revision of the RPM and *arch* package, the architecture for which the package is intended, either Intel/AMD/ARM (for example x86_64, aarch64,...). We can also find *arch=noarch* that is typically used when it is architecture independent, for example scripts, source code or text, data tables, etc. The usual execution includes the execution of the `rpm` command, with the options of the operation to be performed and one or more package names to be processed. Typical operations with RPM packages include:

- **Dependency management**: RPM packages incorporate the idea of database and dependency management of existing packages.

- **Package information**: with the option *-q* accompanied by the package name or with *-p* if done on an rpm file. If the package has not been installed yet, in addition to the option -q, the information to be obtained must be indicated, and if we want to ask all the packages installed at the same time, the option would be *-qa*. The following table summarizes the main options:

Table 1

Consultation	RPM options	Results
Files	*rpm -ql*	List of files contained in the package (passed as parameter)
Information	*rpm -qi*	Package description
Requirements	*rpm -qR*	Prerequisites, libraries or software

- **Installation**: it is done with `rpm -i package.rpm`, or it can be done with the URL where the package is found, to download it from web servers (or ftp). We simply need to use the ftp:// or http:// syntax to indicate the location of the package. The installation can be done as long as the package dependencies are being met, which can be either previous software or libraries that should be installed. In the case of non-compliance, it will list which software is missing, and the name of the package that provides it. The installation can be forced (at the risk of it not working) with the -- *force* or --*nodeps* options, or the dependency information can simply be ignored.

Example of installing a remote package

```
rpm -i http://remote_site/directory/package.rpm
```

will allow us to download the package from the URL/directory website (or ftp) and proceed with the package installation.

- **Update**: `rpm -U package.rpm` (equivalent to the installation), but first checking that the software already exists. It will take care of deleting the previous installation.

- **Verification**: during normal system operation, many of the installed files may change, which is why RPM allows verifying the files for changes. These can occur due to normal process or due to an error that could indicate corrupt data and/or affect its operation. We can check the package by using `rpm -V package`, or with `rpm -Va`, they will all be checked.

- **Deletion**: erase the package from the RPM system (*-e* or *--erase*). If there are dependencies, they may need to be removed first.

Care should be taken about the source of the packages, and only use known and reliable package sources, either from the distribution developer itself, or from trusted and community-recognized sites. A digital "signature" of these is usually offered with the packages so that they can be verified for authenticity; Hash md5, sha128/256/512 functions are often used to verify that the package has not been altered, and other systems, such as GPG (Gnu version of PGP), to verify the authenticity of the package developer. There are also different generic RPM package repositories on the Internet, where they are available for different distributions that use or allow the RPM format or RPM package browsers, such as https://www.rpmfind.net/.

For secure package use, repositories digitally sign packages (e.g., with the above-mentioned GPG). This allows ensuring (if signatures are available) that the packages come from the reliable source since each supplier (repository) includes PGP signature files with the key for their site. Normally for the official repositories of the distribution they are already installed, but there are no third-party signatures that must be installed with a file downloaded from the developer. To do this, we must execute: `rpm --import GPG-KEY-FILE`

Where *GPP-KEY-FILE* is the GPG key file or the file URL that will also be accompanied by a hash (usually md5) to check its integrity. To know the existing signatures in the system, `rpm -qa | grep ^gpg-pubkey` can be executed.

For a specific rpm package, it can be checked whether it has a signature and which one has been used by running `rpm –checksig -y package.rpm` and to verify that a package is correct based on the available signatures, it can be checked with: `rpm -K <package.rpm>`

Obviously, only trusted site signatures should be imported to control risks because when RPM finds packages with signatures that cannot be verified, or the package is not signed, it will notify, and the action to be taken will depend on the actions indicated by the administrator.

RPM is the default Red Hat format (and derived distributions), so it has full support and updates. In Debian (and derived distributions) the DEB format is used (see below), but the `rpm` command also exists although the `alien` command is recommended to convert a .rpm packet to .deb and then install it with the Debian package manager (`dpkg`).

In addition to the distribution packaging base system, each distribution includes a high-level software management system, which adds a superior layer to the base system and facilitates software management tasks with a number of utilities to control the process more amicably.

In the case of Red Hat (and derived distributions) it is common to use the `yum` command, which allows the installation and management of packages in rpm systems and the automatic management of dependencies between packages, facilitating access to multiple different repositories indicated in the */etc/yum.conf* file, or in different files with *.repo* extension in */etc/yum.repos.d*. RPM-based systems include a new package management system called `dnf` (meaning *Dandified YUM*) that is considered the `yum` substitute and default manager for some distributions such as Fedora 22+, CentOS8+, and RHEL8.

The yum configuration is based on:

```
/etc/yum.config (options file)
/etc/yum (directory for some associated utilities)
/etc/yum.repos.d (repository specification directory, one file for each, includes access and
location information for gpg)
```

A summary of typical yum operations is:

Table 2

Order	Description
yum install <name>	Install package with name.
yum update <name>	Update an existing package.
yum remove <name>	Delete package.
yum list <name>	Search package by name (name only).
yum search <name>	Search more broadly.
yum provices <file>	Search for packages that provide the file.
yum update	Update the entire system.
yum upgrade	Same as above including additional packages.

Regarding the advantages of `dnf` in relation to `yum`, we can mention:

- Better performance (speed, < memory usage, efficiency).
- Improvements in the resolution of dependencies and resolution of potential dependency corruption problems in some cases.
- Documented APIs (which did not happen with YUM), and independence from Python versions.

All of this has meant a change and inclusion of independent libraries to resolve dependencies (*libsolv* and *hawkey*), metadata and repository management (*librepo*), and package group management (*libcomps*) by providing well-defined APIs, based on a number of external libraries, which can be improved

independently. In order not to cause problems for administrators, the command parameters have been kept practically compatible with YUM (we only have to change the `yum` command to `dnf`) so an individual package installation will be:

```
$ dnf install package-name
```

The configuration of the DNF system is done through the */etc/dnf/dnf.conf* file and all the repos description files (**.repo*) found at */etc/yum.repos.d*. On the other hand, cached system files can be found at */var/cache/dnf*.

Note

CLI differences between DNF and YUM can be found at: http://dnf.readthedocs.io/en/latest/ cli_vs_yum.html.

1.3. Debian: DEB packages

Debian incorporates interactive tools (in text mode) such as `tasksel`, which allow choosing large sets of packages grouped by the type of task: packages for X, for development, for documentation, etc., or as `dselect` which makes it easy to navigate through the entire list of available packages (an extensive list) and choose those that we want to install or uninstall. In fact, these are only APT mid-level software manager front-ends (equivalent to YUM in RPM-based distributions).

At the basic level of package management (rpm equivalent) Debian uses the `dpkg` command to manage DEB packages with parameters such as -i to install a package (`dpkg -i package.deb`). Although considered a low-level command, it allows all types of tasks to be performed, such as obtaining information, installing, deleting, or making internal changes to software packages.

At a higher level of abstraction (such as `yum` in distributions using RPM) is the APT command series (such as `apt` -general- or `apt-get, apt-cache, apt-file`,...). APT allows managing packages through a list of current and available packages from multiple software sources, either from the facility's own ISO images or from websites (the most common form). This management is done transparently, so that the system is independent from the software sources. The APT system is configured from the files available in */etc/apt*, where */etc/apt/sources.list* is the list of available sources. It could be, for example, for the Bullseye version of Debian:

```
deb https://deb.debian.org/debian/ bullseye main contrib non-free
deb-src https://deb.debian.org/debian/ bullseye main contrib non-free
```

Where "official" sources are indicated for Debian specifying the type of source (web in this case), the site, the version of the distribution and categories of the software to be searched (free, or contributions from third parties or non-free or commercial licence). Software packages are available for different versions of the Debian distribution and are identified with stable, testing, and unstable. The use of one or the other determines the type of distribution (after changing

repository sources on *sources.list*). Mixed package sources may be available, but are not highly recommended as conflicts may occur between the versions of the different distributions.

Once software sources are configured, the main command to manage them is `apt-get` (although some administrators prefer `apt` that does not provide as much information as the first one) and allows installing, updating, or deleting from an individual package, until the entire distribution is updated.

There is also a front-end to `apt-get`, called `aptitude`, with a similar interface but, according to experts' opinions, it has better package dependency management, and more efficient algorithms for resolving package conflicts, so it is advisable for complex package conflict problems.

Some basic `apt-get` functions are:

1) Installing and deleting a package:

```
apt-get install package
apt-get remove package
```

2) Updating the list of available packages:

```
apt-get update
```

3) To update the distribution, we could update the list of packages and then upgrade it:

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
```

The distribution upgrade can also be done by updating the installed packages and verifying the dependencies with the new ones, although some administrators are reluctant to do so because they can delete some packages and install new ones or larger versions, which could generate some problems.

The system update generates a download of a large number of packages which makes it advisable to empty the cache, the local repository, of the downloaded packages (they are kept in */var/cache/apt/archive*) with the `apt-get clean` command (all) or `apt-get autoclean` (those not necessary because there are already new versions) or with `apt-get autoremove` (to remove packages/libraries that were necessary for another package and that has been uninstalled).

The APT system also allows a secure mode (*SecureAPT*) based on a hash (md5) and packet signature (by GPG). If signatures are not available during the download, `apt-get` reports it and generates a list with the unsigned packages, leaving the decision to the administrator whether to install them anyway. For a list of available trusted sources, we can run:

```
# apt-key list
```

The *gpg* keys of the official Debian sites are distributed in a package and to install them:

```
apt-get install debian-archive-keyring
```

Obviously considering that we have *sources.list* with the official sites. The default distribution keys are already installed and only those from other sites should be installed if considered reliable by running `apt-key add key_file` or also with:

```
# gpg --import file.key
# gpg --export --armor XXXXXXXX | apt-key add -
```

Where XXXXXX is a key-related hexadecimal number (see repository instructions for the recommended way to import key and required data).

Another important functionality of the APT system is querying package information through the `apt-cache` command, which allows us to interact with Debian software package lists.

Example: to search for package information

Search for packages based on an incomplete name:

```
apt-cache search name
```

Show the package description:

```
apt-cache show package
```

Which packages it depends on:

```
apt-cache depends package
```

Another useful and interesting `apt` command is `apt-show-versions`, which will show which packages can be updated.

Other more specific tasks can only be performed by `dpkg`, for example, getting the file list of a given package already installed:

```
dpkg -L package
```

Or the entire package list with:

```
dpkg -l
```

Or look up which package an element comes from (a file, for example):

```
dpkg -S file
```

This particular one works for installed packages; `apt-file` also allows the same but for packages that have not been installed yet. Finally, some graphical tools for APT, such as `synaptic` (or in text mode, such as those already mentioned: `aptitude` or `dselect`), should also be mentioned.

In conclusion, it should be noted that the APT management system (in combination with the `dpkg` base) is very flexible and powerful in managing updates, and it is the package management system used in Debian and its derived distributions (Ubuntu, Knoppix, Mint, etc.).

1.4. New packaging formats

To solve some problems with these packaging systems (basically dependencies), new alternatives have emerged such as Snap (or Snappy if the packaging system is referenced), Flatpak or AppImage.

Typical problems with classic packaging systems include:

- The packages are overly dependent on the distribution. Despite being a specific standard (such as DEB or RPM) they may depend on specific packages of the particular distribution, or there may be library dependencies or versions thereof that may not be found in the target distribution.

- This creates a “fragmentation” of the GNU/Linux packages, making it nearly impossible to package applications validly for different GNU/Linux distributions.

- In a single distribution, the package may require versions of libraries prior to or later than the existing ones, creating dependencies that are difficult (or impossible in some cases) to resolve.

- “Universal” packages, valid for any distribution, are not possible.

- In most cases, root privileges are required to install the packages (via `sudo` or `su`).

New systems such as Snap (from Canonical, developer of Ubuntu), Flatpak (also known as `xdg-app`, and with the participation of developers of the Gnome and Red Hat community) or AppImage (as the maximum exponent for packaging portable applications in user space - without being root-) have

different features that try to solve the aforementioned issues with some particularities: Snap is used in both server packages and desktop applications, while Flatpak (and its relationship with Gnome, although it may be used in other window environments) is intended for running applications in user sessions. In the case of Snap, there is the concept of centralized repository (Snap Store or Ubuntu myApps, or other names, depending on the destination platform) controlled by Canonical, while in Flatpak there is no such concept, only multiple repositories to add for having sources of different packages. AppImage is intended so that a user can download the developer app and test/use it without having to worry about anything else and so that the developer can make an app without worrying about multiple distributions.

As evidence, Snap (due to Canonical/Ubuntu support) and Flatpak (due to its importance in RHEL/Gnome) will be analyzed.

1.5. Snap

It has been active as from Ubuntu 16.04 LTS and later versions and is used to a lesser extent in other distributions, with the most downloaded snaps by distribution being the following: Arch Linux/spotify, CentOS/wekan, Debian/spotify, Fedora/spotify, Manjar/vlc.

For its use in the distribution, the *snapd* package (or similar name, depending on the distribution) must be installed, which includes the `snap` command for package management. If we want to create packages, we will also need the *snapcraft* package (and on Ubuntu `multipass`, required for creating isolated environments)

The main parameters are summarized below:

Table 3

Order	Description
snap help	For information on available options.
snap find [*package*]	To search for available packages, either by name or by part of it. It can be combined with grep to find other words associated with the package in its comments.
snap install *package*	To install a package.
snap refresh *package*	To update a previously installed package, if a new version is available, it will be updated.
snap list [--all]	To list packages installed on the system. It can be combined with grep to find a certain package/word of the installed packages. [--all] to list the disabled ones as well.
snap remove *package*	To uninstall a package.
snap changes	To list recent changes to the *snap system*.

Packages are in the */snap* directory, and depending on the type, if they are graphically executed, they are added to the system menus, or if they are CLI executable, they will be called from the shell with names similar to the package name. We can always browse the directories:

- */snap/bin*, to find the binaries available to run.

- */snap/package_name/current/bin*, to find the binaries/executables associated with a package, in its most current version. Although the application launcher is typically found on */snap/bin*, it is recommended that the application be run via the command available in the first directory, or by accessing the system menus where the application is installed.

Each snap is a read-only image of the *squashfs* file system and to access the files within these images, `snapd` mounts the images, one for each installed version of snap, within */snap*. This list of loop devices includes the snaps that have been installed and they are part of the normal operation of `snapd`, so no attempt should be made to delete them. If we want to delete them, we simply will execute `snap remove package` and the app, and the corresponding loop device will be deleted. An important maintenance that can be done to recover disk space is to remove the snaps that are disabled since there is a new version; they are listed with `snap list --all` and deleted with `sudo snap remove --revision=revision_ID name_snap`.

It is necessary to note that in some cases there are problems to resolve with security models, for example, especially for graphical applications in X Window where an application, for example, can use X Window to send false keyboard events and cause an application to interact in an unwanted way. These issues are expected to be resolved by replacing X Window Server with alternatives such as Wayland or Mir.

1.6. Flatpak

Flatpak, also known as `xdg-app`, is a utility for universal package installation and management that provides a process-isolated environment (called Bubblewrap), where users can run applications isolated from the rest of the operating system. This allows applications independent of the Gnu/Linux release or distribution to be installed and packages can be updated without touching the operating system on which they are running.

This package is officially distributed over Fedora 24 and is also included in other distributions such as Centos, Mint, Ubuntu Mate or others that can be installed such as Debian, OpenSuSE, etc. (if not available it can be installed in RHEL and derivatives with `dnf install flatpak`).

The `flatpak command` is used to perform the following tasks:

1) Install the required package repositories. It includes importing GPG keys to identify the repository and its packages, and importing the specific repositories we want to access.

2) Install from the repository the necessary runtime that will provide all the required dependencies for the applications in the repository.

3) This allows to see with different options, which packages are available, install them and run the installed applications.

This summary box shows some of the commands that can be used:

Table 4

Order	Description
flatpak remote-add --gpg-import=keygpg repo urlrepo	Add a repo repository identified by a keygpg (previously downloaded), and located at URL urlrepo.
flatpak remote-list	List available remote repositories.
flatpak install repo runtime version	Install runtime from the repository with the specific version number version.
flatpak remote-ls repo --app	List the applications available in the repo.
flatpak install repo stable application	Install the application in its stable version from the repo repository.
flatpak list	List available applications.
flatpak run name	Run an available (name) application.
flatpak update	Update all installed applications from repository.

As discussed in Snap, Flatpak also has a number of issues pending with security models (especially for X Window graphics applications). This part is expected to be fixed as the X Window Server is replaced by alternatives such as Wayland or Mir (for Ubuntu and Snap).

It is also interesting to discuss the possibilities of AppImage, although it is more designed for non-administrative users, or OrbitalApps, designed to develop portable applications, as the ones that could be carried on USB between various systems.

1.7. Generic management tools

In certain management environments, some generically designed tools can be considered to manage various aspects of the machine. It should be noted that for these tools it is difficult to stay up to date, given the current distribution version plans that mostly have very rapid evolutions. Examples of these tools include:

1) Webmin: it is a long-standing tool that has gone through different states but currently has an active development (2022) although it is not included in some repositories. It is a web tool with a series of plugins that can be added for each service to be managed with forms, where the configuration parameters of the services are specified allowing the possible remote administration of a server.

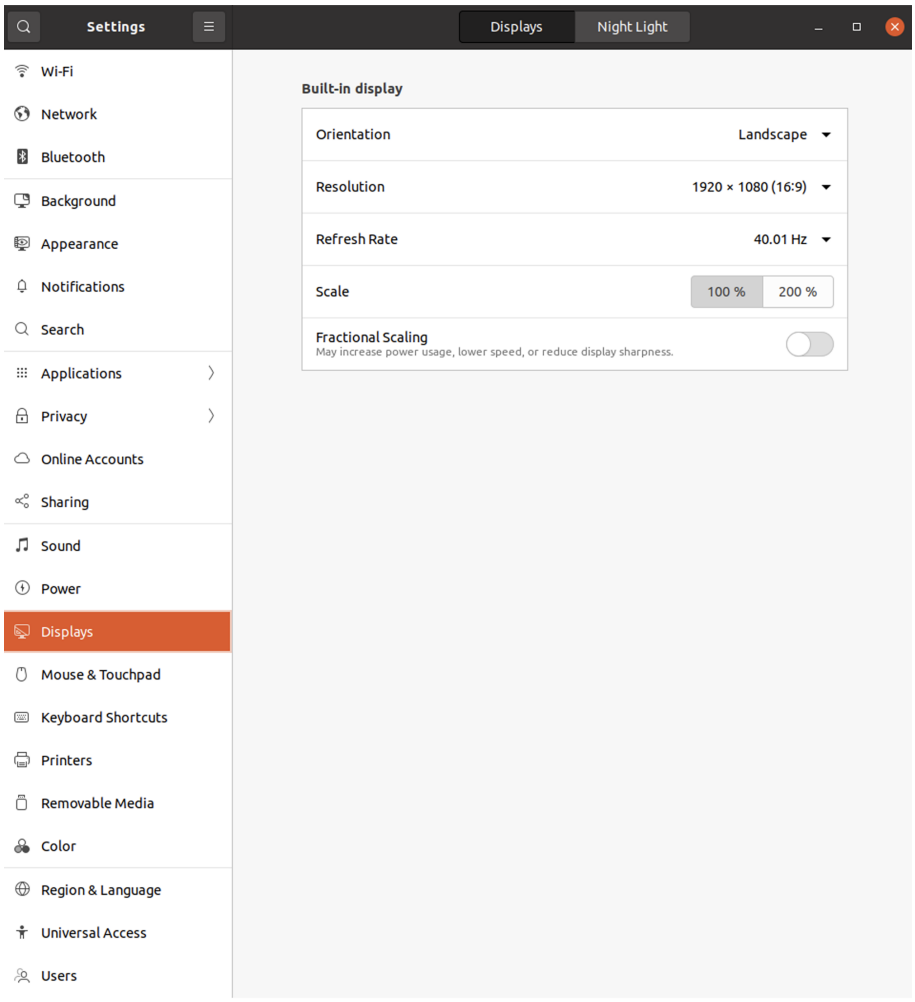
2) Cockpit: a management environment focused on providing a modern, easy-to-use interface for managing and administering servers that is available for the most widely used distributions and in official repositories. It reminds administrators with a certain experience of Webmin, but with better features, API and security model. The most important features of Cockpit are: high quality graphical environment/user interface, modular design expandable by additional modules (including developing proprietary modules), multiple servers from a single dashboard, non-intrusive (it works in conjunction with other management tools without any problems), it uses a systemd socket and does not use memory when not in use. It is based on the existing functionality (it does not require a default setting), does not store server status or data anywhere, does not have special privileges and does not run as root creating a session as a user and has the same permissions as that user, so to perform administrative tasks, a user needs permission to use `sudo` or PolicyKit to escalate privileges.

Some distributions, such as OpenSuSE, have their own tools such as YaST, or in the Gnome and KDE desktop environments, they usually have the concept of "Control Panel", which allow managing both the visual aspect of the graphical interfaces and treating some parameters of the system devices (there are also proprietary software examples, such as cPanel or Plesk, among others, or linked to Cloud providers, such as SPanel).

As for individual graphical management tools, the GNU/Linux distribution itself offers some directly (tools that accompany both Gnome and KDE), tools dedicated to managing a device (printers, sound, network card, etc.), and others for the execution of specific tasks (Internet connection, configuring system service boot, configuring X Window, viewing logs, etc.). Many of them are simple frontends to basic system tools, or are tailored to the particularities of the distribution.

It should be noted, in this section, that the Red Hat distribution and its derivatives have different utilities for different administration functions (see the administration menu), or commands such as `system-config-xxxxx` for different functionalities but which have progressively been replaced by the concept of a desktop environment control panel (whether the Gnome or KDE configuration, for example) with functionalities similar to these utilities. The following figure shows this approximation in the Gnome control panel on Ubuntu 20.04LTS control panel.

Figure 1



1.8. Other tools

In the limited space of this unit, not all those tools that can provide benefits for administration can be discussed, but some of the tools that can be considered basic will be cited:

- **The multiple basic UNIX utility commands**: grep, awk, sed, find, diff, gzip, bzip2, cut, sort, df, du, cat, more, file, which, ip, ss, dhclient, ...

- **The editors**: required for any configuration task. In text mode, the choice is the `vi` editor (although there are alternatives, such as `joe`, `nano`) that is found in all distributions through an improvement called `vim` (*VI iMproved)*. This editor allows coloured syntax in various languages, fast movement and effectiveness but its detractors say that its work modes are unfriendly and the keyboard shortcuts are not the usual ones; however, once we learn to work with it, it is highly efficient and recommendable. And in graphical mode, *Sublime*, *Atom*, or *VisualStudioCode* are recommended, although they are professional editors

more adapted to programmers. However, Sublime, for example, is simple and easy to install and manage.

- **Script languages**: Bash (for any simple - or not simple - administration task, Perl (very suitable for treatment of regular expressions and analysis of files such as filtering, sorting, etc.); PHP (very widely used language in web environments); Python, for rapid app prototyping, ...

- **High-level language compiling and debugging tools**: *GNU gcc* (C and C++ compiler among others), *gdb* (debugger), *xxgdb* (graphic interface for *gdb*), *ddd* (debugger for several languages).

1.9. Distribution characteristics

This section contains some unique aspects of the distributions of both the Red Hat/Centos/Rocky/Fedora branch and the Debian/Ubuntu/Mint branch, which are two of the large branches of distribution, as mentioned by [Soy]:

- **Using the grub boot loader** (GNU utility). Unlike older versions of most distributions, which used to use *lilo*, current distributions use `grub` (Grand Unified Bootloader). It has a text mode setting (typically */boot/grub/grub.conf*) but is modified via scripts in */etc/grub.d* (or */etc/defaults/grub** or */etc/defaults/grub.d* in Debian/Ubuntu) and also supports modification at booting. Most current distributions have already migrated to GRUB2, which includes a more modular configuration system, as well as expanded Linux kernel parameter support, and improved support for various operating systems.

- **Alternative management**. If more than one equivalent software is present for a particular task, a directory (*/etc/alternatives*) indicates which alternative is used. This system was borrowed from Debian, which makes extensive use of it in its distribution. For example, when we enter `update-alternatives --list vi` the system will display */usr/bin/vim.tiny* which means that when a user enters the `vi` command, `vim.tiny` will actually run.

- **TCP/IP network service manager program based on *xinetd***. The classic service management super-daemon in *Nix called *inetd* has been replaced by an enhanced version called *xinetd* (eXtended inet) which is configured via */etc/xinetd.conf* or in the */etc/xinetd.d* directory. In almost all distributions this super-daemon is not installed by default, and almost all of them have migrated to the daemons management/systemd-based service boot environment (see below) so it is not necessary in many cases.

- **Special configuration directories**. The distributions have organized better some of the configuration directories such as: */etc/profile.d*, files

that run when a user opens a shell; */etc/sysconfig*, configuration data for various aspects and services of the system (RHEL branch); */etc/cron*, various directories specifying jobs to be done periodically (via *crontab*); */etc/pam.d*, where PAM (Pluggable Authentication Modules) are a set of libraries that allows the administrator to configure how users authenticate so this task should not be done by applications; */etc/logrotate.d*, configuration of the rotation of log files (usually in */var/log*) for different services and that indicate when a new version must be made, deleted, compressed, etc.

- **Expanded FHS**. For example, Debian has added some particulars about the standard directory structure in */etc*, such as: */etc/default*, configuration files, and values by default for some programs; */etc/network*, network interface data and configuration scripts; */etc/dpkg* and */etc/apt*, package management tool configuration information, */etc/alternatives*, links to default programs, in those where there are (or may be) several alternatives available.It also includes for many software packages the ability to reconfigure them after they are installed, by using the `dpkg-reconfigure` tool. For example:

```
dpkg-reconfigure locales
```

that allows choosing the regional language and character set settings that will be available, or

```
dpkg-reconfigure tzdata
```

which allows resetting the default time zone.

- The services and boot system have traditionally been managed by **SyVinit** which is based on a daemon (init) and a set of scripts in */etc/init.d* and start and stop execution scripts of each level managed by */etc/rc.X*. Currently, most distributions have migrated to a new concurrent service boot system, based on a different philosophy that makes it safer and more efficient, called **systemd** (and which will be seen in the next section).

2. Boot levels and services

Historically, the distributions based the OS boot-up and initial services through a system called *Sysvinit* (or *SysV* type *init*), based on daemons (background execution processes and always active) boot-up levels and associated services. In this system, the activation or stopping of services was performed by using sequential scripts (in */etc/init.d/)* which was an inefficient system and very sensitive to errors due to poor configurations (if we wish to expand on this system, there is a lot of available literature, such as *init* or *Debian Adm Handbook*).

Nowadays, most distributions have migrated (or are in the process) to a new system called *systemd* that has received significant momentum especially after the decisions of the two large distribution branches to adopt them as a default boot system: Debian/Ubuntu/Mint and RHEL/ Fedora/CentOS. An equivalence between commands from one system or another published by Linoxide may be useful for administrators skilled in `sysvinit`.

Systemd is a system management daemon designed for Linux, which replaces the *init* process and, therefore, is placed as a process with PID=1 (although in some distributions, `systemd` daemons are observed as children of the PID=1 process and in others, such as Ubuntu, there is a */sbin/init* process with PID=1 but if we look in detail, it is a link to */lib/systemd/systemd*) at boot of the Linux system. Also the same name, *systemd*, is used to name the complete package of system components, including the daemon `systemd`. This (`systemd`) is the first daemon to start and the last to finish during the shutdown process.

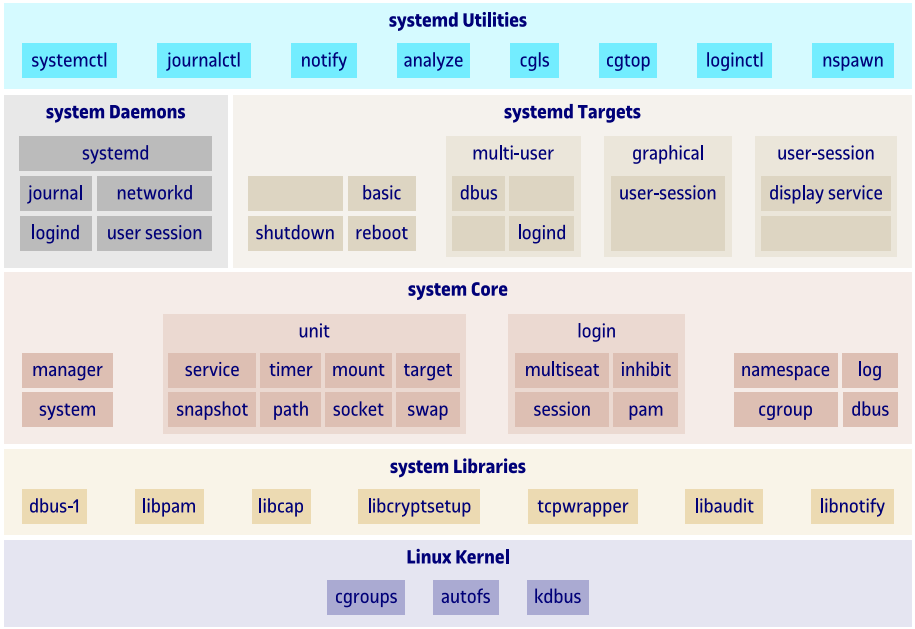
The main design objectives were to be able to express dependencies between daemons to run them concurrently (whenever possible) in the interest of efficiency (reduced boot time) and to improve blockages due to defective configurations. The design and integration of *systemd* has not been free from community criticism, basically because it is an attempt to gather a large number of functionalities in a single environment (in the opposite direction of the UNIX philosophy, of small, interconnected tools, each performing specific and well-defined tasks).

Note

The Boycott systemd site collected much of these criticisms of systemd and some of the discussions or more arguments against and alternatives to systemd can still be found.

Currently, `systemd` has brought together or offers alternatives to a whole series of classic daemons, such as *udev*, *sysvinit*, *inetd*, *acpid*, *syslog*, or *cron*, among others. In addition, network configuration components have also been added, among others, by adding even more dispersed components and functionalities, to a single block of functionality, all controlled by `systemd`. The figure below shows this union of services and functionalities.

Figure 2. Some systemd components



Source: adapted from Wikipedia.

In systemd we can distinguish, among others, components such as:

- **Unit files**: where the initialization instructions for each daemon are collected in a configuration file (called a "unit file"). There are different types such as: *service, socket, device, mount, automount, swap, target, path, timer, snapshot, slice* and *scope*.

- **Core components**:
  - `systemd`, the main daemon of the systemd system acting as a service manager for GNU/Linux.

  - `systemctl`, as the primary command to control the status of the system and service manager, as well as to manage the different units of the system.

  - `systemd-analize`, to analyze the performance of the booting system, and to obtain statistics about it. For example, with `systemd-analize blame`, the booting time of each unit will be obtained and with the option `plot` a file in svg format is generated.

  - Utilities such as `systemd-cgls` and `systemd-cgtop` allow the use of control groups by systemd to be tracked. The Linux kernel uses cgroups to track processes (rather than using -PID- process identifiers). Additionally, systemd also uses cgroups to manage Linux containers using commands such as `systemd-nspawn` and `machinectl`.

- A series of accessory **components** (may vary depending on the current version of the systemd), most of which are daemons (those ending in d in the following list) of the systemd-name form:

- **console**: `systemd-consoled` provides a daemon for the management of text consoles, in order to replace the GNU/Linux virtual consoles/ terminals.

- **journal**: `system-journald` is the daemon responsible for the control of the events log, managed in systemd by read-only binary log files. Logs can be managed within systemd, with *system-journald*, *syslog-ng*, or *rsyslog* depending on the administrator's decision.

- **logind**: deals with system user access.

- **networkd**: it is responsible for managing network interface configurations.

- **timedate**: it controls options related to system time, and time zones.

- **udev**: is a device manager responsible for dynamically managing them using the system */dev* directory, and the actions of adding or removing devices from the user space, as well as firmware load control.

- **libudev**: standard library for using *udev*, which allows third-party applications to access resources managed by *udev*.

- **systemd-boot**: a system bootloader (derived from a previously existing one called *gummiboot*), with support for UEFI firmware (Unified Extensible Firmware Interface), which is a model for the interface between the OS and the firmware, providing support for booting and not to be confused with the MBR boot code method known as legacy BIOS systems or standard boot by BIOS). In principle, *systemd-boot* was initially intended as a lightweight replacement for GNU Grub.

Some graphical frontends are also available for managing services, and consulting the available units, such as *systemd-ui* (for Gnome environments, also known as *systemadm*) and *systems-kcm* (for KDE environments).

As mentioned above, `systemd` introduces the concept of units, which are represented by configuration files, and basically represent information about a system service, listening sockets, system statuses, or other objects relevant to booting.

The files are in (in reverse order of importance if there are equal files in different directories):

- */usr/lib/systemd/system/*: system units (usually from packages).

- */run/systemd/system/*: units created at runtime.

- */etc/systemd/system/*: units created and managed by the administrator.

Regarding unit types, we can find (among others):

Table 5

Unit of	File extension	Description
Service	*.service*	A system service
Target	*.target*	Systemd unit group
Device	*.device*	Device file
Automount point	*.automount*	A filesystem automount point
Mount	*.mount*	A filesystem mounting point
Snapshot	*.snapshot*	A saved status of the systemd manager
Socket	*.socket*	A point of communication between processes
Timer	*.timer*	A systemd timer

For the management of system services, service units (.service) are used, which are used for purposes similar to the old service files present in */etc/init.d/*. `systemd` is basically managed with the `systemctl` command and parameters such as *start, stop, restart, enable, disable, status, is-enabled* to manage the services.

Some distributions still maintain by compatibility commands from previous systems (such as *init* or *upstart*) as, for example, `service` or `chkconfig`, to reduce the impact, but in reality these all move the syntax to `systemctl`.

A command such as:

```
systemctl list-units --type service --all
```

will provide the status of all services (or service type units in systemd terminology).

```
systemctl list-units-files --type service
```

lists the name of the service with its full name and information on whether it is active or not. For a particular service it can be done with:

```
systemctl status service_name                                    e.g. ssh
```

To ask if the service is active:

```
systemctl is-active service_name
```

To ask if the service is enabled:

```
systemctl is-enable service_name
```

In relation to the execution levels (runlevels), `systemd` provides different equivalent units, in target format with names such as: *power off, multi-user, graphical, reboot*, which can be listed with the first command and changed with the second command:

```
systemctl list-units --type=target -all
systemctl isolate name.target
```

To know the current default target (or set it with *set-default*):

```
systemctl get-default
```

For the different machine shutdown and restart options, the following options are available: *halt, power off, reboot, suspend, hibernate.*

One particularity of `systemd` is that it can also act on remote machines via `ssh` connection, for example, a command can be executed with:

```
systemctl --host user@host_remote command
```

To investigate errors from different units, we can do the following:

```
systemctl --state=failed            Displays the units that caused faults
systemctl status unit_with_faults           Displays additional unit information.
```

Or detailed information (to be expanded below) can also be obtained from the PID of the process involved by examining the corresponding journal (logs):

```
 journalctl -b PID_ID
```

Systemd considers many possibilities, depending on the units used and the multiple components available and which will be discussed in subsequent sections. An interesting source of detailed systemd documentation is that developed by ComputerNetworkingNotes in systemctl, services, targets, unit files and units types & states.

3. System status

One of the main daily tasks of the administrator (root) will be to verify the correct operation of the system and to monitor the existence of possible errors or saturation of the machine resources (memory, disks, etc.). This section will detail the basic methods for examining the system status at a given time and taking the necessary actions to prevent further problems.

3.1. Booting the system

During the booting of a GNU/Linux system, a set of interesting information is produced, such as the detection data of the characteristics of the machine, devices, system services, etc., and any problems that may exist are mentioned. In addition, virtual file systems are created with the information so that it can be manipulated in the user-space and in the kernel-space.

In some distributions, much of this boot-up information (in the form of events) can be seen on the system console directly during the boot-up process or simply by switching to it (for example in Ubuntu by selecting Crtl+Alt+F1). However, the speed of the messages (or as mentioned, they are hidden behind a graphic cover sheet) may prevent paying attention to the messages correctly, so we will need tools/access to the information saved to analyze this process.

Basically, we can analyze:

- **dmesg command**: gives the messages of the last kernel boot.

- ***/var/log/messages* file**: general system log, which contains messages generated by the kernel and other daemons (there may be many files with different logs, which are usually in */var/log*, but it depends on its *syslog* service configuration at */etc/syslog.conf*).In some modern distributions, the log system has been replaced with *rsyslog*, which typically has the *log* set to */var/log/syslog*. In other distributions that use `systemd` as a booting system, it also handles the log of events with a component/daemon called `journald`, in this case the `journalctl` command is available, to obtain the list of events from the system log in CLI (and they will also generally be seen in */var/log/)*.

- **Uptime command**: indicates for how long the system has been active.

At booting time, a series of virtual file systems are also generated, which allow us to access information about the boot process, and the dynamic state of the system, from both the user space and the kernel space, and its various components. Among the virtual file systems used, we must highlight:

- **System */proc***: file pseudosystem (*procfs*) that is used by the kernel to store process and system information.

- **System /sys**: pseudo file system (sysfs) that provides a consistent way to access information from devices and their drivers.

- **System /config**: based on *Configfs*, it is similar to *sysfs*. It is only found in some distributions, is complementary and allows creating, managing and destroying objects/data in kernel space, from the user space. It will typically be mounted in */config* or in */sys/kernel/config*.

- **tmpfs file system**: it is typically used to mount temporary storage space (virtually, in volatile memory, without disk storage). Several layouts use it to mount the */run* directory and its subdirectories.

- **devtmpfs file system**: provides device information and it is mounted in */dev* during machine booting. This information will be used by the *udev* device manager (via their *udevd* daemon) which is integrated into systems with systemd.

3.2. */proc* directory

The kernel, during its booting, creates the virtual file system procfs and mounts it in */proc*, where it will save most of the system information. The */proc* directory is deployed on memory and is not saved on disk and data in files/directories can be either static (not modified during OS execution) or dynamic (created/destroyed/modified during execution). Because its structure depends on the activity of the kernel, the files/contents may change in different executions/versions of the kernel.

In the */proc* we will be able to find the images of the running processes, along with the information that the kernel manages about them and each process will have a directory with PID of the process (*/proc/<pidprocess>)* that will include the files that represent their status. This information will be the primary source for debugging programs, or for the system's own commands, such as `ps` or `top`, or `htop` to view the status of running processes. A set of system commands query the dynamic system information from */proc* and some specifics found in the *procps* package.

In addition, the global status files of the system can be found in */proc*, a summary of which is shown below:

Table 6

File	Description
/proc/bus	PCI and USB Bus information directory
/proc/cmdline	Kernel boot line
/proc/cpuinfo	CPU information
/proc/devices	List of character or block system devices
/proc/driver	Information on some hardware kernel modules
/proc/filesystems	File systems enabled in the kernel
/proc/scsi	SCSI, in the scsi disk characteristics file
/proc/interrupts	Hardware interrupt map (IRQ) used
/proc/ioports	I/O ports used
/proc/meminfo	Memory usage data
/proc/modules	Kernel modules
/proc/mounts	Currently mounted file systems
/proc/net	Directory with all network information
/proc/scsi	SCSI Device directory, or SCSI emulated IDE
/proc/sys	Access to dynamically configurable kernel parameters
/proc/version	Kernel version and date

3.3. Kernel: /sys directory

The sys virtual file system makes the device and driver information visible in the user space (this information is available to the kernel) so that other APIs or applications can easily access device information (or its drivers). It is often used layered as the `udev` service for dynamic device monitoring and configuration.

/sys contains a tree data structure of the devices and drivers and is then accessed via the *sysfs* file system (whose structure can vary among different versions). When the kernel detects a device, a directory is created in *sysfs* and the parent/child relationship is reflected with subdirectories under */sys/devices/* (reflecting the physical layer and its identifiers). Symbolic links are placed in the */sys/bus* subdirectory, reflecting how the devices belong to the different physical buses of the system and in */sys/class* the devices grouped are displayed according to their class, such as network, while */sys/block/* contains the block devices.

Some of the information provided by */sys* can also be found on */proc*, but gradually, in the different versions of kernels, it is expected that device information will be migrated from */proc* to */sys* to centralize all its information.

3.4. Udev: */dev* device management

Unlike traditional systems, where the device nodes, present in the */dev* directory, were considered as a static set of files, the *udev* system dynamically provides the nodes for the devices present in the system.

Udev is the device manager for the Linux kernel (which replaces previous ones such as *hotplug* and *devfsd*). This service (now included in `systemd`), manages the device nodes in the */dev* directory, as well as the events generated in user space, due to the new insertion/deletion of hardware devices. Among other features, *udev* supports:

- Allocation of persistent device identifiers, thus avoiding the allocation by order of arrival (or connection) of devices to the system. For example, for storage devices where each one is recognized by a file system id, the disk name, and the physical location of the hardware where it is connected. Exactly the same as the network devices that are identified by the position they occupy on the bus where they are connected.

- Notification to external systems of device changes.

- Creation of a dynamic */dev* directory.

- Execution in user space, thus avoiding the need for the kernel to name devices by allowing the use of specific device names from the device properties.

The *udev* system is basically made up of three parts:

- The **libudev library**, which allows access to device information. Depending on the implementation, this library may be isolated, or as in the case of distributions with `systemd` boot, it has come to be included as a built-in feature of it.

- A **user space daemon**, *udevd*, that manages the virtual */dev* directory. The daemon communicates through a socket established between the kernel and the user space, to determine when a device is added/removed from the system and *udev* manages operations as needed (node creation in */dev*, module/driver load needed from the kernel, etc.). In systems with systemd the operation is controlled by *systemd-udevd*.

- The **udevadm command**, which allows interaction to manage behaviour/control/obtain *udev* information.

Historically, in the */dev* directory are the system device files (nodes) so that a program in user space can access a hardware device or a function of it. For example, the device file */dev/sda* is traditionally used to represent the first disk of the system (SATA bus) and the name sda corresponds to a pair of numbers called the major and minor of the device, and which are those used by the kernel to determine which hardware device it is using. Each of the major and minor numbers is assigned a name that corresponds to the device type as can be seen in the kernel information.

Given its limitations (number scheme, dynamic assignment with removable devices, or the large amount of numbers required) but especially to the non-static situation of */dev*, the name of the device was delegated to *udev* with a series of steps/rules to determine a unique name as needed (thus avoiding creating a large number of devices that probably would not be used). For this purpose, the following is used:

- **Label** or serial number that can be identified by device class.
- **The number of the device** on the bus where it is connected (for example, its identifier on a PCI bus).
- **Bus topology**: position on the bus to which it belongs.
- **Replacement name** in case there is a match between multiple devices.
- **Name in the kernel**.

If the above steps cannot provide a name, the one available in the kernel will be used.

The *udev* system, whose information typically resides at */lib/udev* and */etc/udev* (there may also be a */run/udev* at runtime), among others, includes a number of rules for the naming and the actions to take on devices in */lib/udev/rules.d* (default rules) or */etc/udev/rules.d* (managed by the administrator and take precedence over the default rules).

If *udev* needs to load a driver, it will use the `modalias` to load the correct driver that will be found from the information of the modules that ends up creating (via *depmod* when installing new modules) the corresponding file in */lib/modules/`uname -r`/module.alias*.

3.5. Processes

Among the processes that are running at a given time, we can find 3 types:

- **System processes**, associated with the local operation of the machine, kernel, or processes (called daemons) associated with the control of different services and can be local, or network (when the machine acts as a server providing a service). These are typically associated with the root user although they are usually migrated to pseudo users specialized by service, for example, bin, www-data, mail, etc. (which are "virtual" and non-interactive users) used by the system to execute certain processes.

- **Administrator user processes**: root user processes (interactive processes or running applications) that will also appear as processes associated with the root user.

- **System user processes**: associated with running user applications, whether interactive tasks or not, in text or graphic mode.

The most useful commands for managing processes are listed below:

- `ps`: most used command for versatility and information, list processes, time, process identifier, status, resources and command line used. It supports different parameters/syntax and options between the most used `ps -edaf` or in BSD `ps aux` syntax (see `man ps`).

- `top` (or `htop`): a command that dynamically displays processes by updating changes as well as statistics regarding CPU and memory utilization.

- `kill` (`killall`): it allows deleting processes in the first case by the PID, and in the second case by the name or other parameters. `kill` sends signals to the process such as `kill -9 PID_ID of the_process` (*9* corresponds to *SIGKILL*) or `killall firefox` where it will delete all the processes of the user who executes it and which are called firefox. See also skill possibilities, for example, to delete all processes for a user with `skill -STOP -u user_name`.

- `pstree`: allows generating a process dependency tree. It should be remembered that in *Nix, when the parent process is deleted, all the child processes are deleted, so it may be interesting to know this relationship to manage a set of processes (using `ps -edaf` some additional information can be extracted in the PPID column).

- Suspend an interactive process: this can be done with Crtl-z which will return the control to the terminal and then resume it with the `fg` command.

- Change the execution priority of a process: through the `nice` and `renice` commands we can modify the priorities of the process and change the access they have to the resources.

- It is recommended to consult `man 7 signal` for the valid signals to send to a process.

3.6. System logs

Both the kernel and many of the service daemons, as well as different GNU/Linux applications or subsystems, can generate messages that are saved in files as a trace of their operation and that will also serve to detect errors, or critical situations. These logs are essential in many cases for administration tasks and often take a lot of administration time to process and analyze their contents.

Note

There are several log managers depending on the distribution used; the classic UNIX and GNU/Linux system is initially *Syslogd*, but progressively the *rsyslog* and *Journald* systems (part of systemd) have begun to be used as alternatives, or in conjunction with *syslogd*.

Most logs are stored in the */var/log* directory, although some applications may modify this behaviour. Most of the logs of the system itself are in this directory and are usually set to */etc/syslog.conf or /etc/rsyslog.conf* (depending on the daemon that manages them).

The daemon *syslogd* (or its alternatives such as *rsyslog or journald*), which is responsible for receiving the messages that are sent by the kernel and other service daemons and sends them to a log file (usually */var/log/messages* but it can be configured at */etc/syslog.conf*). The configuration allows redirecting the messages to different files depending on the daemon that produces them and/or also classify them by importance (priority level): alarm, warning, error, critical, etc.

As mentioned, most distributions use *rsyslog* and *journald* and their configuration may be different, for example, Ubuntu uses `systemd` -managed *rsyslog* that can be checked by making `systemctl status rsyslog` and the log master file is */var/log/syslog* in addition to others and is configured with */etc/rsyslog*). To look at the latest log, we often use (with `rsyslog`):

```
tail -f /var/log/syslog
```

which will show any changes to the device that may appear in the file.

Other useful commands for extracting information from the system are:

- `uptime`: time since the system is active and charging.

- `last`: it analyzes system inputs/outputs (*/var/log/wtmp*) *log* of users, and system bootings. Or `lastlog`, control of the last time users accessed the system (information in */var/log/lastlog*).

- Commands or applications for combined log processing: commands that issue summaries (or alarms) of what happened in the system such as `logwatch`, `logcheck` or display of the logs in graphical form such as `gnome-logs`.

- `logger`: allows inserting messages into the log system such as `logger "Checkpoint at `date`"` which then with tail /var/log/syslog we can see as *Apr 24 10:48:28 aopcrs adminp: Checkpoint at Sun 24 Apr 10:48:28 CEST 2022*

rsyslog (through its configuration in */etc/rsyslog*) allows greater control of filters and TCP utilization as a configuration mechanism to receive/send logs from one machine to another (remote log management) allowing deployment of a centralized server of logs from multiple client machines that also incorporates native support for MySQL and Postgres databases so that logs can be stored directly in these databases (among others).

With regard to the other daemon that has been mentioned (*journald*), included in the systemd, it has aroused many criticisms since it uses binary formats to save the logs (unlike the previous ones that are in text mode), but it has gained importance in the distributions that support systemd (or in some it is a combined method between both daemons (see for example `man systemd-journald` or `man rsylogd` in Ubuntu for example and `systemctl status rsyslog` or `systemctl status systemd-journald`).

To read journald logs:

```
# journalctl
```

With `systemctl status systemd-journald` we can see where the logs are being saved (for example, in Ubuntu in */run/log/journal/...*) and which will be processed by `journalctl`. The configuration file is in this case at */etc/systemd/journald.conf*. There are different parameters for journalctl, among the most interesting ones, we have:

- `journalctl -b`: Last boot messages (similar to `dmesg`)

- `journalctl -b -p err`: Error messages on the last boot.

- `journalctl --since=yesterday`: Messages appeared since yesterday (combine --since --until for a period).

- `journalctl -f`: The process becomes active and will display the next messages as they are generated by the terminal.

- `journalctl _PID=1`: It ists process messages with a particular PID, UID, or GID.

- `journalctl /usr/sbin/cron`: Messages from a specific executable.

- `journalctl /dev/device`: Messages regarding a specific device.

- `journalctl –disk-usage`: Disk space used by logs.

- `journalctl _TRANSPORT=kernel`: Messages from the kernel (equals -k).

- `journalctl –list-boots`: It lists the latest system boots and their timestamps.

- `journalctl _SYSTEMD_UNIT=crond.service`: Messages from a systemd service, or unit controlled systemd.

- `journalctl -F _SYSTEMD_UNIT`: Units available for log.

See the pages of the *journalctl* manual, and section 7 of *systemd.journal-fields*, for a comprehensive description of available fields and options.

3.7. Memory

Regarding the system memory, it should be noted that it is divided into two: physical memory and virtual memory. Normally (unless specific servers), large amounts of physical memory will not be available and it is advisable to use virtual memory (called swap memory) that will be implemented on the disk. This memory (swap) can be implemented as a file in the file system, although it is usually defined as a partition dedicated to this purpose (swap partition), created during the installation of the system and of the 'Linux Swap' type.

To examine memory information, there are different commands/files:

- `ps`: it allows knowing which processes are running and with the percentage and memory used options.

- `top`: is a dynamic ps version that is upgradable by time periods and can classify processes based on memory used or CPU time.

- `free`: information about the global state of physical memory and virtual memory.

- `vmstat`: information about the status of virtual memory, and its use.

- `lsmem`: lists the available memory ranges and their status.

- `pmap PID`: information about the memory map of a process.

- */etc/fstab* file: the swap partition (if any) appears, also with `fdisk -l` we can find out its size or consult */proc/swaps*.

- `dstat`: is a replacement for some system statistics commands such as `vmstat`, `iostat`, and `ifstat` that overcomes their limitations and is much more versatile with additional options (usually not installed by default).

3.8. Disks

One of the complementary aspects of CPU and memory management is the disk subsystem, since it will maintain the (massive) information when the computer has no power (there is some type of memory that keeps its information without power such as that used in USB (disks) or solid-state disks, but they must be differentiated from the RAM memory discussed in the previous section since it loses its content when it has no power). Below we will see what technologies exist, how they are organized, what partitions are, and what file systems can be used within a wide variety. In *Nix (just as all OSs), a disk must be partitioned (divided into sections) and on each of them create a filesystem for it to be used. To do this, the administrator must 'mount it' (associate it to a subdirectory from the / directory) and then it will be accessible to the system/users. This can be done manually (for example with `mount -t ext4 /dev/device /mnt`) or programmed via an entry in the */etc/fstab* file.

To know the available disks (or storage devices) we can look at the system boot information (with the `dmesg` command or in the file*/var/log/messages*), where devices such as */dev/hdx* are detected for IDE devices (currently in disuse) or SATA/SCSIs with */dev/sdx* type devices. Other devices, such as USB-connected hard drives, solid-state drives (usually called pendrives), removable drives, external CD-ROMs, are often devices with some type of scsi emulation, so they will also look like */dev/sdx* type devices. It should be noted that the different layers of hardware+software in the operating system hide the disk technology that can be magnetic (mechanical) or solid state (electronic only) and called SSD (Solid State Disk). The latter are increasingly encompassing market share

due to their speed, consumption, acceptable price in relation to magnetic disks (although these remain, for now, those used when large quantities of storage are needed (10-15 TeraBytes disks are still magnetic).

Any storage device will present a series of partitions of its space and typically a disk with the classic MBR format supports a maximum of four physical partitions, or more if they are logical (multiple partitions of this type can be placed on a physical one). Each partition can contain different filesystems, either from the same operating system or from different ones.

The classic partition format (developed in the 1980s), has a structure called MBR (Master Boot Record), and is a special type of boot sector (512 or more bytes in the first sector of the disk), which saves the disk partitions as organized, as well as containing code (loader) that allows the disk to boot on an operating system resident in any of the partitions, starting at the request of the BIOS.

MBR has some important limitations, but the two main ones are that it supports only 4 physical partitions called primary (or 3 primary ones and an extended one that can accommodate 128 logical partitions) and a disk capacity limitation of 2TB. Because of these limitations, the current disk partitioning scheme is migrating to the GPT (GUID Partition Table) scheme that does not have these limitations. New versions of BIOS such as EFI and UEFI support GPT, in addition to allowing access to disks larger than 2TB, but there are some manufacturers that have made various modifications to the MBR scheme to support the two schemes and support GPT as well.

GPT is part of the UEFI (Unified Extensible Firmware Interface) specification that replaces traditional booting with BIOS (although most computers maintain both methods) for EFI-supported systems, UEFI (e.g., OS X and Windows). On Linux (and some BSDs), they can be booted from GPT, with/without EFI/UEFI support in older BIOS, and they have the `gdisk` command (which allows managing the partition table for GUID disks), `fdisk`, and Grub2 boot with GPT support included. In the case of `fdisk`, the support is not complete and it is recommended to use `gdisk` for full GPT disk support.

The `fdisk` command (on MBR disks, in GPT `gdisk` or `gparted` is recommended), or any of the `fdisk` variants (such as `cfdisk`, `sfdisk`). For example, when examining a disk where Linux is installed:

```
# fdisk -l /dev/sda
Disk /dev/sda: 5 GiB, 5368709120 bytes, 10485760 sectors
Disk model: XX.YY HARDDISK
Units: sectors of 1 * 512 = 512 bytes Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
Device     Boot   Start      End Sectors  Size Id Type
/dev/sda1  *       2048  8984575 8982528  4.3G 83 Linux
/dev/sda2       8986622 10483711 1497090  731M  5 Extended
/dev/sda5       8986624 10483711 1497088  731M 82 Linux swap
```

5GiB MBR disk with three partitions (identified by the number added to the device name), where we can see a boot partition (Boot column with *) Linux type, an extended partition, and within a swap logical partition (we can see that sda2 and sda5 share the same start/end sector although the swap has 2 bytes less (for alignment reasons) than the extended one. Similar information can be found in */proc/partitions*.

During booting, the system mounts the default partitions and there may be others that can be mounted on-demand or automatically mounted when a device is available (e.g., USB drives). The most important commands and files for obtaining information from disks in addition to those mentioned are:

- `mount`: it gives information about mounted filesystems (either real devices or virtual filesystems, such as /proc or loop). This information can be seen from the */etc/mtab* or */proc/mounts* file. It also allows (the administrator or user if enabled on */etc/fstab*) to mount a disk for example with `mount -t ext4 /dev/sda4 /mnt`.

- `df -h`: reports on filesystems and allows verification of used and available space. This command easily allows to monitor disk space and prevent a filesystem from becoming saturated (especially if it is the /) since below 10-15% free could lead to system errors and bad functioning.

- `du -h directory`: command that provides information about the space occupied by a given /directory/subdirectories and that includes many parameters.

- `tree`: it displays the directory tree visually.

- `udiskctl`: it allows interaction with the daemon *udisk* and provides an interface between block devices (disks) and allows information to be extracted from them, for example `udiskctl dump`.

- `sync`: it allows us to synchronize cached writings (memory) with the disk device to avoid inconsistencies in it.

- `blkid`: it displays all disk information including UUID (single disk identifier).

- `gdisk`: it is a command for modifying partition tables (GPT) on a GUID disk.

- `gnome-disks`: graphical utility for managing disks.

- `gnome-disk-image-mounter`: it allows mounting files that are disk images.

- `ldmtool`: it is a tool for managing Microsoft Windows dynamic disks.

- `lvmdiskscan`: lists devices that can be used as physical volume.

- Create a disk in RAM: on certain occasions it may be necessary to create a disk in memory; in order to do it: `mount -t tmpfs -or size=10G myramdisk /tmp/ramdisk` where the directory */tmp/ramdisk* must exist previously. Another way will be with the ram device (see `man ram`)

- ***/etc/fstab*** file: it indicates which devices must be mounted at booting or which can be mounted by users. This does not prevent other discs from being mounted on demand with the `mount` command, or dismantling them with `unmount`.

To maintain the occupied disk space, it is necessary to delete old temporary files (for example in */tmp* and */var/tmp)* and log files to prevent overgrowth (the `bleachbit` program is recommended to be used in graphical mode). For systematic deletion of log files, the logrotate command is recommended, configured through */etc/logrotate.conf* and the */etc/logrotate.d* directory (with a file for each application to manage its logs).

There are also a number of files that the administrator will need to review (and delete if necessary) which may include: core files (these are large files with images from the memory of programs that have caused an error), among which we can mention: email system (in */var/spool/mail* or in the user directory), browser caches or other applications (usually in the user directory) and HOME directories of the users themselves (in general).

4. Energy management

As a special case of control devices similar to *udev*, but not integrated into it, is the management of energy-related devices (and in some distributions also the disks). These components are integrated into the *freedesktop.org* initiative, which aims to develop interoperable technologies for developing desktop environments. Various desktop environment projects, such as GNOME, KDE and Xfce, collaborate with freedesktop.org (also known as X Desktop Group or XDG) to develop methodologies and techniques that add value to the developments of these environments.

One of the active projects for GNU/Linux is D-bus, which is an interprocess communication (IPC) mechanism, and which allows processes, running concurrently, to communicate with each other to exchange information or services. In GNU-Linux, it is used to communicate running desktop applications in the same desktop session, as well as communicate the desktop session with the operating system, either the kernel, or daemons or particular processes.

D-Bus is currently used in KDE, Gnome, and Xfce desktop applications, so two applications/processes can exchange messages with each other, through the infrastructure provided by the project. In *systemd*, the D-Bus code was rewritten and became more efficient, which has increased its performance. There is a project called *kdbus* (Linux Kernel D-Bus implementation), which will offer an alternative in the future, and which implements peer-to-peer communications to implement IPC through kernel mediation. D-Bus interactions can be viewed and monitored via the busctl or busctl monitor command.

Another project maintained by freedesktop.org is called DeviceKit and have enabled the development of Upower and Udisks as interfaces and services, through D-Bus, to manage energy management and storage devices. Since many distributions do not use `udisk`, only `upower` will be seen, which allows listing energy devices (AC or batteries), listening to device events, and asking about their history or statistics, for example: `upower -e` (lists battery devices, AC or display), `upower -d` (shows all parameters), `upower -i <device_name>` (provides device-specific information obtained with -e), `upower --monitor` (monitors all UPower daemon events). It can also be obtained with cat */sys/class/power_supply/BAT0/capacity* or by installing the `acpi` utility.

5. File system

On each machine with a GNU/Linux system, file systems of different types can be mounted. It is common to encounter the Linux file systems themselves created in different disk partitions (or remote disks), such as *ext4, raiser, btrfs, zfs, xfs, nffs* (Windows) among others, and where each one has its properties and characteristics.

Two partitions are usually found in a typical configuration:

1) the one corresponding to "/" (root filesystem) and

2) the one corresponding to the exchange or swap file.

In business/professional-oriented configurations, partitions are typically separated to manage and administer the contents better (for example, differentiated partitions for */, /boot, /home, /usr, /opt, /var,* among others). Another common configuration can be three partitions: */*, *swap*, */home*, where */home* will be dedicated to user accounts. This allows the accounts of users to be separated from the system, isolating into two separate partitions, and the necessary space can be given for the accounts in another partition.

> Partitions are made to clearly separate more static or dynamic parts of the system, and when facing saturation issues, facilitate extending partitions, or isolating parts for backup more easily (e.g., user accounts in the */home* partition).

The type of partitions must be indicated (`fdisk` command) according to their objective (for example *swap* partitions is Linux swap type and has ID=82). The one corresponding to the '/ ' is identified as Linux and ID=83 and must have one of the standard file systems, for example *ext4*, which includes journaling (allows having a log of what happens in the file system and then it can be used to recover it more efficiently in case of errors). Ext4 is the usual file system, but the administrator can select Reiser (hardly used as there are no new developments), XFS, and some more extended with important support such as Btrfs and ZFS. The file systems that can be found in GNU/Linux are described below:

- **Traditional systems on Linux/*Nix**: initially *ext2*, *ext3*, (evolution of the previous one with journaling concept) and *ext4*, which significantly improves the benefits of *ext3*. But we can also use: *btrfs* (designed by Oracle, with clear compression and on-line fragmentation), *zfs* (designed

by Sun Microsystems, with snapshots, dynamic disk striping, dynamic error control, it is not opensource but it can be installed on any Linux -Ubuntu uses ZFS for containers), *xfs* (developed by Silicon Graphics, similar to *ext4* with good features for large files), *jfs* (developed by IBM, with low CPU consumption and good features for large and small files).

- **Support for non-GNU/Linux environments**: *msdos*, *vfat*, *ntfs*, access to different *fat16*, *fat32* and *ntfs* (Windows) *hfs* and *hfsplus* (MacOS) systems. Many of these systems are implemented in user space through FUSE (a component that allows managing file systems in user space) with the benefits that it implies.

- **Systems associated with physical supports**, in the case of CD/DVD, such as *iso9660* and *udf*.

- **Network file systems** (widely used): NFS, Samba (*smbfs*, *cifs*), allow access to file systems available on other machines in a transparent manner over the network.

- **Network distributed file systems**: such as GFS, Lustre, Ceph, or HDFS.

- **Pseudo filesystems**, such as procfs (*/proc*), sysfs (*/sys*) or file systems in RAM *tmpfs*.

Detailed information on file systems can be found on Linux at http://www.kernel.org.

5.1. Mount points

Apart from the main/filesystem and its possible splits into extra disk partitions (*/usr /var /tmp /home*), the partitions of other operating systems can be accessed in GNU/Linux if they exist (and that one or the other is selected at boot time). Sometimes, it is interesting to be able to access these files in read or write and there is no problem for Linux to access Windows partitions and files (either FAT or NTFS) and they are accessed through FUSE (module integrated in the kernel that allows access to the files in user space).

For the data to be read or written, the partition has to be available within the root file system (*/*). Therefore, a file system "mount" process must be performed somewhere in the directory tree.

Depending on the distribution, default directories are used, but the administrator can create his/her own. Typically, there are subdirectories of the root, for example */cdrom, /mnt*, */media* (preferred lately by distributions for

removable devices). According to the FHS standard, */mnt* should be used for temporary mount of file systems, while */media* is used to mount removable devices.

```
mount -t filesystem-type device mount-point
```

The assembly process is carried out by using the `mount` command:

The filesystem type can be: *msdos* (fat), *vfat* (fat32), *ntfs, iso9660* (for cdrom), *ext4*, *xfs*, among others.

The device is the corresponding input in the directory */dev* to the location of the device (the IDEs -obsolete- */dev/hdmn*, where '*m*' is *a,b,c,d and 'n'* is the partition number, the SCSIs with */dev/sdmn*, where 'm' is *a,b,c,d*... and 'n' the partition number.

For example:

```
mount -t iso9660 /dev/sdc /mnt/cdrom
```

mounts the CD-ROM (if it is the sdc device) to */mnt/cdrom*.

```
mount -t ntfs /dev/sda2 /mnt/win
```

mounts the second partition of the first device (C drive:Windows), NTFS type in */mnt/win*.

When the disk is no longer needed it can be 'removed' with:

```
umount /mnt/cdrom
umount /mnt/win
```

For removable media, USB/CD/DVD type, `eject` can be used for media removal.

In the device specification, especially in the */etc/fstab* file, */dev/device* is used as an identifier, that can be the device partitions, or RAID type special software (*/dev/mdx*) or LVM (type */dev/mapper*), or UUID, which are unique identifiers assigned to disks or partitions, used at boot time. We can use the `blkid` command or access */dev/disk/by-uuid* to know the associated UUIDs.

Note that all files/directories are given reduced access control lists in 3 groups (owner, group, rest) and with rwx permissions (read, run, write including deletion) so there will be 9 bits left as *rwxrwxrwx*, where the first 3 correspond to the owner, the second ones to the group and the rest to users who do not own or belong to the group. For a directory, *x* denotes the permission to be able to enter that directory (with the `cd` command, for example). To modify

the rights/property over a directory or file, it can be done with: `chown` (change owner of the files, only the root can do it), `chgrp` (change owner group of the files, only the root can do it), `chmod` (change specific permissions (rwx) of the files).

6. Users and groups

As mentioned in the initial module, users of a GNU/Linux system have an associated account (defined with some of their data and preferences), along with disk space so they can arrange their files and directories. This space assigned to the user is his/her property (and a general group as users for example) and can only be used by the user (unless the permissions specify different things). Within accounts associated with users, we can find different types:

- **administrator**: with root identifier (*UID=0*), which is only (or should be) used for administration operations. The root user has all permissions and full access to the machine and configuration files. Therefore, it is recommended to always work with a user account and when we want to perform some privileged activity, such as installing a package, scaling privileges with the `sudo` command or changing the root user with `su - root`. Nowadays, almost all desktop environments do not allow the root user to connect from the interface by default.

- **User accounts**: accounts for any machine user who can perform any activity on their HOME and in some directories such as /tmp but in the rest he/she will only be able to look and in some such as /var this activity will even be restricted.

- **Special services accounts**: *lp, wheel, www-data*... accounts that are not used by people, but by system services (usually to not use the root for managing these services), and normally as shell they have */bin/false* in the file */etc/passwd* to prevent them from being interactive accounts.

The user will be created with an identifier known as a name or login (or user identifier), a password, and an associated personal directory (the account). As a name, the initial of the name and the last name are generally used, all in lowercase, although it is an unwritten rule. System user information (local, unless external directory systems such as YP/NIS or LDAP are used, which will be seen later).

The user can change the password (and it is advisable to do so regularly or at least the first time they enter to change the one that the root user has given them) with the `passwd` command and it must be a combination of letters, numbers and special characters (for example .!-_) and must not be made up of any dictionary word or the like because it can represent a major security problem.

Most distributions currently implement a mechanism based on a summary function (hash) in addition to continuing to use the salt value and save them in */etc/shadow* that is only accessible to the root. The cryptographic hash function is a function that, from an arbitrary block of data, returns a fixed-size bitchain (called digest) to be used as a password since any changes to the input data will change the value of the result string (digest). In addition, salt is continued to be used to avoid that, by comparison and knowing the hash algorithm, the word that generated the digest can be obtained.

The algorithm definitions of which passwords are supported and how, are configured in the PAM (Pluggable Authentication Modules) system, a mechanism included in all Linux to manage the authentication of tasks and services, allowing to model custom security policies for the different services.

The password setting in Debian, for example, will be in the */etc/pam.d/common-password* file where we can see a line indicating `sha512` which is the hash algorithm (512-bit secure hash algorithm). The hash value (digest) will be saved in the */etc/shadow* file (in the second field of the line belonging to the corresponding user) in the following format: *idsalt$value_hash* where *id* identifies the algorithm used and can take the following values: 1(MD5), 2a (Blowfish), 5 (SHA256), 6 (SHA-512). Therefore, 5salt$digest is a SHA-256 and 6salt$digest password in SHA-512. The salt represents a maximum of 16 characters following the "$" symbols following the id, and the encrypted string size is set to MD5=22 bytes, SHA-256=43 bytes, and SHA-512=86 bytes where all password characters are representative and not just the first 8 as in 3DES (the original encryption algorithm). To generate a hash of a password word, the passwd command must be used and the result will be stored directly in the */etc/shadow* file (obviously a user will only be able to change his/her password and not that of other users).

User and group information can be found in the following files:

- */etc/passwd*: user information (name, HOME directory, etc.).
- /*etc/group*: information about user groups.
- */etc/shadow, /etc/gshadow*: encrypted passwords (hash value) of users (shadow) or groups (gshadow) and settings for validity, change, etc.

All of these files are organized by lines, each of which identifies a user or group (depending on the file). On each line there are several fields separated by the character ":" and it is important to know what information they contain. All of this information can be found in the introduction module.

When a user enters the system, after the user/password has been validated, they are placed in their HOME directory and the order interpreter (shell) set to */etc/passwd* is executed on the line corresponding to the user (field 7)

and can begin to execute orders or work interactively with the system. Only the system root (or users in its group) have permission to manipulate the information of the other users and groups, register them, unsubscribe them, etc. (some orders, such as chfn, will allow users to change their information if it is configured that it can be done). Each order for handling users has several different parameters for managing user information and/or fields mentioned above.

The most representative commands for managing users/groups include:

- `adduser`: adds a new user to the system. The way the user is created (if no additional information is specified) can be configured in the */etc/adduser.conf* file. It supports a different set of options to specify the home directory, the shell to use, etc. It is recommended to use this command and not useradd.

- `usermod`: with this order, most of the fields found in the *passwd* and shadow file can be modified, such as the home directory, shell, password expiration, etc.

- `chfn`: it changes the user's personal information, contained in the comments field of the *passwd* file (field 5).

- `chsh`: it changes the user shell.

- `deluser`: deletes a user from the system, and deletes all their files according to the parameters indicated (allowing even a backup), etc. The default settings to be used with this order are specified in the */etc/deluser.conf* file.

- `passwd`: it changes a user's password, its expiration information, or it locks or unlocks a certain account.

- `addgroup`: it allows adding a group to the system.

- `groupmod`: it allows modifying the information (name and GID) for a given group.

- `delgroup`: it deletes a certain group. If any user still has it as primary, it cannot be deleted.

- `gpasswd`: it is used to change the group password.

- `whoami`: it is used for knowing which user we are identified with (it may seem like a nonsense question, but it is a common question when working with different users or as root) and it will display the current user identifier.

- `groups`: used to know which groups the user belongs to and it will display user and group ID.

It is also interesting that the current user can 'become' another user without exiting the current session via the `login` or `su` command (useful action when it is necessary to run/access information as another system user if their password is available) or changing groups with the `newgrp` order. This last order should only be used when it does not belong to the group in question and its password is known (which must be activated in the */etc/gshadow* file). If only the permissions of a given group are needed to execute an order enabled for this group, the `sg` command can be used.

There is a directory (usually */etc/skel*), where we find the files included when a new account is created and they will be the ones that initially configure the user session (environment variables, keyboard, alias, etc.). The administrator must adapt these files according to what their users need as an initial session (each user can then modify them according to their needs).

It is necessary to mention that in some installations the users/groups and other variables/configurations are not defined on each machine but on a server known as user identity/information manager. This allows a set of machines to be used by users without having a local account but on the server which makes the user's computer independent. These services, also referred to as network information systems or directory services, are identified as NIS or LDAP (or in Windows as Active Directory).

To determine if the user is local or from a directory service, we can check if the user is defined in */etc/passwd* and if it is, the user will be local (regardless of whether it is also defined in the directory service). To determine if the user belongs to a NIS system, we can also refer to the *passwd* and *group* lines of the configuration file */etc/nsswitch.conf* if *nis* (or *nisplus*) appears that will indicate that this machine is a client of a NIS server. In general, it does not imply any modification for the user, since the management of the machines is transparent, and especially if combined with files shared by NFS (Network File System) that allows for their account to be available regardless of which machine they work with.

If the user wants to change the password in NIS systems, instead of the passwd command, the `yppasswd` command must be used which will change it in the database and propagate it to all clients in the NIS domain. These concepts will be expanded in the network services section.

7. Print servers

There are two classic printing systems: LPD (Line Printer Daemon) of the BSD-UNIX branch and System V Line Printer (or LPR), common in different UNIX. GNU/Linux has originally integrated both systems, either primarily by using LPD and emulating LPR or depending on the distribution integrating one or the other by default.

LPD is a fairly old system (dated back to the origins of the mid-1980s BSD branch) and is often lacking support for modern devices as it was not intended for current printers. To solve this problem, LPD is combined with Ghostscript, which is an environment that produces output in postscript format for a very wide range of printers with drivers. This software is also combined with filtering software that eventually defines, depending on the file to be printed, the most suitable for the task to be performed. The process with LPR is very similar.

The way to determine which printing system is installed is by checking the main command that in BSD (LPD) system is `lpr`, or in SystemV (LPR) is `lp`.

While these print servers are functional, they offer little flexibility and a complex setup task for quality prints. To solve these problems, the CUPS project was developed with the aim of unifying criteria, making the configuration process (for printers and printing service) simple, flexible and of good quality. Nowadays, CUPS has become the de facto standard for GNU/Linux and, although LPD and LPR are still included for compatibility, they are not commonly used.

At present, the administrator can find local (typically connected to USB) or remote printers that are connected to a network. For this type of printer, we must know the communication protocol used, which can be TCP/IP (for example, in HP printers), or other higher-level ones over TCP/IP, such as IPP (CUPS), JetDirect (some HP printers), etc.

7.1. CUPS

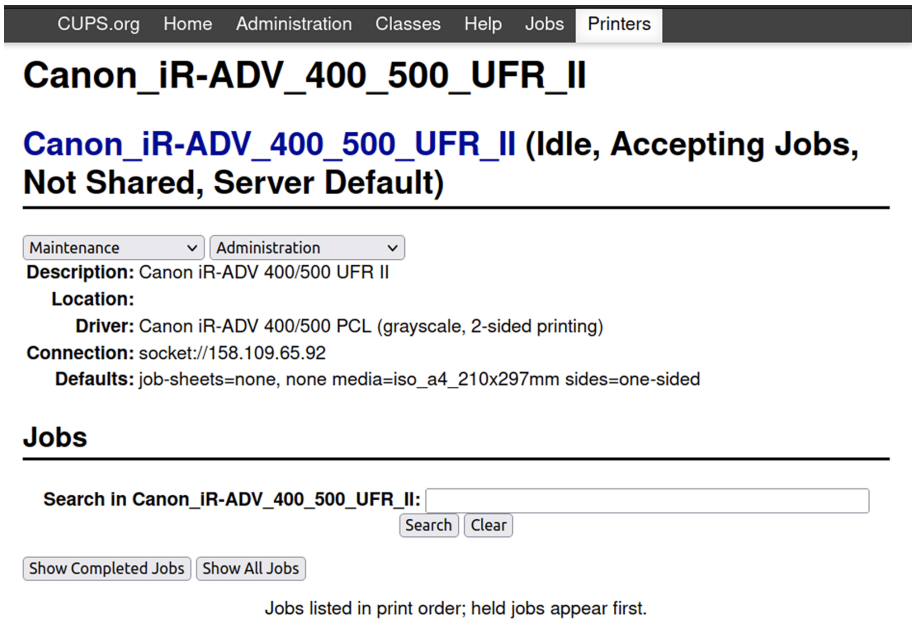
CUPS is a new print system architecture that features a support layer for BSD LPD and allows it to interact with servers of this type. It also supports a new printing protocol called IPP (based on http), but only available when the client and the server are CUPS type and it also uses a type of drivers called PPD that identify printer capabilities and properties.

CUPS has a management system based on a set of files: */etc/cups/cupsd.conf* centralizes the configuration of the printing system, */etc/cups/printers.conf* controls the definition of printers and */etc/cups/classes.conf* controls the groups of printers. In */etc/cups/cupsd.conf* the system is configured according to a series of sections of the file and the policies of the different actions.

It should be noted that CUPS is designed for both clients and the server to operate under the same environment; if clients use LPD, a compatibility daemon called *cups-lpd* (in packages such as *cupsys-bsd)* must be installed. In this case, CUPS will accept jobs, from an LPD system, but it will not control the access.

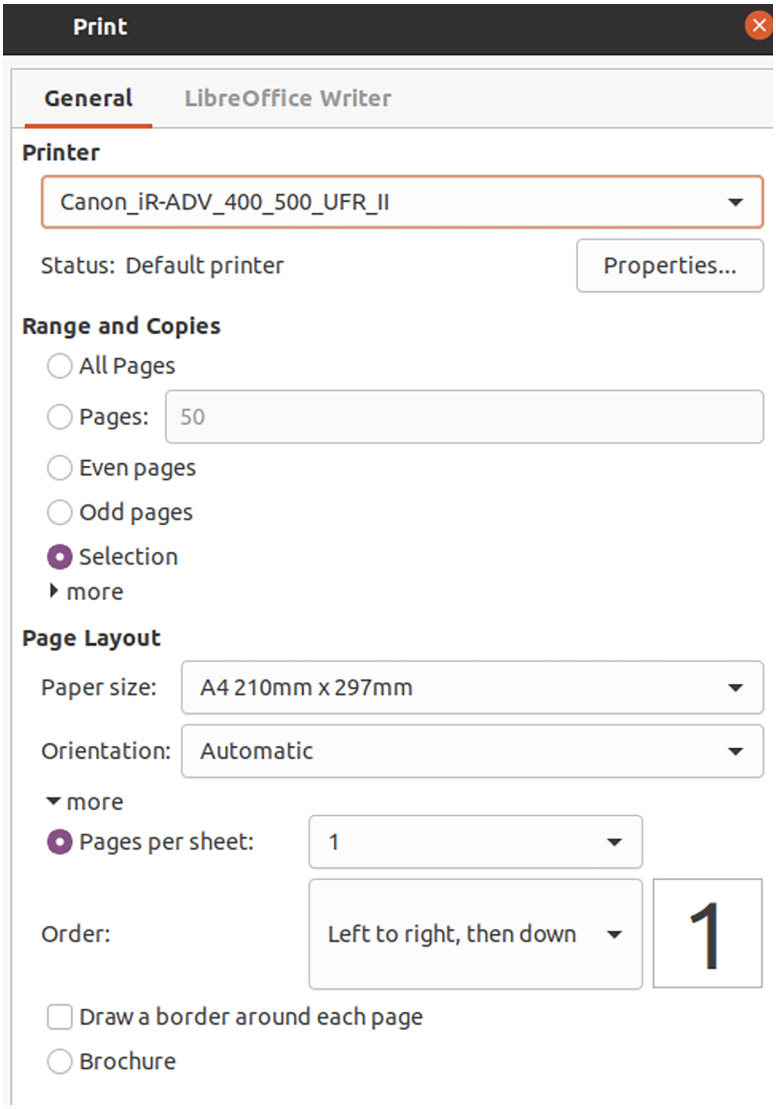
For command line administration, CUPS is somewhat peculiar, as it accepts both LPD and SystemV commands on clients, and the administration is usually done with the SystemV `lpadmin` command. However, graphical tools such as those in the Gnome or KDE control panel are often used or other tools such as `gtklp`, `system-config-printers` are available or the web interface incorporates the same CUPS system into the URL *http://localhost:631* as shown in the following figure:

Figure 3



And it is displayed as shown in the following figure from the client (for example Libre Office):

Figure 4


Print
General
LibreOffice Writer
Printer
Canon_iR-ADV_400_500_UFR_II
Status: Default printer
Properties...
Range and Copies
All Pages
Pages:
50
Even pages
Odd pages
Selection
more
Page Layout
Paper size:
A4 210mm x 297mm
Orientation:
Automatic
more
Pages per sheet:
1
Order:
Left to right, then down
1
Draw a border around each page
Brochure

8. Storage redundancy: RAID

RAID (Redundant Array of Inexpensive Disks) is an environment that allows defining a high-availability storage scheme using low-cost disks without large infrastructures (cabins, fibre, etc.). The main objective of the design is to provide fault tolerance from the device level (disk set), to different possible types of physical and logical faults, and to prevent data loss or system consistency faults. Since we will be working with information replicas, some schematics are designed to increase the performance of the disk system, extending its bandwidth available to the system and applications.

Typical deployment on enterprise-class servers is via hardware cards that allow a RAID to be configured with disks connected directly to the server (DAS, direct-attached storage) but it can also be done in GNU/Linux via software and no additional hardware is required.

It is important to note that RAID provides significant fault tolerance capability, but available data should always be backed up periodically. This strategy is important, because if more devices than the RAID's capacity fail at the same time, the only way to prevent data loss will be through the recovery of previously saved files.

RAID distinguishes a number of possible levels or configurations (each hardware manufacturer, or specific software can support one or more of these levels). Each RAID level is applied over a set of disks, sometimes referred to as a RAID array, which are usually disks of equal size (or by groups) but it is not an indispensable condition. Hardware driver features should be analyzed as some do not allow disks (or groups) to be of different sizes; in others they can be used, but the array is defined by the smaller disk (or group) size.

Typical RAID levels are (although in some cases the nomenclature may depend on each manufacturer):

- **RAID 0**: data is distributed equally between one or more disks without parity or layered redundancy (striping) information and does not involve fault tolerances. Only data is being distributed; if the disk physically fails, the information is lost and we must retrieve it from backups. What does increase is performance, depending on RAID 0 deployment, as read and write operations will be split between the different drives

- **RAID 1**: an exact copy (mirror) is created on a set of two or more drives. In this case, it is useful for reading performance (which can increase linearly with the number of disks), and especially for having tolerance for the failure of n-1 disks in the array. RAID 1 is often suitable for high

availability, such as 24x7 environments, where access to disk resources is critical. In hardware controllers it is common to support hot swapping of the disks allowing, once the fault is detected, to replace the affected disk with a new one without shutting down the system and for the controller to automatically synchronize the data to restore the initial operational state.

- **RAID 2, 3, and 4**: they use different divisions, groups, and redundancy codes but they are not typically used. RAID 5 and RAID 6 are used instead, because they have better features and/or capabilities.

- **RAID 5**: splitting is used at the block level, distributing parity across all disks. It is widely used, due to the simple parity scheme, and because this calculation is easily implemented by hardware, with good features. RAID 5 will require a minimum of 3 disks to be deployed and only allows one disk to fail (the failure of a second disk causes complete data loss). The maximum number of disks in a RAID 5 redundancy group is theoretically unlimited, but in practice it is common to limit the number of drives as the drawbacks of using larger redundancy groups are a higher probability of simultaneous failure of two disks, a longer reconstruction time, and a higher probability of finding an unrecoverable sector during a reconstruction. One alternative that provides dual parity protection, thus allowing for more disks per group, is RAID 6.

- **RAID 6**: it was not part of the original levels and it extends the RAID5 level by adding another parity block, so it divides the data at the block level and distributes the two parity blocks between all members of the set. An algorithm based on Reed-Solomon code is used for parity calculation and therefore RAID6 is a special case of this with only two codes (2 parity blocks). RAID 6 provides protection against dual disk failure and failure when a disk is being rebuilt and a minimum of 4 disks are required to deploy it.

- **RAID 5E and RAID 6E**: this is how RAID 5 and RAID 6 variants that include backup drives are referred to. These disks can be connected and ready (hot spare) or on standby (standby spare). In this 'E' classification, backup disks are available for any of the drives and do not result in any performance improvement, but reconstruction time (for hot spare disks) and management tasks are minimized when failures occur. A backup disk is not actually part of the assembly until a disk fails and the assembly is rebuilt over the backup disk.

Nested RAIDs:

- **RAID 0+1 (or 01)**: a stripe mirror is an embedded RAID level. For example, two RAID 0 groups are deployed, which are used in RAID 1 to mirror each other (categories are read from right to left and the last, 1 in this case, is the top one). One advantage is that in the event of a failure, the RAID 0

level used can be rebuilt thanks to the other copy, but if disks are to be added, they must be added to all RAID 0 groups.

- **RAID 10 (1+0)**: mirror splitting, RAID 1 groups under RAID 0. So, in each RAID 1 group, a disk can fail without losing data. However, this will require replacing them, since if not, the disk remaining in the group becomes a possible point of failure of the entire system. It is a configuration commonly used for high performance databases (for its fault tolerance and speed, not based on parity calculations).

- **RAID 50 (or 5+0)**: it combines the block-level splitting of a RAID 0 with the distributed parity of a RAID 5, thus a split RAID 0 set of RAID 5 elements. The configuration of the RAID sets impacts on the overall fault tolerance as, for example, a configuration of three RAID 5 sets of 3 disks each has the highest storage capacity and efficiency, but can only tolerate a maximum of three potential disk failures of each RAID 0 set. Because system reliability depends on the rapid replacement of broken disks so that the assembly can be rebuilt, it is common to build six-disk RAID 5 sets with an online backup disk (hot spare) that allows immediate rebuilding to begin in the event of the failure of the set. RAID 50 improves RAID 5 performance, especially in writing, and provides better fault tolerance than a single RAID level. This level is recommended for applications that require high fault tolerance, capacity, and random search performance.

It is recommended to expand the information before deciding on a particular design and use any of the online RAID calculators to learn about it.

Some considerations to keep in mind about RAID in general:

- RAID improves system uptime, as some of the levels allow the disks to fail and the system to remain consistent, and depending on the hardware, even the problematic hardware can be changed without the need to shut down the system, a particularly important issue in critical systems.

- RAID can improve application performance, especially in systems with mirror deployments, data splitting may allow linear read operations to increase significantly.

- RAID does not protect data; obviously, destruction by other means (virus, general malfunction, or natural disasters) is unprotected and it is vital to have backup copies.

- Data recovery is not simplified; if a disk belongs to a RAID array, it has to be recovered in that environment and data cannot be accessed individually on each disk.

- It usually does not improve typical user applications (e.g., desktop applications) because these applications have random access components to data, or to small datasets; they may not benefit from linear readings or sustained data transfers.

- Some schemes speed up reading operations, but on the other hand, they penalize writing operations (RAID 5 case for parity calculation to write). If the use is basically writing, we will need to look for which schemes do not penalize them (some cases, such as RAID 0.1, or some RAID 10 modalities are equivalent to writing on a single disk, or even increase performance).

- Information transfer is not facilitated; it's quite easy to move data without RAID, by simply moving the drive from one system to another. In the case of RAID, it is almost impossible (unless the same hardware controller is available) to move one disk array to another system.

For GNU/Linux, RAID hardware is supported by multiple kernel modules, associated with different sets of manufacturers. This allows the system to be able to abstract itself from hardware mechanisms and make them transparent to the system and to the end user. These kernel modules allow access to the details of these controllers, and their configuration of very low-level parameters, which in some cases (especially on servers that support high load) to adapt (tuning) the disk system to the needs of the service.

As mentioned above, it is possible in GNU/Linux to create a RAID software through a controller integrated into the kernel, called Multiple Device (md), that can be considered as the kernel level support for RAID. This can deploy RAID levels, typically 0,1,5,6 and nested (e.g., RAID 10), on different block devices such as SATA or SCSI drives. This module also supports a mode of operation called linear with two or more disks which are combined into a physical device. Disks are "added" to each other, so writing linearly on the RAID device will fill disk 0 first, then disk 1 and so on. Disks do not have to be the same size. There is no redundancy at this level and if a disk fails, all data will be lost.

The command used to create and manage RAIDs in Gnu/Linux is called `mdadm` and below are some examples regarding SCSI disks:

Creating a linear array (striping on consecutive disks):

```
mdadm --create --verbose /dev/md0 --level=linear --raid-devices=2 /dev/sda1 /dev/sdb1
```

where a linear array is generated from the */dev/sda1* and */dev/sdb1* partitions, creating the new */dev/md0* device on which the archiving system can already be created and mounted:

```
mkfs.ext4 /dev/md0; mkdir /mnt/linearRAID
```

```
mount /dev/md0 /mnt/linearRAID
```

For a RAID0 or RAID1 we simply change the level to *raid0* or *raid1*. With `mdadm --detail /dev/md0` we can check the parameters of the newly created array and we can check the input in */proc/mdstat* to determine the active arrays, as well as their parameters. `mdadm` has options to examine and manage the different RAID arrays software created (see `man mdadm`) and it should be noted that RAID arrays by software are not automatically recognized at boot time (as with RAID hardware), as they actually depend on the construction with `mdadm`. For the definition of an array software to be persistent, it must be recorded in the configuration file */etc/mdadm/mdadm.conf* (location may depend on distribution) through:

```
mdadm --detail --scan >> /etc/mdadm/mdadm.conf
```

To configure a RAID5 with three SATA disks and one partition per disk, we must:

- Prepare the disks: with `fdisk` creating an '**fd**' type partition (Linux Raid autodetect) over each one.

- Create the RAID 5 called /dev/md0:
  `mdadm --create --level=5 [--chunk=1024] --raid-devices=3 /dev/md0 /dev/sdb1 /dev/sdc1 /dev/sdd1` (the chunk parameter has special relevance in RAID performance and sizes from 512 onwards are recommended depending on average file size - see manual page).

- Create the file system: `mkfs.ext4 /dev/md0`

- To be activated on the following boot:
  `mdadm --detail --scan >> /etc/mdadm/mdadm.conf`

- For mounting: `mkdir /mnt/RAID5; mount -t ext4 /dev/md0 /mnt/RAID5`

Other useful orders:

- To assemble an array (it collects the components of a previously created array into an active array):
  `mdadm --assemble /dev/md0 /dev/sdb1 /dev/sdc1 /dev/sdd1`

- To add a new drive: `mdadm --add /dev/md0 /dev/sde1` (this drive will remain as a spare), `mdadm --grow /dev/md0 --raid-devices=4` (to move the spare drive to active) and then the file system must be resized with `resize2fs /dev/md0 size`

- Mark a disk with errors and remove it: `mdadm --fail /dev/md0 /dev/sde1; mdadm --remove /dev/md0 /dev/sde1`

- Stop/Start an array (remember to unmount/mount the filesystem before/after stopping/booting the array): `mdadm --stop|run /dev/md0`

- Obtain array information: `mdadm --detail /dev/md0`

- Monitor the array (be careful with this command as it can be a massive source of emails if it is not configured correctly): `mdadm --monitor --scan --mail=[email address] --delay=1800 &`

- If it is desired to be mounted during the booting, it must be included in the */etc/fstab* file.

In relation to the operation of the array, the information of */proc/mdstat* can be examined or its status can be checked by the email or `mdadm --detail /dev/md0`. If any failure occurs in the array, it would go into degraded condition, and the system would lose its capacity to tolerate a subsequent failure, so it will be necessary to remove the disk and replace it (if there are no spare disks) and the system would automatically start the reconstruction.

9. Logical volume disks: LVM

LVM (Logical Volume Manager) is an environment that allows to be abstracted from the physical disk system, its configuration and number of devices/space, so that the (operational) system can handle this work through logical volumes with a simpler and more effective view. LVM was based on ideas developed from storage volume managers used in HP-UX and Gnu/Linux implements version 2 called **lvm2**.

LVM architecture typically consists of:

- **Physical volumes** (PVs): they are the hard drives, or partitions of them, or any other element that appears as a hard drive on the operating system (e.g., a RAID software or hardware).

- **Logical volumes** (LVs): these are visible in the system as a block device (absolutely equivalent to a physical partition), and can contain a file system (e.g., users' home) and can be given any desired name, facilitating administration tasks.

- **Volume groups** (VGs): it is the administrative unit that encompasses resources (logical and physical volumes). This unit saves the available PV data, and how LVs are formed from PVs. Obviously, to be able to use a VG group, physical PV media must be available, which will be organized into different LV logic units.

Using logical volumes, a more flexible treatment of storage system space (which could have a large number of disks and partitions) is allowed, depending on the needs. The space may be managed, both by more appropriate identifiers and by operations that allow adapting the needs to the space available at any given time. That is, LVM allows:

1) To dynamically resize logical groups and volumes, taking advantage of new PVs, or extracting some of the initially available ones.

2) Snapshots of the file system making it easy to create a new device that is a snapshot at the time of a LV situation. We can, for example, create the snapshot, mount it, test various new software operations or configuration, or other items, and if it does not work as expected, return the original volume to its state before testing.

3) Redundancy: it can be used together with RAID to determine the desired degree of redundancy (see `man lvmraid`).

To create an LVM, 4 basic steps must be followed:

- Use fdisk to initialize disks by creating a type **8e** partition (Linux LVM)
- Define and initialize the physical volumes (PV):
  `pvcreate /dev/device [/dev/device]`
- Define the volume groups by grouping PVs into a VG:
  `vgcreate vg-name /dev/device [dev/device]`
- Initialize the logical volumes on each VG (LV):
  `lvcreate –L size –n lv-name vg-name`
- We can expand (or reduce) a volume with `pvcreate /dev/new_device` and then `vgextend vg-name /dev/new_device;` to reduce `vgreduce vg-name /dev/device`

Example for creating, mounting, and unmounting an LVM:

```
apt install lvm2; apt install system-config-lvm
pvcreate /dev/sdd1 /dev/sde1
vgcreate myvol /dev/sdd1 /dev/sde1
vgdisplay
lvcreate -n mylogvol -L 10g myvol
mkfs.ext4 /dev/myvol/mylogvol
mkdir /mnt/lvm-test
mount /dev/myvol/mylogvol /mnt/lvm-test;
df –h
lvdisplay
umount /mnt/lvm-test
lvremove /dev/myvol/mylogvol
```

Some additional commands to those mentioned are: `lvchange` (change of the attributes of a Logical Volume), `lvconvert` (converts a Logical Volume from linear to mirror or snapshot), `lvrename` (it changes the name of a Logical Volume), `lvresize` (resizes a Logical Volume), `lvs` (information about Logical Volumes), `lvscan` (searches all disks for a Logical Volumes).

10. Non-interactive jobs

In administration tasks, it is usually necessary to perform certain tasks at time intervals, either by scheduling and performing the tasks at times of low utilization of the system, or due to the periodic nature of the tasks to be executed.

To perform these types of jobs, there are several options for planning and executing tasks:

- ***nohup***: this is the simplest case used by users as it allows them to perform a non-interactive task and it is maintained even if the user exits the session.

- ***at***: it allows launching an action in a future time by scheduling the instant at which it will start, specifying the time (*hh:mm*) and date, or whether it will be done today or tomorrow.

```
at 10pm task (runs task at 10:00 pm)
```

- ***cron***: it sets a list of tasks to be performed through a configuration in */etc/crontab*. Specifically, each entry in this file will have: minutes | hour | day of the month | month | day of the week | user | task/command.

The content will be similar to:

```
10 3 * * * root task1
20 3 * * 7 root task2
30 3 1 * * root task3
```

where a series of tasks to be done are being scheduled: every day (“ ***** ” indicates “any”), weekly (the seventh day of the week), or monthly (the first of each month). The tasks would be executed by the *crontab* command, but the *cron* system assumes that the machine is always on. If not, it is best to use *anacron*, which checks if the action was not performed when it should have been done, and executes it. In this example, task1 will run every day at 3:10, task2 on the 7^{th} day of each week at 3:20, and task3 on the 1st of each month at 3:30.

There may also be some files, *cron.allow, cron.deny*, to limit who can place jobs (or not) in *cron*, and using the `crontab` command, a user can define jobs in the same format as above and which will be saved in */var/spool/cron/crontabs*. In some cases, there is also a */etc/cron.d* directory where jobs can be placed and which is treated as an extension of the */etc/crontab* file. In some versions, the

system already specifies its periodic system jobs directly on subdirectories of the */etc* such as *cron.hourly, cron.daily, cron.weekly and cron.montly*, where the system jobs that need this periodicity are placed.

In systems with `systemd` booting, there is *systemd-cron*, as a service based on timers and calendars, which allows using the standard directories *cron.hourly*, *daily*, *monthly* and with `systemctl start|stop crond.service` the service can be started/stopped.

Depending on the distribution and support of `systemd`, a classic *cron* support layer will be used, or the definition of systemd's timers definition may be used to create tasks only. It is recommended to refer to the distribution man pages, referring to `systemd`, `systemd.timer`, `cron` and `crond` to know the methods enabled.

11. Network management

The GNU/Linux operating system is taken as an example of a standard communications architecture and since its inception has incorporated different protocols and network-based services to improve its interconnectivity and versatility. Many changes have occurred in recent years with the rise of Internet-connected devices (which have grown exponentially and IPv4 device addresses have been exhausted), stemming from the proliferation of smartphones, tablets (but also TVs, video game consoles, cars, etc.), as well as the bandwidth needs that have caused evolutions that will still take a few years to stabilize.

Among them, the most direct one is the change of IP protocol that will be in its version 6 (IPv6) and that will force all systems to be changed to operators and users to support this new connection protocol. The other significant concept in communication environments is the Internet of Things which will soon change society as it is known (by the number and diversity of devices connected to the Internet).

Beyond these challenges, this section will discuss the current TCP/IP-based communication structure and protocols, which is the base protocol used on the Internet.

11.1. The TCP/IP protocol

The TCP/IP protocol synthesizes an example of standardization and a global communication will and it consists of a set of basic protocols that have been added to the main one to meet the different needs in computer-computer communication such as TCP, UDP, IP, ICMP, ARP. [Com].

Nowadays, the most common use of TCP/IP for the user is the remote connection to other computers (SSH), the use of remote files (NFS), or their transfer (HTTP, Hypertext Transfer Protocol) among others. The most important traditional TCP/IP services (and which nowadays still exist or have evolved) are:

a) File transfer: initially the FTP service allowed files to be obtained/sent to another computer. Nowadays it is a little-used protocol. Protocols such as a WebDAV over HTTP have allowed the file transfer to be done more simply and without specific applications beyond a browser and web server.

b) Remote connection (login): initially, the network terminal protocol (`telnet`) that allowed interactive interconnection, but it is in disuse due to its insecurity. The service currently used is called SHH and it encodes the information and makes the packets on the network illegible to an external observer.

c) Mail: this service allows sending messages to users on other computers. This mode of communication has become a vital element in users' lives and it allows emails to be sent to a central server so that they can then be retrieved through specific programs (clients) or read through a web connection.

The advancement of technology and the low cost of computers have allowed certain services can be offered already configured over certain computers working on a client-server model. A server is a system that provides a specific service for the rest of the network. A client is another computer that uses this service. All of these services are generally offered over TCP/IP:

- **Network file systems** (NFS): allows a system to access files on a remote system so that storage devices (or some of them) are exported to the system that wants to access them, and it can "see" them as if they were local devices.

- **Remote printing**: allows access to printers connected to other computers.

- **Remote run**: enables a user to run a program on another computer. There are different ways to perform this execution: either through a command (`ssh`) or through systems with RPC (Remote Procedure Call) that allow a program on a local computer to perform a program function on another computer.

- **Name servers**: in large installations there is a set of data that needs to be centralized to improve its utilization, e.g., user name, keywords, network addresses, etc. All of this makes it easy for a user to have an account for all machines in an organization (for example, a NIS service, -Network Information Service-, or LDAP, -Lightweight Directory Access Protocol-). The DNS (Domain Name System) is another name service but it has a relationship between the machine name and the logical identification of this machine (IP address).

- **Graphic terminal servers** (network-oriented window systems): allows a computer to view graphical information about a display that is connected to another computer. The most common of these systems in GNU/Linux is X Window and it operates through a display manager (dm) or in the case of Windows® systems, terminal services (now known as remote Desktop Services) allow a user to access applications and data stored on another computer via the network.

However, in recent decades, the services offered in TCP/IP have proliferated to meet the needs of both individual users and large facilities services. Among the most important for *nix systems, we can list the following, among others:

- **autofs, amd**. Auto-mounting disks over the network.
- **audit**. Saving remote information for audit purposes.
- **bootparamd/tftp**. Allows the machines without disks to obtain network parameters and the operating system itself.
- **cups**. Network printing service.
- **cvs/subversion**. Concurrent version system.
- **inetd/xinetd**. Network environment (daemon) that centralizes a set of services and filters.
- **imap/pop**. Internet Service Message Access Protocol for remote mail system access.
- **dhcp**. Dynamic Host Configuration provides network parameter information to clients on a subnet.
- **firewall**. Packet Filtering firewall used to manage and filter all TCP/IP packets on the Linux kernel.
- **heartbeat**. High Availability Services to increase redundancy in critical services.
- **http**. Web service management.
- **ipmi**. Remote machine management via OpenIPMI.
- **ipsec**, **kerberos**, **ssl/ssh**. Protocols/services for coded communications and authentication.
- **iscsi**. Management and access to disks via the SCSI protocol over a network.
- **ldap**. Lightweight Directory Access Protocol, information access services on large networks.
- **named**. Domain Name System, name/domain service vs. IP.
- **netdump**. Sending network information for kernel error diagnosis.
- **netfs/nfs/smb**. Network disk mount: Network File System (NFS), Windows (SMB).
- **ntalk**. Chat service.
- **ntpd**. Time synchronization services.
- **proxy**. Proxy Caching Server services for http.
- **rsync**. Remote Synchronization for file backup services.
- **rtp**: streaming audio/video.
- **snmpd**. Simple Network Management Protocol to monitor, management & monitoring of devices connected to a network.
- **sql**. Network database service.
- **vncserver**. Service for Virtual Network Computing and using Remote Desktop Sharing.
- **voip.** Voice over IP.
- **ypbind/ypserv**. NIS information services on GNU/Linux systems.
- **zeroconf DNS**. Network information services when a DNS service does not exist.

A list of TCP/IP services can be found in the list of services:ports (standard communication points) mapping in the */etc/services* file of any GNU/Linux distribution.

TCP/IP are actually two communication protocols between separate computers. On the one hand, TCP (transmission control protocol), defines the communication rules so that a computer (host) can 'talk' to another one and on the other hand, IP (internet protocol) defines the protocol that allows identifying the networks and establishing the paths between the different computers. That is, it routes data between two computers across networks.

An alternative to TCP (connection-oriented protocol in which the sender and receiver must be simultaneously connected) is the UDP protocol, which treats the data as a message (datagram) and works offline (the receiver does not have to be connected when the message is sent) but communication is not reliable since the packets may not arrive or arrive duplicated or with errors.

There is another alternative protocol called ICMP that is used for error/control messages or to extract information about a network.

In the OSI/ISO communications model, it is a 7-layer theoretical model adopted by many networks in which each has an interface to communicate with the previous and subsequent ones. Although the OSI model is the reference model, it is often preferred to use the TCP/IP or Internet model that has four layers of abstraction (as defined in RFC 1122) and it is often compared to the seven-layer OSI model to establish equivalences. The TCP/IP model and related protocols are maintained by the Internet Engineering Task Force (IETF) and the layered subdivision they perform is:

- **Layer 4 or application layer** (similar to OSI Model Layers 5 (session), 6 (presentation), and 7 (application): this layer handles high-level protocols and presentation/coding and dialogue aspects.

- **Layer 3 or transport layer** (similar to layer 4 (transport) of the OSI model): linked to protocols and quality of service parameters in relation to reliability and error control. That is why it defines two Transmission Control Protocols (TCPs), communication-oriented and error-free, and User Datagram Protocol (UDP), which is connectionless and therefore there may be duplicate or out-of-order packets.

- **Layer 2 or Internet layer** (similar to layer 3 (Network) of the OSI Model): it sends source packets from any network and makes them reach their destination, regardless of the route chosen and the networks they must travel. The specific protocol of this layer is called the Internet Protocol (IP), which must decide the best route and suitability of the packets to achieve the desired objective.

- **Layer 1 or media access layer** (similar to layer 2 (data connection or link) and layer 1 (physical) of the OSI model): it addresses all aspects that an IP packet requires to actually perform a physical link and includes the technology details.

In summary, TCP/IP is a family of protocols (including IP, TCP, UDP) that provide a set of low-level functions used by most applications. As mentioned above, there is a new version of the IPv6 protocol, which replaces the IPv4 and which significantly improves the previous one in topics, such as a greater number of nodes, traffic control, security or improvements in routing aspects.

11.2. Network physical devices (hardware)

From the physical point of view (OSI model layer 1), the most widely used hardware for local networks (LAN, local area network) is known as Ethernet. Its advantages are its low cost, acceptable speeds (from 10 to 10,000 Megabit/s - that is, 10 Gbit per second in the latter case), and ease of installation. Although there were different ways of connecting, the ones used nowadays are twisted pair and optical fibre.

Ethernet technology uses communication intermediates (hubs, switches, routers) to configure multiple network segments and divide traffic to improve information transfer capabilities. Typically, in large institutions these LANs are interconnected via optical fibre. For domestic or small business connections, the network technology that has been widely broadcast is through connections via a copper telephone cable (ADSL Asymmetric Digital Subscriber Line) or lately, via optical fibre (FTTH Fibre to the Home), where in any case a device (router) that converts these protocols to Ethernet (or wireless Wi-Fi protocols) is necessary.

Speed is one of the selection parameters between one type of technology or another and the cost is beginning to be affordable for private connections, especially due to the incorporation of broadband services, such as digital TV or Video on Demand, to the usual ones. On ASDL we find bandwidth options from 12Mbit/s to 50Mbit/s, depending on the location of the central unit and the client, and on fibre we can find speeds between 100Mbit/s and 500Mbit/s. It should be noted that the values given on ADSL are generally for download, while for upload they are usually approximately 1/10 of the download value, whereas in fibre it is generally symmetrical (same upload speed as download speed).

In GNU/Linux, network interface card (NIC) controllers are included in the kernel or can be loaded as modules. The */sys/class/net* directory will provide all identified network devices that can be viewed via `ip a`, but network devices can also be located via `lspci` or `hwinfo` by locating hardware devices.

We can also view configured devices and all (dynamic) status and configuration information on */proc/net* and view all instant packet information sent and received and their status for each interface on */proc/net/dev*.

11.3. General concepts about networks

Some general networking concepts will be summarized below:

- **Internet/Intranet**: the term intranet refers to the application of Internet technologies within an organization, basically to distribute and make information available within the company. These services are usually mounted on private IP addresses (it will be seen later), so these machines are not recognized from the Internet and their output to the Internet is through a router (in most cases the same thing goes for machines in a home environment). If any of these services need to be accessed from outside the institution, proxy services or routers that can redirect packets to the internal server should be used.

- **Node**: a node (host) is a machine that connects to the network (in a broad sense, a node can be a computer, tablet, phone, printer, disk cabinets, etc.), i.e., an active and distinguishable element in the network that claims or provides some service and/or shares information.

- **Ethernet address** (or MAC address): a 48-bit number (e.g., 00:88:40:73:AB:FF – in octal code) that is located on the physical device (hardware) of the Ethernet network controller (NIC) and is recorded by the Ethernet network manufacturer (this number must be unique in the world, so each NIC manufacturer has a pre-assigned range). It is used in layer 2 of the OSI model and it is possible to have 2^{48} or 281.474.976.710.656 potential MAC addresses.

- **Host name**: Each node must also have a unique name on the network. They can be just names or use a hierarchical domain naming scheme. Names must be a maximum of 32 characters between a-zA-Z0-9.-, and contain no spaces or #, starting with an alphabetic character.

- **Internet address** (IP address): it is composed of a set of numbers and it will depend on the IP protocol version, and it is used universally to identify computers over a network or Internet. For version 4 (IPV4) it consists of four numbers in the range 0-255 (32 bits), separated by dots (e.g., 192.168.0.1) which enables 4.294.967.296 (2^{32}) different host addresses, which has shown to be insufficient and especially because each individual has more than one computer, tablet, phone, etc. For version 6 (IPv6) the address is 128 bits and is grouped into four hexadecimal digits forming 8

groups, for example fe80:0db8:85a3:08d3:1319:7b2e:0470:0534 is a valid IPv6 address. A four-digit group can be compressed if it is null (0000).
For example:
fe80:0db8:0000:08d3:1319:7b2e:0470:0534=fe80:0db8::08d3:1319:7b2e:0470:0534.
Following this rule, if two consecutive groups are null, they can be grouped together for example: fe80:0db8:0000:0000:0000:0000:0470:0534=fe80:0db8:::0470:0534. Care should be taken as fe80:0000:0000:0000:1319:0000:0000:0534 cannot be summarized as fe80::1319::0534 as the number of null groups on each side is unknown. The first zeros of each group may also be omitted with fe80:0db8::0470:0534 remaining as fe80:db8::470:534. In IPv6, therefore, 340.282.366.920.938.463.463.374.607.431.768.211.456 addresses are supported (2^{128} or 340 sixtillions of addresses), which demonstrate a large enough number to not have the limitations of IPv4 for a long time. Name translation into IP addresses is performed by a DNS server (Domain Name System) that transforms node names (human readable) into IP addresses (`named` or `dnsmasq` are services that perform this conversion in GNU/Linux).

- **Port**: a numeric identifier of a connection point (similar to a mailbox) on a node that allows a message (TCP, UDP) to be read by a particular application within this node (for example, two machines communicating by ssh will communicate via port 22, but these same machines may have http communication by port 80). Different applications can be made by communicating between two nodes through different ports simultaneously.

- **Router node** (gateway): is a node that performs routing (data transfer routing). A router, depending on its characteristics, may transfer information between two networks of similar or different protocols and may also be selective.

- **Domain name system** (DNS): it allows securing a single name and facilitating the administration of databases that carry out the translation between name and Internet address and are structured in the form of a tree. For this purpose, domains separated by points are specified, of which the highest (from right to left) describes a category, institution or country (COM, commercial, EDU, education, GOV, government, MIL, military (government), ORG, non-profit organizations, XX two letters per country, or in special cases three letters CAT, Catalan language and culture, among others). The second level represents the organization, and the third and others, departments, sections, or divisions within an organization (e.g., www.uoc.edu or admin@remix.nteum.cat). The first two names (from right to left, uoc.edu in the first case, nteum.cat (in the second) must be assigned (approved) by the ICANN (Internet Corporation for Assigned Names and Numbers, worldwide name management body on the Internet) and the remaining ones can be configured/assigned

by the institution. A rule that governs these names is the FQDN (Fully Qualified Domain Name) that includes a computer name and the domain name associated with that computer. For example, if we have a host named 'remix' and the domain is 'nteum.cat', the FQDN will be 'remix.nteum.cat'. On some networks where DNS is not available or accessible, the */etc/hosts* file (which must be on all devices in the network) can be used and will meet the same objective, or the mDNS service (e.g., `avahi`) that implements the so called zero-configuration networking (*zeroconf*) for DNS/DNS-SD multicast configuration can be installed. This service allows programs to publish and discover services and hosts over a local network. In an opening policy, the ICANN has regulated the request/inclusion of new generic higher order domains (called gTLDs) oriented by sectors, for example, cities (.barcelona, .miami...), food (.beer, .coffee...), geocultural, music and art, education, computer science, etc.

- **DHCP, bootp**: these services are high-level protocols that allow a client node to obtain network information (such as node IP address, mask, gateway, DNS, etc.). It is used in a home environment so that devices connected to a Wi-Fi/LAN, for example, can receive the parameters from the internal network and then connect to the Internet. This mechanism is also used by large machine organizations to facilitate management on large networks or where there are a large number of mobile users.

- **ARP, RARP**: in networks (such as Ethernet), IP addresses are automatically discovered through two protocols: ARP and RARP. ARP uses messages (broadcast messages) to determine the Ethernet address (OSI model layer 3 MAC specification) corresponding to a particular network address (IP). RARP uses broadcast messages (message that reaches all nodes) to determine the network address associated with a particular hardware address.

- **Socket library**: In UNIX, the entire TCP/IP deployment is part of the operating system kernel and the way to use them by a programmer is through the API (Application Programming Interfaces) that the operating system deploys. For TCP/IP, the most common API is the Berkeley Socket Library (Windows uses an equivalent library called Winsocks). This library allows to create a communication point (socket), associate it with an address of a remote node/port (bind), and offer the communication service (for example, via *connect*, *listen*, *accept, send, sendto, recv, recvfrom*).

11.4. Assigning an Internet address

Internet IP addresses were originally assigned by the Internet Network Information Center (government Internet agency responsible for domain names and IP addresses until 18/09/1998). It is currently managed by the

Internet Corporation for Assigned Names and Numbers (ICANN). In the following sections, the characteristics of the IPv4 protocol and the IPv6 protocol will be shown.

11.4.1. IPv4

The IP address in IPv4 has two fields: the left one represents the network ID and the right one represents the node ID. With this in mind, the 4 numbers between 0-255, or four bytes, each byte is either part of the network or the node. The network part is assigned by the ICANN and the node part is assigned by the institution or provider.

There are some constraints: **0** (for example, 0.0.0.0) in the network field is reserved for default routing and **127** (for example, 127.0.0.1) is reserved for self-referral (local loopback or local host), **0** in the node part refers to this network (for example, 192.168.0.0) and **255** is reserved for broadcast (the same packet is sent to all host in the network) (for example, 198.162.255.255).

Different mappings can have different types of networks or addresses:

- **Class A** (*network.host.host.host*): 1.0.0.1 to 126.254.254.254 (126 networks, 16 million nodes) define large networks.

- **Class B** (*network.network.host.host*): 128.1.0.1 to 191.255.254.254 (16K networks, 65K nodes) the first node byte is typically used to identify subnets within an institution).

- **Class C** (*network.network.network.host*): 192.1.1.1 to 223.255.255.254 (2 million network bits, 254 nodes).

- **Class D and E** (*network.network.network.host*): 224.1.1.1 to 255.255.255.254 reserved for multicast (from a node to a set of nodes that are part of a group) and experimental purposes.

Some address ranges have been reserved so that they do not correspond to public networks. These addresses belong to private networks and messages will not be routed over the Internet, which is commonly known as the Intranet. These are:

- **Class A**: 10.0.0.0 up to 10.255.255.255

- **Class B**: 172.16.0.0 up to 172.31.0.0

- **Class C**: 192.168.0.0 up to 192.168.255.0

A concept associated with an IP address is the mask, which will then allow defining subnets and automatically route packets between these subnets. To do this, it is necessary to define a network mask that will be the significant bits of the subnet and that will allow to define whether two IPs are within the same network or not. For example, in a given static IPv4 address (e.g., 192.168.1.30), the network mask 255.255.255.0 (i.e., 11111111111111111111111100000000 in binary representation) indicates that the first 24 bits of the IP address correspond to the network address and the other 8 are machine-specific.

In IPv6, and since they are 128 bits, only the number of 1s (notation that is also used in IPv4) will be expressed, to facilitate its reading. In the example above, for IPv4 it would be 24 and generally would be 192.168.1.30/24 and in IPv6, for example, for the address fe80:0db8::0470:0534 the mask could be expressed as fe80:0db8:::0470:0534/96, where it indicates that for this address, the first 96 bits correspond to the network.

The broadcast address is special, as each node on a network listens to all messages (in addition to its own address). This address allows datagrams, usually routing information and warning messages, to be sent to a network and read by all nodes in the same network segment. For example, when ARP seeks to find the Ethernet address corresponding to an IP, it uses a broadcast message, which is sent to all machines on the network simultaneously. Each node on the network reads this message and compares the IP being searched to its own, and returns a message to the node that asked the question if it matches. For example, on a 192.168.1.0/24 network, the broadcast address is 192.168.1.255.

11.4.2. IPv6

IPv6 address types can be identified by taking into account the ranges defined by the first few bits of each address (the value specified after the bar is the mask equivalent to the number of bits not considered for that address):

- **::/128**. The all-zero address is used to indicate the absence of an address, and no node is assigned.

- **::1/128**. It represents the *loopback* address that a node can use to send packets to itself (corresponds to 127.0.0.1 of IPv4). It cannot be assigned to any physical interface.

- **::1.2.3.4/96**. The supported IPv4 address is used as a transition mechanism on dual IPv4/IPv6 networks (hardly used).

- **::ffff:0:0/96**. The mapped IPv4 address is used as a transition mechanism.

- **fe80::/10**. The local link prefix specifies that the address is only valid on the local physical link.

- **fec0::**. The local prefix (site-local prefix) specifies that the address is only valid within a local organization. RFC 3879 has made it obsolete and it must be replaced by Local IPv6 Unicast addresses.

- **ff00::/8**. The multicast prefix used for this type of address.

It should be noted that broadcast addresses do not exist in IPv6, and functionality can be emulated using the multicast FF01::1/128 address.

If the address is a built-in IPv4 address, the last 32 bits can be written in decimal base as ::ffff:192.168.1.1 or ::ffff:c0a8:0101, not to be confused with ::192.168.89.9 or ::c0a8:0101. The format ::ffff:1.2.3.4 is called the mapped IPv4 address, and the format ::1.2.3.4, is called the compatible IPv4 address.

IPv4 addresses can be easily transformed into the IPv6 format. For example, if the IPv4 decimal address is 158.109.64.1 (in hexadecimal, 0x9e6d4001), it can be converted as 0000:0000:0000:0000:0000:0000:9e6d:4001 or ::9e6d:4001. In this case, we can use the IPv4 mixed supported notation which would be ::158.109.64.1. This type of supported IPv4 address is hardly being used, although the standards have not made it obsolete.

When we want to identify a range of addresses that can be made by the first bits, this number of bits is added after the bar character"/". For example, fe80:0db8::0674:9966/96 would be equivalent to fe80:0db8:: and also to fe80:0db8::0470:0534/96

IPv6 addresses are represented in DNS using AAAA records (also called quad-A records, as they are four times the length of A records for IPv4) specified by RFC 3363 (there is another view called A6, but while it is more generic, it is also more complex and can further complicate the transition between ipv4 and ipv6).

There is a smooth transition from IPV4 to IPV6 that can be consulted on Google with large disparities between countries including the EC (for example, France with a 71% implementation or Spain with a 3%). While there are a number of mechanisms that will allow for coexistence and progressive migration, both of the networks and user equipment (such as double stack, tunnels, or translation), the major change in adoption is by large network operators and is linked to political decisions.

11.5. Subnets and routing

Two complementary concepts to those described above are **subnets** and **routing** among them. Subnets means subdividing the node part into small networks within the same network to improve traffic, for example.

A subnet takes responsibility for sending traffic to certain IP address ranges by extending the same concept of A-B-C class networks, but only applying this redirection on the node part of the IP. The number of bits that are interpreted as the subnet identifier is given by a netmask which is a 32-bit number (same as the IP).

The one who defines which packets are going to one side or another will be the host that meets the router role and will interconnect several network segments/networks with each other. The router is generally known as a gateway and is used as the host that helps reach the outside (e.g., Internet) from the local network. To obtain the subnet identifier, a logical **AND** operation must be performed between the mask and the IP, which will give the IP of the subnet.

For example, let it be an institution that has a class B network with number 172.17.0.0, and its netmask is 255.255.0.0. Internally, this network is made up of small networks (one per floor of the building, for example). Thus, the address range is reassigned on 20 subnets 172.17.1.0 to 172.17.20.0. The point connecting all these subnets (backbone) has its own range/address, for example 172.17.1.0. These subnets share the same network IP, while the third is used to identify each of the subnets within it.

The second concept, *routing*, represents how messages are sent over subnets. For example, there are three departments with Ethernet subnets:

- Sales (subnet 172.17.2.0).
- Clients (subnet 172.17.4.0).
- Human resources (subnet 172.17.6.0).
- Backbone with FFDI (subnet 172.17.1.0).

To route the messages between the computers of the three networks, three gateways will be required that will each have two network interfaces to switch between Ethernet and FFDI. These will be:

- SalesGW IPs:172.17.2.1 and 172.17.1.1
- ClientsGW IPs:172.17.4.1 and 172.17.1.2
- ResourcesHGW IPs:172.17.6.1 and 172.17.1.3

I.e., an IP to the subnet side and another IP to the backbone.

When messages are sent between sales machines, there is no need to exit to the gateway as the TCP/IP protocol will find the machine directly. The problem is when the Sales0 machine wants to send a message to HHRR3. The message must flow through the two respective gateways. When Sales0 “sees” that HHRR3 is in another network, it sends the packet to its SalesGW gateway, which in turn will send it to HRGW and which in turn will send it to HHRR3. The advantage of subnets is obvious, as traffic between all Sales machines, for example, will not affect client or human resource machines (although it means a more complex and expensive approach to designing, and building the network).

IP uses a table for routing packets between different networks and in which there is a default routing associated with the 0.0.0.0 network. All addresses matching this, as none of the 32 bits are required, are sent to the default gateway towards the indicated network. On SalesGW, for example, the table could be:

Table 7

Management	Mask	Gateway	Interface
172.17.1.0	255.255.255.0	-	fddi0
172.17.4.0	255.255.255.0	172.17.1.2	fddi0
172.17.6.0	255.255.255.0	172.17.1.3	fddi0
0.0.0.0	0.0.0.0	172.17.2.1	fddi0
172.17.2.0	255.255.255.0	-	eno1

The '-' means that the machine is directly connected and does not require routing. The procedure to identify whether this is performed or not is carried out through a very simple operation with two logical ANDs (subnet AND mask and origin AND mask) and a comparison between the two results. If they are the same, there is no routing, if they are different, the packet must be sent to the machine defined as gateway in each network so that it can forward the message.

For example, a message from 172.17.2.4 to 172.17.2.6 will mean: 172.17.2.4 AND 255.255.255.0 = 172.17.2.0 and 172.17.2.6 AND 255.255.255.0 = 172.17.2.0. Because the results are the same, there will be no routing and the messages are sent directly.

Instead, if sent from 172.17.2.4 to 172.17.6.6, it can be seen that there will be a routing through 172.17.2.1 with an interface change (cable to fibre) to 172.17.1.1 and from it to 172.17.1.2 with another interface change (fibre to cable) and then to 172.17.6.6.

The default routing will be used when no rule satisfies the match. In the event that two rules match, the one that does so more accurately, that is, the one that has the least zeros, will be used. The `ip route` command can be used to build routing tables, but if more complex rules (or automatic routing) are required, routing protocols such as RIP, EGP, or BGP can be used.

To install a machine over an existing network, it is therefore necessary to have the following information obtained from the network provider or its network administrator (although DNS, if the network allows it, can be used by publics DNS such as Google 8.8.8.8, 8.8.4.4):

- Node IP
- Network mask IP
- Gateway IP (on the same node network)
- DNS IP

If building a network that will never have an Internet connection, it is recommended to maintain an order and use private IPs to avoid administration problems within that network. Next, we will see how the network and node are defined for a private network (care must be taken, as if the machine is connected to the network, it could harm another user/host with this assigned address).

11.6. Interface configuration (NIC)

In this section, we will see in a little more detail what was already mentioned in the Linux introduction module about the configuration of the network (Ethernet) and the Wi-Fi network.

Once the GNU/Linux kernel is loaded, it runs the `systemd` daemon (equivalent to the `init` process in previous versions) and that according to the configuration in */etc/systemd* (or in */usr/lib/systemd*) will configure the system and will allow the configuration of the network between the services.

Network devices are automatically created when the corresponding hardware is booted and in current versions, the devices are named by their position on the bus and/or MAC (see `man systemd.net-naming-scheme` for more information) assigning 2 letters for each device, for example **en** (Ethernet) **ib** (InfiniBand), **wl** (Wireless local area network, WLAN), **ww** (Wireless wide area network, WWAN). Active devices can be viewed with `ip link` or `networkctl`.

From this point on, the network interface can be configured, which involves two steps: assigning the network address to the device and initializing the network parameters to the system. The command used for this is the `ip`. An example would be:

```
ip addr add 192.168.110.23/24 dev eno1
```

This indicates configuring the device eno1 (Ethernet) with IP address 192.168.110.23 and network mask 255.255.255.0. To delete the assigned IP, we can run `ip addr del 192.168.110.23/24 dev eno1`. To enable the interface, we can run `ip link set eno1 up` and to disable `ip link set eno1 down`.

In the Debian branch, the `ifdown/ifup` commands can be used, although the corresponding package (`apt install ifupdown`) must be installed, which allows the devices to be easily activated and deactivated independently by using the */etc/network/interfaces* file to obtain all the necessary parameters (see `man interfaces` for its syntax).

In GNU/Linux there are different ways to configure the network so that the administrator should not have to enter the configuration parameters in each boot. One of the simplest ways is through the commands mentioned above (`ifup`, `ifdown`) that take their information from the */etc/network/interfaces* file. To modify the network parameters of the eno1 interface, we can do the following:

- **`ifdown eno1`**. Stops all network services. We can also run `systemctl stop networking`.

- `vi /etc/network/interfaces` (or preferred editor tool, but `vi` normally, is available on all *nix systems). It allows editing and modifying the corresponding parameters.

- **`ifup eno1`**. This command allows to start the network services on eno1 (or `systemctl networking start`).

Consider that we want to configure over Debian an eno1 interface that has a fixed IP address of 192.168.0.123 and with 192.168.0.1 as a gateway. We edit */etc/network/interfaces* to include a section such as:

```
auto eno1
iface eno1 inet static
   address 192.168.0.123
   netmask 255.255.255.0
   gateway 192.168.0.1
```

If it has the `resolvconf` package installed (in some distributions, such as Ubuntu, it is installed by default) it can add lines to specify DNS information. For example:

```
auto eno1
iface eno1 inet static
```

```
address 192.168.0.123
netmask 255.255.255.0
gateway 192.168.0.1
dns-search nteum.cat
dns-nameservers 8.8.8.8 8.8.4.4
```

Instead of `auto eno1`, we can also use, for example, the `allow-hotplug eno1` sentence, which indicates the interface that can be activated with `ifup --allow=hotplug eno1`. Lines starting with `allow-` are used to identify interfaces that could be activated by different subsystems (`allow-auto` and `auto` are synonyms).

Remember that if the machine has multiple network interfaces, the previous section can be repeated with the corresponding device, but the last three lines (*gateway*, *dns-search* and *dns-nameservers*) should only be there **once**, as they are common for all interfaces.

After activating the interface, the arguments of the *dns-search* and *dns-nameservers* options are included in */etc/resolv.conf*. The nteum.cat argument of the *dns-search* option corresponds to the *resolv.conf* search option argument and the 8.8.8.8 and 8.8.4.4 arguments for the *dns-nameservers* option correspond to the name-server options arguments at *resolv.conf*. If the *resolvconf* package is not installed, the */etc/resolv.conf* file can be manually modified (and if the */etc/network/interfaces* is installed and not used to configure DNS, the files found in */etc/resolvconf.d* can be modified).

It should be noted that distributions with `systemd` include the *systemd-resolved* service (see `man systemd-resolved.service`) that provides network name resolution for local applications. In addition to calls to glibc native APIs, *systemd-resolved* listens to local DNS requests at IP address 127.0.0.53. DNS servers are determined from the global configuration at */etc/systemd/resolved.conf* (see `man resolved.conf`), and the `resolvectl` command can be used to resolve domain names, IPv4 and IPv6 addresses, DNS resource records, and services from *systemd-resolved.service*. This command can be used to resolve domains or IPs (e.g., `resolvectl query uoc.edu`) including the protocol used for the search, on the network/interface on which the request has been resolved, and whether or not the request is authenticated.

To improve compatibility, *systemd-resolved.service* checks */etc/resolv.conf* to find out if DNS servers are configured by this method (but only if it is not a symbolic link to */run/systemd/resolve/stub-resolv.conf*, */usr/lib/systemd/resolv.conf* or */run/systemd/resolve/resolv.conf*).

Expand with the manual sheet how local addresses (with *localhost* domain or *localhost.localdomain*) and those indicated in */etc/hosts* are resolved.

Returning to the network configuration, an equivalent configuration by DHCP (i.e., a DHCP server that will give the network configuration parameters) is simplified to:

```
auto eno1
iface eno1 inet dhcp
```

The ip command (from the *iproute2* package and replacing the traditional `ifconfig`) allows configuring devices, establishing tunnels, *routing* rules, etc. Below are a series of examples for network configuration for the ip command:

- `ip addr addr add 192.168.1.1 dev eno1`: it defines an IP to eno1.
- `ip addr show`: it displays the settings.
- `ip addr del 192.168.1.1/24 dev eno1`: it removes IP from eno1.
- `ip route add default via 192.168.0.1`: it adds a gateway.
- `ip link set eno1 up`: it activates interface.
- `ip link set eno1 down`: it disables interface.
- `ip route show`: it displays routing.
- `ip route add 10.10.20.0/24 via 192.168.50.100 dev eno1`: it adds a rule.
- `ip route del 10.10.20.0/24`: it deletes a rule.
- `post-up ip route add 10.10.20.0/24 via 192.168.1.1 dev eno1`: it is added to define a static rule in */etc/network/interfaces*.

Another way to configure the network (recommended for users with mobility and standard configurations) is through the **Network Manager** (NM) package. This package consists of a graphical interface (**nm-connection-editor**) for the configuration of network devices (and it can coexist with the configurations in */etc/network/interfaces*) or can be configured through the files, taking into account that the interfaces we want the NM to manage in */etc/network/interfaces* must be disabled. NM will **not** manage interfaces defined in */etc/network/interfaces* as long as */etc/NetworkManager/NetworkManager.conf* contains:

```
[main]
plugins=ifupdown,keyfile
[ifupdown]
managed=false
```

We must change managed=true if we want NM to manage the interfaces defined in */etc/network/interfaces*. Whenever the *NetworkManager.conf* file is modified, the values updated with `mncli general reload` (for more details of the configuration see `man NetworkManager.conf` or *https://wiki.gnome.org/Projects/NetworkManager/SystemSettings*). In some situations, the NM may generate conflicts with some network devices that have been previously configured with the NM and then configuration is desired via /

etc/network/interfaces, so it is recommended to uninstall the NM or delete the configuration files from the corresponding interface from the */etc/NetworkManager/system-connections/* directory.

With `systemd` adoption, it is possible to use a service called *systemd-networkd* to manage network devices. This service, when active, detects and configures network devices as they are discovered and also creates virtual network devices. Configuration files are read from files located in the */lib/systemd/network* network directory, the */run/systemd/network* volatile network directory, and the */etc/systemd/network* local management network directory (see *systemd.network(5)* and for low-level link settings, see *systemd.link(5))*. Interaction with *systemd-networkd* can be done with the `networkctl` command.

Some distributions (such as Ubuntu) may use a new network configuration environment called Netplan that addresses NM compatibility issues and deficiencies. Netplan allows to easily configure the network on a Linux system via a YAML syntax file for the required network interfaces in */etc/netplan/*.yaml*. From this description, Netplan will generate all necessary settings for the network manager used (*NetworkManager* or *Systemd-networkd*).

11.7. Advanced network configuration

It is necessary in GNU/Linux to differentiate between a physical interface and a logical interface. A physical interface is what has so far been called an interface (for example, **eno1**) and a logical interface is a set of values (sometimes called profiles) that can be assigned to the variable parameters of a physical interface. *Iface* definitions in */etc/network/interfaces* are actually definitions of logical interfaces not of physical interfaces, but unless the opposite is indicated, a physical interface will be configured, by default, as a logical interface.

However, if we have a laptop computer that is used at different sites (e.g., home and work) and we need different configurations for each site network, we can use the logical interface definitions. Two logical interfaces, such as home and work (instead of **eno1** as previously done), must be defined first:

```
iface house inet static
   address 192.168.1.30
   netmask 255.255.255.0
   gateway 192.168.1.1

iface work inet static
   address 158,.109.65.66
   netmask 255.255.240.0
   gateway 158.109.64.1
```

In this way, the physical **eno1** interface can be activated for the home network with `ifup eno1=home` and to reconfigure it for the work with `ifdown eno1; ifup eno1=work`. The mechanism is very powerful and can be expanded through configuration based on a series of conditions, using a mapping section. The syntax of a mapping section is as follows:

```
mapping pattern
   script name_script
   [map script]
```

The script called in the mapping section will be executed with the name of the physical interface as argument and with the content of all map lines in the section. Before finalizing, the script will display the result of the transformation by the standard output. For example, the following mapping section will cause `ifup` to activate the eno1 interface as the logical home interface:

```
mapping eno1
      script /usr/local/sbin/echo-home
where /usr/local/sbin/echo-house is:
     #!/bin/sh
     echo home
```

This can be helpful if we have, for example, two different network cards (one for home and one for work). If the *ifupdown* package is installed, the */usr/share/doc/ifupdown/examples/* directory contains a transformation script that can be used to select a logical interface, based on the MAC (Media Access Controller) address. The script must first be installed in an appropriate directory with:

```
install -m770 /usr/share/doc/ifupdown/examples/get-mac-address.sh /usr/local/sbin/
```

We can then add a section like the following in the file

```
/etc/network/interfaces:
   mapping eno1
      script /usr/local/sbin/get-mac-address.sh
      map 02:23:45:3C:45:3C house
      map 00:A3:03:63:26:93 work
```

See other more sophisticated configuration scripts, such as **guessnet**, among others.

11.7.1. Network Configuration on IPv6

In relation to IPv6 configuration, GNU/Linux systems incorporate this functionality through their implementation in the kernel or through modules (in Debian they are included in the kernel and some specific architectures

through a module called ipv6). Basic tools such as `ping` and `traceroute` have their IPv6, `ping6` and `traceroute6` equivalents. At the basic level, with the */etc/network/interfaces* file, an IPv6 network is configured similarly to IPv4 (it must be verified in advance that the available router supports IPv6 that relays data to the global IPv6 network):

```
iface eno1 inet6 static
   address fe80:0db8::0470:0534
   netmask 64
   # Disable autoconfiguration #
   autoconf 0
   # Gateway is automatically configured
   # if it had to be configured
   # gateway fe80:0db8:1234:5::1
```

IPv6 subnets typically have a 64-bit network mask, which means there are 264 different addresses within the subnet and allows a SLAAC (Stateless Address Autoconfiguration) method to select an address based on the MAC address of the network interface. By default, if SLAAC is enabled on the network, the kernel will find IPv6 routers automatically and configure network interfaces.

This type of configuration can have privacy consequences, since if we change networks frequently, it would be easy to identify the device on these networks. IPv6 privacy extensions address this issue and will assign additional randomly generated addresses to the interface, change them periodically, and use them for outbound connections, while inbound connections can use SLAAC-generated addresses. An example of this setting is to enable on */etc/network/interfaces*:

```
iface eno1 inet6 auto
   # Prefer assigned addresses
   # randomly for outbound connections.
   privext 2
```

If an IPv6 connection is not available, the alternative method is to use a tunnel over IPv4. It is important that IPv6 tunnel providers conform to RFC 3053 (RFC describing the procedure for requesting the creation of an IPv6 tunnel on a host called Point of Presence or PoP). There are a large number of vendors with own deployments and based on different business objectives. Two of them that can be tested are Hurricane Electric Free IPv6 Tunnel Broker or Router48 among others.

11.8. Network configuration in RHEL (style) and derivatives

Red Hat and its derivatives (Centos, Rocky, Fedora,...) use a different file structure for network configuration: */etc/sysconfig/network*. For example, for static network configuration:

```
NETWORKING=yes
HOSTNAME=my-hostname
         Hostname defined by cmd hostname
FORWARD_IPV4=true
         True for NAT firewall gateways and routers. False for any other case
GATEWAY="XXX.XXX.XXX.YYY"
         Default gateway IP address.
```

For DHCP configuration the gateway line must be removed as it will be assigned by the server. And if NIS is incorporated, a line must be added with the domain server:

```
NISDOMAIN=NIS-DOMAIN
```

To configure the eno1 interface in */etc/sysconfig/network-scripts/ifcfg-eno1* (for example for a private network):

```
DEVICE=eno1
BOOTPROTO=static
BROADCAST=172.16.1.255
IPADDR=172.16.1.2
NETMASK=255.255.255.0
NETWORK=172.16.1.0
ONBOOT= yes     We will activate the network on the boot
```

We can also add:

```
TYPE=Ethernet
HWADDR=08:00:12:34:56:78
GATEWAY=172.16.1.1
IPV6INIT=no
USERCTL=no
PEERDNS=yes
```

Or for a DHCP configuration:

```
DEVICE=eno1
ONBOOT=yes
BOOTPROTO=dhcp
```

To disable DHCP, we must change BOOTPROTO=dhcp to BOOTPROTO=none. Any changes to these files must restart the services (depending on the network manager) with `systemctl restart network|NetworkManager`.

The host name can be changed by using the `hostnamectl set-hostname` command or manually changed in */etc/sysconfig/network* in HOSTNAME=new-name.

11.9. Configuring a Wi-Fi (wireless) network

The basic configuration of a Wi-Fi network has already been discussed in the Introduction to Linux module, so this section will only give some additional aspects about it. The configuration of Wi-Fi interfaces basically uses the **wireless-tools** package (in addition to the **iproute2** package and the `ip` command). This package uses the `iwconfig` command to configure a wireless interface, but it can also be done through the */etc/network/interfaces* (some distributions include the **iw** package, which is equivalent to **iproute2**, and will replace **wireless-tools**; Debian, for example, includes both).

As a use case, it will be shown how to load the drivers of an Intel Pro/Wireless 2200BG card as a general method, but in current kernels these drivers are already included in the kernel, so it is not necessary to perform these previous steps although it serves as an example. Typically, the software that controls the cards is divided into two parts: the software module that will be loaded into the kernel through the `modprobe` command and the firmware, which is the code that will be loaded onto the card and given by the manufacturer (see Intel page for this model or Linux Wireless project https://wireless.wiki.kernel.org/en/users/Drivers/iwlwifi). As we are talking about modules, it is interesting to use the Debian **module-assistant** package, which allows us to easily create and install a module (another option would be to install the fonts and create the corresponding module). The software (found on the manufacturer's page and referred to as ipw2200) will be compiled and installed by using the module-assistant `m-a` command package:

```
aptget install module-assistant
m-a -t update
m-a -t -f get ipw2200
m-a -t build ipw2200
m-a -t install ipw2200
```

From the address indicated by the manufacturer (in its documentation), the firmware version compatible with the driver version is downloaded, decompressed and installed in */lib/firmware* (where X.Y is the firmware version):

```
tar xzvf ipw2200fwX.Y.tgz /tmp/fwr/
```

```
cp /tmp/fwr/*.fw /lib/firmware/
```

The module can then be loaded with `modprobe ipw2200`, the system is rebooted and then the result of device initialization, or errors, if any, can be seen with `dmesg | grep ipw`, (we can also check if it is loaded with `lsmod`):

The **wireless-tools** package and `iwconfig` command can then be used to analyze the Wifi device, and then configured as instructed in the Linux Introduction module.

11.9.1. The files host.conf, nsswitch.conf

The */etc/host.conf* file is maintained for compatibility and contains specific name resolution information. In current versions, its functionality is assumed by the */etc/nsswitch.conf* (Name Service Switch or NSS) file to determine the sources from which to obtain information from the name/domain services. The *host.conf* file has the following options:

- **order** (for compatibility only). It indicates the order of how the name search will be performed (e.g., bind, hosts, nis).

- **multi**. It indicates that it can be on, so it will return all valid addresses for a host that is in */etc/hosts* file instead of just returning the first (off). On can cause major delays when */etc/hosts* is a big file.

- **trim**. It is designed to be used with local hosts and domains and may appear more than once and should be followed by a list of domains, separated by ':,;' with a starting point and will automatically trim the given domain name from the end of any host name resolved through DNS.

- **reorder**. If enabled (on), the resolution library will attempt to rearrange host addresses with local addresses first. The default value is off.

Among the most important databases managed by the */etc/nsswitch.conf* file are:

- **aliases**: mail aliases.
- **ethers**: ID Ethernet.
- **group**: names and group IDs.
- **hosts**: host names and IPs
- **networks**: network names and IDs.
- **passwd**: passwords and user information.
- **protocols**: network protocols.
- **services**: network services.
- **shadow**: Shadow user passwords.

An example would be:

passwd: compat
group: compat
shadow: compat
hosts: dns [!UNAVAIL=return] files
networks: nis [NOTFOUND=return] files
services: nis [NOTFOUND=return] files

Where the first column is the database and the following are the service specification, e.g., files, db, or nis and where its sequence is the order until the result is obtained. Additional options can be indicated to take for a result such as [NOTFOUND=return].

The 'compat' descriptor is similar to 'files' (files on the local machine) except that it allows some additional words/options in local information files. The files consulted when indicated as a 'files' service are those that have already been discussed in this document and in the Introduction to Linux:

- **aliases**: /etc/aliases
- **ethers**: /etc/ethers
- **group**: /etc/group
- **hosts**: /etc/hosts
- **initgroups**: /etc/group
- **netgroup**: /etc/netgroup
- **networks**: /etc/networks
- **passwd**: /etc/passwd
- **protocols**: /etc/protocols
- **publickey**: /etc/publickey
- **rpc**: /etc/rpc
- **services**: /etc/services
- **shadow**: /etc/shadow

11.9.2. The /etc/hosts file

This file acts as a name server and is especially useful on a local network that does not have high variability of the IPs assigned to the names (or private local networks without private DNS server):

```
127.0.0.1 localhost
192.168.168.254 remix.nteum.local remix
# Private network nodes
192.168.168.1   nodo1.nteum.local node1
192.168.168.2   nodo2.nteum.local node2
# IPv6 hosts
::1         localhost ip6-localhost ip6-loopback
ff02::1  ip6-allnodes
```

```
ff02::2  ip6-allrouters
```

Aliases can be used for a machine name, which means that machine can be called in different ways for the same IP address.

In reference to the loopback interface, this is a special type of interface that allows the node to make connections with itself (for example, to verify that the network subsystem operates without accessing the network). By default, IP address 127.0.0.1 has been specifically assigned to the loopback (a ssh command 127.0.0.1 will connect to the same machine). It is very easy to set up (typically done by the network initialization script). In many systems (including Debian) the following line is added to the */etc/hosts* file: 127.0.1.1 <host_name>. This is created on systems that do not have permanent IP and to prevent any program from causing problems (for example, Gnome). The <*host_name*> label will match the machine name defined in */etc/hostname*.

For a system with defined IP, this line must be commented on and it is advisable to put a domain of the fully qualified domain name (FQDN) type (as the one that appears in the DNS) or if it is not necessary to insert one defined as <*host_name*>.<*domain_name*> instead of just putting <*host_name*> (for machines that do not have a record in a DNS, it is advisable to put as a local domain, for example *nteum.local* or *nteum.localdomain*).

11.10. Routing configuration

An important aspect of network management is routing configuration. While there is a topic on its difficulty, usually very simple routing requirements are needed. In a node with multiple connections, routing is about deciding where to send and what is received. A simple node (a single network connection) also needs routing, as all nodes have a loopback and a network connection (e.g., Ethernet or Wi-Fi). As explained above, there is a table called routing table, which contains rows with various fields, but with three extremely important ones: destination **address**, **interface** where the message will leave from and IP **address**, which will perform the next step in the network (gateway).

The `ip route` command allows modifying this table to perform the appropriate routing tasks. When a message arrives, its destination address is viewed, it is compared to the entries in the table, and sent through the interface, where the address best matches the package destination. Otherwise, if a gateway is specified, the message will be sent through the appropriate interface. Let's consider, for example, that our node is on a class C network with an address of 192.168.110.0 and has an address of 192.168.110.23; and the router with an Internet connection is 192.168.110.3. First, we need to configure the interface (eno1 in this example):

```
ip addr add 192.168.110.23/24 dev eno1
```

```
ip link set eno1 up
```

We do not need to, but we could add that all the packets in the network were linked to the interface with:

```
ip route add 192.168.110.0/24 dev eno1
```

If the packets need to be sent to an IP outside of the 192.168.110.0/24 network, it would be very difficult to have all the proper routes for all the machines to which we want to connect. To simplify this task, there is the (gateway) default route, which is used when the destination address does not match in the table with any of the entries, to configure it, run:

```
ip route add default via 192.168.110.3
```

Then we can see the result with `ip route` that will give information such as:

```
default via 192.168.110.3 dev eno1 proto static metric 100
192.168.110.0/24 dev eno1 proto kernel scope link src 192.168.110.23 metric 600
```

To add and to delete a route, for example:

```
ip route add 10.0.0.0/16 via 172.16.1.1
ip route del 10.0.0.0/16 via 172.16.1.1
```

To determine where a packet will be routing for a given IP, we can run:

```
ip route get 8.8.8.8
8.8.8.8 via 10.0.2.2 dev eth0 src 10.0.2.15
```

11.11. Configuring network services

The next step in network configuration is the configuration of the servers and services that will allow another user to access the local machine or its services. Server programs will use ports to listen to client requests, which will be routed to this service as *IP:port*.

In most *Nix systems there is a super-daemon called *inetd* that manages different network services by listening to connections on certain sockets (points of communication identified by ports). When it detects a connection, it decides which service the socket corresponds to, and it invokes a program to service the request; once the program is complete, it continues listening in the socket for new requests. Essentially, *inetd* allows running a daemon to invoke several others, reducing the load on the system, *inetd* is obsolete in GNU/Linux, and its replacement is called *xinetd*, which incorporates better features, performance and safety.

11.12. Configuring the xinetd

xinetd is the recommended option when we need to manage network services efficiently and securely and its configuration is through the */etc/xinetd.conf* file and the */etc/xinetd.d* directory.

There are some services that, although they could be put under the influence of *xinetd*, but it is advisable either because of the delay in the booting (`sshd` case) or because of the load that means booting and shutting it down in each request, which would generate an extra unnecessary load on services with many transactions. Daemons that are advised not to be under the influence of *xinetd* include:

- **ssh**: secure interactive connection as a `telnet` replacement and includes two configuration files */etc/ssh/ssh_config* (for the client) and */etc/ssh/sshd_config* (for the server) configuration files.

- **exim**: mail transport agent (MTA), including configuration files: */etc/exim/exim.conf*, */etc/mailname*, */etc/aliases*, */etc/email-addresses*.

- **fetchmail**: daemon used for downloading mail from a POP3 account, configuration file: */etc/fetchmailrc*.

- **procmail**: program used for filtering and distributing local mail, configuration file: *~/.procmailrc*.

- **DHCP**: service for management (server) or obtaining IP (client), */etc/dhcp/dhclient.conf* (client), */etc/dhcp/dhcpd.conf* (server). Directories/name may vary depending on the package installed.

- **DNS**: name service, e.g., dnsmaq in */etc/dnsmasq.d*.

- **CVS**: concurrent version control system, */etc/cvs-cron.conf*, */etc/cvs-pserver.conf*.

- **NFS**: network file system, */etc/exports*.

- **Samba**: network file system and printer sharing on Windows nnetworks, */etc/samba/smb.conf*.

- **CUPS**: printing system, */etc/cups/**

- **Web services**: for example, Apache2 on */etc/apache2/**.

- **Proxy services**: for example, Squid on */etc/squid/**.

The configuration of *xinetd* is very simple and is done through the */etc/xinetd.conf* file (which may include more files from the */etc/xinetd.d/* directory for better structuring). This file has a defaults section (parameters that will apply to all services) and service section (or files for each service to be configured), which will be the services that will be launched through *xinetd*. A typical example of the configuration might be that defaults are placed on *xinetd.conf* and the services are placed in separate files in the */etc/xinetd.d* directory, although, in this example, everything has been put together for simplicity:

```
# xinetd.conf
# They apply to all servers and can be modified for each service
   defaults
   {
      instances      = 10
      log_type       = FILE /var/log/service.log
      log_on_success = HOST PID
      log_on_failure = HOST RECORD
      }
   # The service name must be in /etc/services to get the correct port
   # If this is a non-standard server/port, use port = X
   service ftp
   {
      socket_type = stream
      protocol       = tcp
      wait             = no
      user     = root
      server       = /usr/sbin/proftpd
      }
   # SSH: although it is not recommended to put it under the control of xinetd but a
   # possible configuration is provided.
   service ssh
   {
      socket_type = stream
      protocol       = tcp
      wait             = no
      user     = root
      port       = 22
      server       = /usr/sbin/sshd
      server_args = -i
      }

   # tcp version of the ECHO service
   service echo
   {
      disable = yes
      type = INTERNAL
```

```
        id = echo-stream
        socket_type = stream
        protocol = tcp
        user = root
        wait = yes
        }
    # udp version of the ECHO service
    service echo
    {
        disable = yes
        type = INTERNAL
        id = echo-dgram
        socket_type = dgram
        protocol = udp
        user = root
        wait = yes
}
```

The disabled services (lines start with #) will not be available and neither will those with disable = yes. In the defaults section, we can insert parameters such as the maximum number of simultaneous requests for a service, the type of record (log) that we want to have, from which nodes we will receive default requests, the maximum number of requests per IP that will be handled, among others.

An interesting parameter is cps, which limits the number of incoming connections with 2 arguments: the first is the number of connections per second to be handled by the service and if it is greater, the service will be disabled by the number of seconds indicated in the second argument. By default, there are 50 connections and a 10-second interval that are considered appropriate parameters to contain a DoS attack.

There must be one service section for each service, and it may contain specific – and sometimes very detailed – parameters of the service. See `man xinetd.conf` for more details.

11.13. Security basics

It is important to take into account the security aspects in network connections (and it will be seen in more detail in the following sections) but a minimum of recommendations will be:

1) Do not activate services that are not required in both */etc/xinetd.conf* and independent services (standalone). *xinetd* is not installed in most distributions and if it is, the services must be analyzed and commented on or put *disable* =

yes. List the *systemd* managed services with `systemctl list-units --type service --all` and disable them with `systemctl disable name.service` (be careful with disabling services that are necessary for OS operation).

2) If an FTP service is installed (typically managed via *xinetd*), modify the */etc/ftpusers* file to deny that certain users can connect via ftp.

3) Modify the */etc/security/access.conf* file to indicate where (or the terminals) the users can connect from. It is important, for example, to indicate where the *root* user can be connected from.

4) Disable services such as `rlogin`, `rsh` that are not encrypted and allow access to the machine through the */etc/hosts.equiv* file without having to enter an access key (password). The functionality of these services is replaced by `ssh` securely and in an encrypted way.

5) In many distributions (Debian for example) it is important to configure */etc/security/access.conf*, the file indicating the rules of who and from where (login) can be connected to this machine. This file has a line in order with three fields separated by ':' of the *permission:users:origin* type. The first will be a + or - (access or denied), the second a user/s name, group or *user@host*, and the third a device name, node, domain, node or network addresses, or ALL. For example:

```
+:root:192.168.200.1   192.168.200.4   192.168.200.9
```

11.14. IP options

There are a number of IP traffic options that we should mention. Their configuration can be done dynamically - that is, they will only be active until the machine is restarted - (modifying pseudo-files of */proc/sys/net/ipv4/)* or statically in the kernel variables managed at */etc/sysctl.conf*. In */proc/sys/net/ipv4/* there are a number of files that identify the different options whose content must be modified, which in turn modifies this variable in the kernel. For example, if we want to enable *ip_forward*, we should run:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Or a 0 if we want to deactivate. The most commonly used are: *ip_forward* used for packet forwarding between interfaces or with IP Masquerading; *ip_default_ttl*, which is the lifetime for an IP packet (64 milliseconds by default) *ip_bootp_agent* variable logic (Boolean), which accepts packets (or not) with source address of type 0.b.c.d and destination of this node, broadcast or multicast.

To modify the IP options in the kernel variables statically, the */etc/sysctl.conf* file must be modified (see */etc/sysctl.d/* for additional variables and *sysctl.conf* (5) for more information). For example, to disable IPV6, edit the file and include a line with *net.ipv6.conf.all.disable_ipv6 = 1*. And to allow the *ip_forward* one with *net.ipv4.ip_forward=1*, save and to update the changes executing the command `sysctl -p` and the changes made will be seen.

11.15. Multiple IPs over one interface

There are some applications where it is useful to configure multiple IP addresses to a single network device. Service providers (ISPs) frequently use it to provide customized features (e.g., from World Wide Web sites) to their users. Aliases are appended to virtual network devices associated with the new device in a format such as *device:virtual_number*, for example, eno1:0.

A typical configuration would be in */etc/network/interfaces* to assign, for example, three IPs to **eno1**:

```
auto eno1
allow-hotplug eno1
iface eno1 inet static
      address 192.168.1.42/24
      gateway 192.168.1.1

auto eno1:0
allow-hotplug eno1:0
iface eno1:0 inet static
      address 192.168.1.43/24

auto eno1:1
allow-hotplug eno1:1
iface eno1:1 inet static
      address 192.168.1.44/24
```

Or also with the `ip addr add` command indicating the corresponding device.

11.16. DHCP service

DHCP (Dynamic Host Configuration Protocol) is a service that allows configuring the network parameters of each node that connects to it centrally and simply. Configuring a client (typically installed by default in all distributions) is very easy, as the *dhcp-client* package only needs to be installed, if it isn't already (e.g., in Debian *isc-dhcp-client*) by adding the word **dhcp** in the input corresponding to the interface we want to operate under the dhcp client (e.g., */etc/network/interfaces* must have *iface wlo1 inet dhcp* in order to configure wifi interface by dhcp).

Server configuration requires a little more attention, but does not present any complications. The installation, for example, in Debian is `apt-get install isc-dhcp-server` (it is one of the dhcp servers, but there are others such as `dnsmasq` or `kea`). In general, *dhcpd* configuration can be done by modifying the */etc/dhcp/dhcpd.conf* file. An example of this file is:

```
# Example of etc/dhcp/dhcpd.conf:
default-lease-time 1200;
max-lease-time 9200;
option domain-name "remix.cat";
deny unknown-clients;
deny bootp;
option broadcast-address 192.168.11.255;
option routers 192.168.11.254;
option domain-name-servers 192.168.11.1;

subnet 192.168.11.0 netmask 255.255.255.0 {
            not authoritative;
            range 192.168.11.1 192.168.11.254

            mars host {
                        ethernet hardware 00:00:95:C7:06:4C;
                        fixed address 192.168.11.146;
                        option host-name "mars";
            }
            saturn host {
                        ethernet hardware 00:00:95:C7:06:44;
                        fixed address 192.168.11.147;
                        option host-name "saturn";
            }
}
```

This will allow the server to assign the address range 192.168.11.1 to 192.168.11.254 as described by each node. If the corresponding host segment { ... } does not exist, they are randomly assigned. IPs are assigned for a minimum time of 1,200 seconds and a maximum of 9,200 (if these parameters do not exist, they are assigned indefinitely). On this server, we must also modify the */etc/default/isc-dhcp-server* file and modify the INTERFACESv4= "eno2" line to indicate the interface where the requests will be received. Finally, the service can be restarted with `systemctl restart isc-dhcp-server`.

With `/usr/sbin/dhcpd -d -f` we can see information about the service or we can also see its status with `systemctl status isc-dhcp-server`.

Since the DHCP protocol is performed via broadcast since the requesting node does not have the network configured (discovery event generated by the DHCP client), routing to other networks cannot be performed since the DHCP protocol considers the server and client to be on the same network. To get from one network to another where the dhcp server is located (and thus avoid having to put one server for each subnet), an intermediate agent called DHCP relay can be used. A DHCP relay operates on the machine that acts as an intermediary for the dhcp broadcast packets. Its installation is through `apt-get install isc-dhcp-relay` and it is configured with the */etc/default/isc-dhcp-relay* file by adding *SERVERS*= IP_DHCP__Server where IP_DHCP_Server will be the IP of the dhcp server. The file contains other options, such as defining the interface from which to listen to requests (if it is not indicated, it is all of them), and another option is to send to the DHCP server.

11.17. IP Network Address Translation (NAT)

NAT is a resource for a set of machines configured on a network (usually private) to use a single IP address as a gateway. This allows nodes on a local network, for example, to exit to the Internet (i.e., those using the private IP, for example, 198.162.10.1); but they cannot accept connections directly from outside, but only through the machine that has the actual IP. The IP Network Address Translation (NAT) is a replacement for what was known as IP Masquerade and is part of the kernel.

The most common case, nowadays, is having a set of virtualized machines connected to the host via NAT or a set of computers on a private internal network (which in turn will do NAT with the router). These example can be extrapolated to any physical devices on a private network in order to get access to Internet for example.

The nodes that must be masqueraded will be within a second network and each of these machines should have the address of the machine that performs the masking as default gateway.

On this machine (router) one interface can be configured to "look" at the internal network (for example, **eno2** connected to the 192.168.1.0/24 network) and another, to the external network (for example, **eno1**, connected to the 10.0.2.0/24 network - in this example, two private networks are used but if these machines need to connect to the Internet, somewhere there must be a NAT between private and public network); the steps for its configuration are:

1) Activate *ip_forward*, directly:

echo 1 > /proc/sys/net/ipv4/ip_forward or permanently modifying the */etc/sysctl.conf* file by removing the comment to the line *net.ipv4.ip_forward=1* and running `sysctl -p`.

2) Activating masquerade:

```
iptables -t nat -A POSTROUTING -o eno1 -j MASQUERADE
```

In order for the changes to be permanent (and not have to be made after each boot) it is recommended to install the *iptables-persistent* package (which will save the iptables rules when the machine is turned off and will load them when it is turned on) and to activate the ip_forward by modifying the file *sysctl.conf*. We can run `iptables -t nat -L` to check that the iptables rule is active. If the 10.0.2.0/24 network is connected to the Internet (obviously through another NAT), the connectivity of the internal machines could be verified by running, for example, `ping google.com`.

In case of connection problems, the command `tcpdump -i eno1|eno2` can be used to see if the packages are reaching eno1 or eno2 and detect where the problems are.

This is the simplest way with iptables, but it can also be done with the command `ip route add nat <parameters>` to manage rules that route and move addresses (this is called Stateless NAT).

11.18. Bridging

Bridging is a method of sharing connections; for example, the Internet connection between two or more computers, which is useful if a router with more than one Ethernet port is not available or is limited by this number. It is essentially used so that a computer that is connected to the Internet can, by another network device, connect to a second computer that does not have an Internet connection and provide it with Internet access. In physical systems it may seem strange, but it is common when we have a set of virtualized machines and we want everyone to have access to the Internet.

The command to use is the `brctl`, and it is included in the **bridge-utils package** that must be installed (`apt-get install bridge-utils`). This will allow us to set up and use a new interface (bridge), in the same way as **eno1**, **eno2**, etc., which does not exist physically (it is virtual) and will transparently obtain packages from one interface and move them to the other. The simplest way to create an interface is through `brctl addbr br0` and physical devices can be added with `brctl addif br0 eno1 eno2`.

Permanent configuration can be done via */etc/network/interfaces*:

```
iface eno1 inet manual
iface eno2 inet manual
# Bridge setup
iface br0 inet dhcp
```

```
    bridge_ports eno1 eno2
```

If a static configuration needs to be made it must be changed in *br0*:

```
iface br0 inet static
   bridge_ports eno1 eno2
   address 192.168.1.2
   netmask 255.255.255.0
   gateway 192.168.1.1
```

See Bridging Network Connections (Debian) for more details.

11.19. Domain Name System: (DNS)

The functionality of the DNS service is to convert machine names (readable and easy to remember by users) into IP addresses or vice versa. That is, if a GNU/Linux machine connected to the Internet is running a `host cv.uoc.edu`, the answer will be *cv.uoc.edu has address 52.215.162.218* (among others) and if `host 52.215.162.218` is executed, we will obtain *218.162.215.52.in-addr.arpa domain name pointer ec2-52-215-162-218.eu-west-1.compute.amazonaws.com* (which is the machine where the UOC CV really is, i.e., a machine on AWS and cv.uoc.edu is an alias to this machine).

The Domain Name System (DNS) is a tree-shaped architecture that prevents duplication of information and facilitates searching. Therefore, a single DNS makes no sense but as part of the tree. One of the most widely used applications provided by this service is called *named*, included in most GNU/Linux distributions, and is part of a package called *bind* (currently version 9.x), coordinated by the ISC (Internet Software Consortium). DNS is simply a database, so the people who modify it need to know its structure, otherwise the service will be affected. As a precaution, special care should be taken to save copies of files to avoid any interruption in the service. DNS servers, which can convert most DNS nodes to their corresponding IP are called recursive DNS servers. This type of server cannot change the names of the DNS nodes there are, they just ask other DNS servers the IP of a given DNS node. Authorized DNS servers can manage/change the IP addresses of the DNS nodes they manage and are typically contacted by a recursive DNS server for the purpose of knowing the IP of a given DNS node. A third variant of DNS servers is cache DNS, which simply stores information obtained from other recursive DNS servers.

The `bind` configuration is not complex but requires dedication and in this case a widely used DNS server called *dnsmasq* (and can also act with DHCP server) has been chosen which is very simple and recommended for medium/ small installations. For the *bind* configuration, we can consult information such as that of Linuxconf.org, among others.

Dnsmasq allows to simply configure a DNS server (forwarder) (and DHCP server in the same package) for small/medium networks and can handle requests for machine names that are not in the global DNS.

The way to quickly configure a DNS for proprietary and forwarder domain machines is to run `apt-get install dnsmasq`. If a simple DNS is desired, then it is already configured considering that the */etc/resolv.conf* file (or equivalents) will have an external DNS server (for example nameserver 8.8.8.8).

The settings can be adjusted on the */etc/dnsmaq.conf* file, but with the default settings we can already query external domains and it will act as DNS-cache. To complete the installation, a line with nameservers 127.0.0.1 must be added as nameserver in */etc/resolv.conf* (or equivalent service) before the other nameserver so that the requests can be resolved first by this DNS and then sent out. On machines where IP is obtained by DHCP and to prevent each IP renewal from updating the *resolv.conf*, we can modify */etc/dhcp/dhclient.conf* and remove the comment to the *prepend domain-name-servers 127.0.0.1* line; which will add the indicated line each time the IP is renewed.

Dnsmasq will resolve the internal domains from the FDQNs of the */etc/ hosts* machines such as for example if we have a line such as *192.168.1.37 remix.nteum.world remix*, we can run `host remix.nteum.world` and it will respond to us with the corresponding IP. The */etc/dnsmasq.conf* configuration file includes a number of options for organizing internal domains and other parameters not only for DNS configuration but also for the DHCP server.

Another interesting DNS server is MaraDNS, which can be used as a recursive DNS (cache) or as authoritative name server. Its configuration is very simple and there is a lot of documentation on the developer page that explains how it is done based on the role of the DNS we want to deploy.

11.20. Information service: NIS (YP)

In order to facilitate administration and provide user comfort in networks of different sizes, information services that propagate user information are usually executed on different machines so that these users do not need to have local accounts on all the machines. Some of these services are the Network Information Service, NIS (or Yellow Pages, YP, in the original Sun

definition), LDPA (Lightweight Directory Access Protocol), or Active Directory (for Windows systems but where the server can be installed in GNU/Linux via the Samba4 package).

The NIS architecture is of the client-server type, i.e., there is a server that will have all the databases and some clients that will query this data transparently for the user. Therefore, the possibility of configuring "backup" servers (actually called secondary) should be considered so that users are not blocked in case the main server crashes. That is why the architecture is really called multi-server (master+mirrors+clients).

In this section, we will discuss the configuration of GNU/Linux as a NIS client/server and the information that can be distributed in NIS is as follows: users (login names), passwords (*/etc/passwd|shadow*), user home directories and group information (*/etc/group*), networks, hosts, among others. This configuration presents the advantage that, from any client machine or from the same server, the user can connect with the same account and password and to the same directory (although the directory must be previously mounted on all client machines by NFS or via the *automount* service).

To install the server on a machine (nis server) the nis package must be installed (`apt-get install nis`). During the installation, a NIS domain will be requested, under which it will group all the machines it will serve. It is recommended that this NIS domain does not match the Internet domain or host name. For example, if the server has name.domain=remix.*nteum.world* it could be put as nis domain= NISREMIX (note that it is case sensitive, so NISREMIX is different from Nisremix). This domain name can be queried with the order nisdomainname, (or */proc/sys/kernel/domainname* can also be consulted) and can be reconfigured with `dpkg-reconfigure nis`. The */etc/default/nis* file must then be modified to modify the NISSERVER=master line to indicate the server role. Also (although it can be done in a second step), the */etc/ypserv.securenets* file must be adjusted to modify the line indicating 0.0.0.0 0.0.0.0 that gives access to everyone and restrict it to the client network, for example 255.255.255.0 192.168.1.0.

Finally, the */etc/hosts* must be modified to have the machine with the name defined in */etc/hostname* (it can be changed/displayed with the `hostnamectl` command) with an FQND line such as *192.168.1.30 remix.nteum.world remix*.

The server configuration is done with the `/usr/lib/yp/ypinit -m` command; in some distributions it is necessary to verify that the */etc/networks* file exists, which is essential for this script. If this file does not exist, an empty file can be created with `touch /etc/networks`; this script will request which the NIS servers will be, indicating the local machine by default, and must end in Crtl+D. To restart the service, we can run `systemctl restart nis`.

It should be noted that from this point onwards, the commands to change password or user information, such as `passwd`, `chfn` or `adduser`, are valid only for changing local information. If we want to change NIS information, commands such as `yppasswd`, `ypchsh` and `ypchfn` must be used. If users are changed or local files are modified, NIS tables can be reconstructed by running the `make` command in the */var/yp* directory to update the tables.

The configuration of a backup server is similar to that of the master, except that in */etc/default/nis* it must be configured as NISSERVER = slave. On the master, it should be indicated that it distributes the tables automatically to the backup ones, putting NOPUSH = "false" in the */var/yp/Makefile* file. To initialize the master who its backup server is, `/usr/lib/yp/ypinit -m` will be executed indicating the name of the backup server. This will reconstruct the maps but will not send the files. To do this, on each backup server, `systemctl stop nis; /usr/lib/yp/ypinit -s name_master_server; systemctl start nis` must be executed. To verify that the NIS server is active, we can check with `systemctl status ypserv` or with `rpcinfo -p` and the domain with `ypwich -d domain`.

It is recommended after using `adduser` to add a new user on the server, run `cd /var/yp; make` to update the NIS tables (and whenever any user characteristics are changed, for example the keyword with the `passwd` command, because it will only change the local password and not the NIS password).

A NIS client is a machine that is attached to an existing NIS domain running `apt-get install nis`. The NIS package installation procedure will request a domain name (NIS domain name) that will describe the set of machines that will use the NIS and that has been previously entered into the server (NISREMIX in this example). Since the NIS server address has not been configured yet, the client might attempt to locate it by sending a broadcast message. If it doesn't find it, it will (after a short time) give a message similar to the following: [....] *Starting NIS services: ypbind[....] binding to YP server...failed (backgrounded)*.

The NIS client uses the *ypbind* command to find a server for the specified domain, either via broadcast (not advised) or by searching for the server indicated in the configuration file */etc/yp.conf* (recommended). This file must have a line with *ypserver 172.16.1.1* where the IP is that of the master server (this file supports different syntaxes and can be consulted with `man yp.conf`). In order for a user who is not defined on the client machine to access through the nis service, the */etc/nsswitch.conf* file must be modified to include nis in the following query bases (see `man nsswitch.conf`):

```
passwd: compat nis
group: compat nis
shadow: compat nis
```

```
netgroup: nis
hosts: files dns nis
```

We can then restart the service with `systemctl restart nis` and verify if it accesses the tables executing a `ypcat passwd` that will display the list of users propagated from the server and with `rpcinfo -p` or with `systemctl status ypbind` we will see the *ypbind* service active.

Although the *autohome* and *automount* can be configured for user directories, it is recommended to mount the */home* directory to the clients by NFS (this will be seen in the next sections) so that users have access to their files on each client machine.

11.21. Remote connection services: ssh

For remote connection, it is recommended to use `ssh` (a command that is already in all operating systems, included in Windows with *putty* or *mobaterm*) instead of other options that are not secure (e.g., `telnet`, `rlogin` or `rsh`). The OpenSSH environment (client and server) provides an encrypted connection (it is advisable not to use version 1.0 of the protocol) and a number of utilities, such as `ssh` (client) `scp` (secure copy), and `sftp` (secure file transfer), key management (`ssh-add`, `ssh-keysign`, `ssh-keyscan`, and `ssh-keygen`) and `sshd` servers (ssh server), `sftp-server` (ftp server), and `ssh-agent` (authentication agent). All current distributions incorporate the ssh client and sshd server by default (otherwise, install the *openssh-server* package for the server and the *openssh-client* for the client) and the *OpenSSL* library, used by these programs, needs to be updated as security issues have been encountered (e.g., *heartbleed* but it has been quickly fixed in GNU/Linux distributions).

Connection to a remote server is done by running `ssh -l user hostname` or `ssh user@hostname` and through SSH other connections, such as X11 or any other TCP/IP connection, can be encapsulated. If parameter -l is omitted, the user will connect to the same local user and in both cases the server will request the password to validate the user's identity.

SSH supports different authentication modes (see `man ssh`), for example, one based on the RSA algorithm and public key. With the `ssh-keygen -t rsa|dsa` command, we can create the user identification keys that will be saved by default in the *$HOME/.shh* directory, the *id_rsa* and *id_rsa.pub* files, which are the private and public keys, respectively. The user could copy the public (*id_rsa.pub*) to the remote server with `ssh-copy-id user@host_remote` that will include the public key in the user's *$HOME/.ssh/authorized_keys* file of the remote machine. This file may contain as many public keys as sites from which we want to connect to this machine remotely. Syntax is one key per

line although the lines will be of considerable size. After we have entered the public keys of the user-machine in this file, this user will be able to connect without password from that machine.

Normally (if keys have not been created), the user will be asked for a password, but since the communication will always be encrypted, it will never be accessible to other users who can listen over the network.

To remotely execute a command, we can simply run `ssh login-name@remote-host remote-command`, for example `ssh user@remote-host ls -al`.

Both the client and server support multiple configurations that can be changed from */etc/ssh/ssh_config* and */etc/ssh/sshd_config* (we must remember to restart the server after changing files with `systemctl restart ssh`). On the server, some of the most commonly used options are *PermitRootLogin yes|no* to allow the root user to connect or not, *IgnoreRhosts yes* to avoid reading the users' *$HOME/.rhosts* and *$HOME/.shosts* files, and *X11Forwarding yes* to allow Xwindows applications on the server to be displayed on the client screen (very useful in remote server management with the command ssh -X host_a_administer command among others).

11.22. Chained connections

Very often, access to internal networks is only possible through an SSH server (e.g., that are in the DMZ –demilitarized zone that is a physical or logical subnetwork that contains and exposes an external service to untrusted users–) and from this it is possible to connect to internal SSH servers and then reach the machine of interest. The way to do this is to first connect to the M1, then to the M2 (to which it is only possible to connect from the M1) and from it to the M3 (to which it is only possible to connect from the M2). One way to facilitate authentication is to use agent forwarding with parameter -A and by setting the public key to all *$HOME/.ssh/authorized_keys* so when the different commands are executed, the key request can be forwarded to the previous machine and so on, the commands will be `ssh -A m1.org` and from this `ssh -A m2.org` and in turn `ssh -A m3.org` but can automate with `ssh -A -t m1.org ssh -A -t m2.org ssh -A m2.org` where the option -t has been included for `ssh` to assign a pseudo-tty to it and only the final screen will be displayed). The SSHmenu applet can help automate all these types of commands.

Another way to effectively manage this "multi-hop" connection is through the ssh *ProxyCommand* option (see `man ssh_config`) that will enable the connection more efficiently. The following commands are defined in *$HOME/.ssh/config*:

```
Host m1
```

```
    HostName m1.org
Host m2
    ProxyCommand ssh -q m1 nc -q0 m2.org 22
Host new
    Hostname 158.109.174.19
    User root
    ForwardX11 yes
    ProxyCommand ssh -X user@router.uoc.edu -p 223 -W %h:%p
```

The first command (when running `ssh m1`) will simply connect to m1.org but the second command when running `ssh m2` will establish an ssh connection over m1 but the *ProxyCommand* command will use in `nc` command to extend the connection over m2.org. Another way to use it is with the third configuration when running `ssh new` will connect to router.uoc.edu with the 'user' user and port 223 and then to the IP indicated by *Hostname* and the user indicated in *User*.

11.23. Remote file services: NFS (Remote File System)

The NFS system allows a server to export a file system so that it can be used interactively from a client. The latest version of NFSv4 includes a series of additional daemons (*idmapd*, *statd*) as well as a series of modules for the new functionality of this version.

To install (on Debian) the client must run `apt-get install nfs-common` and the server with `apt-get install nfs-kernel-server`. The server is managed through `systemctl start|stop|restart|staus nfs-kernel-server`). The server uses a file (*/etc/exports*) to manage remote access and the control of file systems.

On the client (or another user via `sudo`), the root can mount the remote system via the `mount -t nfs Ipserver:remote_directory local_directory` command and from this point on, the remote-directory will be seen within the local directory (it must exist before running the `mount`).

This task on the client can be automated by using the */etc/fstab* file including a line, for example, *Ipserver:/home /home nfs defaults 0 0* which indicates that the */home* directory of the *Ipserver* host will be mounted on the local */home* directory. In addition, this file system will be mounted with the default parameters (see `man mount` section mount options for *ntfs* and `man nfs` for specific options for NFSv4). The last two zeros indicate that the file system should not be dumped and that the `fsck` will not be activated on it.

The */etc/exports* file on the server acts as the ACL (Access Control List) of the file systems that can be exported to the clients. Each line contains a filesystem to be exported followed by the clients that can mount it, separated by blank spaces. Each client can be associated with a set of options to modify behaviour (see `man exports` for a detailed list of options). An example of this could be:

```
# Example of /etc/exports
/pub        *(ro)
/soft       192.168.32.0/24(ro)
/home       192.168.10.0/24(rw,no_root_squash,no_subtree_check)
```

The first line exports the */pub* directory to any client in read-only mode, the second line exports the */soft* directory to the 192.168.32.0/24 network in read-only mode and the third line exports the */home* directory to the 192.168.10.0/24 network in read-write (rw) mode, with *no_subtree_check* (it indicates that the path/file verification will not be performed on a request on the server) and *no_root_squash* (indicates that the root users of both machines are equivalent).

Two useful commands for working with NFS are `exportfs` (shows and allows to update the modifications that have been made) and `nfsstat` (that will allow us to obtain NFS operation statistics).

11.24. Virtual Private Network (VPN)

A VPN is a network that uses the Internet as a data transport, but prevents it from being accessed by members outside of it. Having a VPN means having different nodes on different networks joined together on a single network through a tunnel where encrypted communication packets are sent. It is used when remote users are accessing a corporate network to maintain data security and privacy. To configure a VPN, we can use a variety of SSH (SSL), CIPE, IPSec, PPTP methods, among others.

As a proof of concept, in this section we will use OpenVPN, which is an SSL VPN-based solution, and it can be used for a wide range of solutions, for example, remote access, point-to-point VPN, secure WiFi networks, or enterprise distributed networks. OpenVPN deploys OSI layer 2 or 3 using SSL/TLS protocols and supports certificate-based authentication and other certification methods, and should not be confused with an application proxy server. OpenVPN has different access and configuration options where the Access Server version (with web interface it allows only 2 simultaneous VPN connections for free) and the Community Edition version with text mode configuration and unlimited connections, depending on how it is configured. This latest version will be used in this test.

There are other open-source options to consider when setting up a VPN, among which we can mention Wireguard. This package allows configuring a VPN very simply and quickly using the latest generation cryptography. The design goal is to make it simpler and use fewer resources than IPSec and deliver equal or better performance than OpenVPN.

11.25. Installation and testing in raw mode

A Debian machine and an Ubuntu machine will be used in this section, but it is similar in all other distributions. OpenVPN must first be installed on both machines: `apt-get install openvpn`. Depending on the distribution, it may give some errors, as it tries to start the service, but since it is not configured yet, it shows some warnings. Then, it must be tested in raw mode if the connectivity between server and client works or if there is any impediment (for example, a firewall). To check this, we must run:

1) From the server:

```
openvpn --remote 10.9.8.2 --dev tun1 --ifconfig 10.9.8.1 10.9.8.2
```

2) From the client:

```
openvpn -remote 10.9.8.1 --dev tun1 -i-fconfig 10.9.8.2 10.9.8.1
```

Both machines must be connected; in this example, the IP of the client machine is 10.9.8.2 (Ubuntu) and the IP of the server machine (Debian), is 10.9.8.1. A series of messages will appear on the two terminals that have executed the commands, ending with a message similar to '*Initialization Sequence Completed*'. On another terminal, a ping 10.9.8.1 can already be performed from the client and ping 10.9.8.2 from the server to check that it works. And we can also see the *tun1* virtual interface created on both machines executing `ip a`.

To finish the application, we must simply do a *Crtl+C* (and we will see how the tun1 virtual interface disappears as well). It must be noted that this mode only allows for verification of connectivity but is not effective for stable communication (see OpenVPN documentation).

11.26. VPN with static key exchange

To analyze this service, an OpenVPN option called VPN with pre-shared secret will be used, which offers a simple way to configure an ideal VPN for testing or point-to-point connections. Its advantages are simplicity and that an X509 PKI certificate is not required to maintain the VPN. The disadvantages are that

it only allows one client and one server. By not using the PKI mechanism (public key and private key), the key must exist in each peer and must have been previously exchanged through a secure channel.

We must remember that in this example the VPN tunnel will have over the IP=10.9.8.1 server and the client with IP=10.9.8.2. The communication will be encrypted between the client and the server over UDP port 1194 (which is the default OpenVPN port). After installing the package, the static key must be generated in */etc/openvpn* and copied securely to the client:

```
cd /etc/openvpn
openvpn --genkey --secret static.key
scp static.key root@10.9.8.2:/etc/openvpn
```

In this case, the secure copy (scp) command has been used to transfer the *static.key* file to the client (*10.9.8.2*) over a secure channel. The */etc/openvpn/tun0.conf* server configuration file:

```
dev tun0
ifconfig 10.9.8.1 10.9.8.2
secret /etc/openvpn/static.key
```

The */etc/openvpn/tun0.conf* client configuration file:

```
remote 10.9.8.1
dev tun0
ifconfig 10.9.8.2 10.9.8.1
secret /etc/openvpn/static.key
```

Before verifying the VPN operation, we must ensure on the firewall that the 1194 UDP port is open on the server and that the *tun0* virtual interface used by OpenVPN is not blocked on either the client or the server. It must be taken into account that 90% of connection issues encountered by new OpenVPN users are related to the firewall.

To verify OpenVPN between two machines, run both on the server side and on the client side (*--verb 6* will display additional information to the output and it can be avoided in subsequent runs):

```
openvpn --config /etc/openvpn/tun0.conf --verb 6
```

This will give an output that will end with something similar to *Initialization Sequence Completed*. To verify its operation, for example, a *ping 10.9.8.1* can be executed from the client, so we will see its response and also a message of the tunnel activity in the server console.

To add compression over the link, the following line must be added to the two full configuration **comp-lzo** files, and to protect the connection via a NAT router/firewall and follow the IP changes via a DNS, if one of the peers changes, add to the two configuration files:

```
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

More information can be found in Static Key Mini-HowTo. As we have seen, the previous execution is in console, so the console is blocked with the VPN execution; but once purged, it is necessary to boot up as daemon and for this, it is necessary to add to the *letc/openvpn/tun0.conf* configuration file the word **daemon** and change in the file */etc/default/openvpn AUTOSTART="tun0"* to indicate the name of the VPN (it is also possible to put "all" that will execute all the */etc/openvpn/*.conf*) configuration files. Then it is necessary to boot on both sides (`systemctl start openvpn`) and test with an `ip a` command that there is a *tun0* and its functionality with a `ping`.

Once the connection is established, it can be used as if it were any other IP and access the services over the remote host via the VPN tunnel.

To create a fully functional VPN, it must be created with a PKI infrastructure, and for this purpose OpenVPN presents differences between versions (prior to 2.3 or after it, see OpenVPN 2x HowTo). To do this, it is recommended to follow the documentation, but the problem is in generating the PKI certificates, for which an easy-rca2 script can be downloaded from GitHub (https://github.com/OpenVPN/easy-rsa-old) for version 2.3 or easy-rca3 for the remaining versions (https://github.com/OpenVPN/easy-rsa).

11.27. Useful network management tools

There is a set of complementary (or replacement) packages and tools that either improve machine safety (recommended in harsh environments), or aid in network (and overall system) configuration in a more friendly way. These packets can be a great help to the network administrator to prevent local users or intruders that exceed their attributions (usually not by the local user, but through phishing) or help the new user to properly configure the services.

In this regard, it is necessary to consider:

1) Iptables: is a set of built-in functionalities into the Linux kernel to intercept and manage network packets. The main component is `iptables`, which functions as a firewall tool allowing not only filter actions but also network address translation (NAT) actions – as already used in the corresponding section of this document – or redirections/registration of communications.

Once the rules are configured, we can check the open services/ports, for example, from another machine, with a tool such as `nmap` that will display the open ports on the machine configured with iptables.

2) GnuPG: GnuPG is a full implementation of the OpenPGP standard defined by RFC 4880. GnuPG (or GPG) enables encryption and signing of data of all types and features a very versatile key management system, as well as access modules for all types of public key directories. GPG can operate on command-line or integrated into tools through its libraries and also provides support for S/MIME (Secure/Multipurpose Internet Mail Extensions).

To create the pair of keys, we must run `gpg --gen-key` by answering its questions. To display the keys created, we must run:

```
gpg --list-keys
```

or

```
gpg -v -fingerprint
```

which will result in key information. The next thing is to create a revocation certificate because, although it is not important now, when the public key is on a server and we want to destroy it (for different reasons), the only way is to revoke it. To do this, we must execute `gpg -a --gen-revoke` and copy the information between [BEGIN] and [END] into a file and keep it safely, since it can annul the public key. The public key must then be "public" on a server, such as *pgp.mit.edu*, by running

```
gpg --keyserver=x-hkp://pgp.mit.edu -a --send-keys KEY_ID
```

where the last number is the KEY_ID obtained from the key. To incorporate a public key of another user into our key-ring, we will

```
gpg --import <file>
```

or ask the `key-server gpg --search-keys <description>`, so we get the key needed to then use it to encrypt data for that user, for example. We can use `gpg --delete-key <description>` to delete it.

GPG solves the issue of the need for key exchange in advance with the public and private key mechanism, but this brings a new problem: if someone's public key is received (e.g., from a key server), how do we know that this key actually belongs to the person to whom it is said to belong? Anyone could create a key in someone else's name and use it, and no one would realize there's a phishing!

That's why care must be taken when accepting other keys and ensuring that the key is from a certain person and that it is secure (checking it with the Key-ID and fingerprint). An easy mechanism is to sign a key and the more known users sign this key, the more (everyone) will be sure that it belongs to the user they know (this is called key-signing). A user can sign another key (with theirs) to ensure that that key belongs to someone that they are sure of who it is, and also if a public key is received from someone they do not know but it is signed by several people who they know, then this key can be trusted to belong to that person (trust).

To sign a public key it must be in the key-ring and execute the command `gpg --edit-key <description>`, where "description" can be the name/email or any data or part of the key we want to sign. Then it will enter command mode and *sign -> confidence level (0-3) -> Y -> passwd -> save* should be indicated.

Subsequently, this public key should be uploaded back to the server with the new signature: `gpg -a --send-keys <key-ID>`. To update the keys that we have in the local (key-ring), we must run `gpg --refresh-keys`. To encrypt a file, we can run `gpg -e -a -r <description> file`, which will be called *file.asc*; since the *-a* that indicates that the output must be ASCII has been included, if it does not exist, it will be generated in binary as *file.gpg*. To decrypt, `gpg -d -or file.asc output_file` must be done, which will request the *passwd* and generate it in output_ file.

For using it in mail or other applications, it is advisable to integrate `gpg` with the application used (e.g., see GNUPGP + Thunderbird information):

3) Logcheck: One of the activities of a network administrator is to check log files daily (more than once a day) for possible attacks/intrusions or events that may give hints about these issues. This tool selects (from log files) condensed information of potential problems and risks and then sends it to the manager, for example, via email. The package includes utilities to run autonomously and remember the last verified entry for subsequent runs. The list of files to be monitored is stored in */etc/logcheck/logcheck.logfiles* and the default setting is appropriate (if most of the */etc/syslog.conf* file was modified). Logcheck can work in three modalities:

- **Paranoid**. This mode is very detailed and should be limited to specific cases such as firewalls.
- **Server**. It is the default mode and the recommended one for most servers.
- **Workstation**. It is the right mode for desktop stations.

This tool allows for a full configuration of the filters and rules can be classified as "intrusion attempt" (cracking), stored in */etc/logcheck/cracking.d/*; "security alert" stored in */etc/logcheck/violations.d/*, and those that are applied to the rest of the messages.

4) PortSentry and **Tripwire**. PortSentry is part of a set of tools that provide host-level security services for GNU/Linux, protect against port search and detect signs of suspicious activity. Tripwire is a tool that will assist the administrator by notifying them of possible file changes and modifications to prevent possible (major) damage. This tool compares the differences between current files and a previously generated database to detect changes (insertions and deletion), which is very useful for detecting possible modifications to vital files, such as in configuration files.

5) Tcpdump and **Wireshark**: Tcpdump is a very powerful tool that is in all distributions and will be used to analyze network packets. This program allows for the dump of network traffic and can analyze most of the protocols used in current distributions. Wireshark is another (more complex) tool that has a graphical interface for analyzing packages and also allows them to be decoded and it analyzes their content (it acts as *a* network sniffer). Both tools are installed under the usual procedure and are pre-configured in almost all distributions.

Given the importance of analyzing where the packages come from and where they are going, some common `tcpdump` commands will be displayed:

- `tcpdump`: default parameters -v or -vv for the displayed information level, -q fast output.
- `tcpdump -D`: it makes interfaces available for capture.
- `tcpdump -n`: it displays IP instead of addresses.
- `tcpdump -i eno1`: it captures the eno1 traffic.
- `tcpdump udp`: UDP packages only.
- `tcpdump port http`: port 80 (web) packages only.
- `tcpdump -c 20`: first 20 packages only.
- `tcpdump -w capture.log`: it sends the data to a file.
- `tcpdump -r capture.log`: it reads data from a file.
- `tcpdump host` *`www.uoc.edu`*: only packages containing this address.
- `tcpdump src 192.168.1.100 and dst 192.168.1.2 and port ftp`: it displays ftp packages with source from 192.168.1.100 and destination to 192.168.1.2.
- `tcpdump -A`: it displays the contents of the packages.

6) fail2ban: it is an essential tool to protect against brute force attacks used especially for controlling SSH connections, but it can be configured to monitor any login service through blocking the attacker's IP via iptables (default Linux kernel firewall).

7) system-config-*: in RHEL and derivative distributions (Fedora, Centos, Rocky, ...) and in some cases in Debian and derivative distributions also, there is a wide variety of graphical tools called *system-config-something-* and where *something-* is what they are designed for. In general, if we are in a graphic environment, we can reach each one of them by using a menu.

8) Other tools:

- **Nmap**: scanning and auditing a network for security purposes.
- **OpenVas**: vulnerability analysis.
- **Snort**: intrusion detection system, IDS.
- **Suricata**: highly scalable IDS that is very flexible and complete.
- **Netcat**: simple and powerful utility for debugging and exploring a network.
- **Hping3**: it generates and sends ICMP/UDP/TCP packages to analyze the operation of a network.

12. Local security

Local security is essential for system protection, as, typically, after a first attempt to access from the network, it is the second barrier of protection before an attack manages to gain part of the control of the machine. In addition, most attacks end up using internal system resources. In this section, the different points of risk will be analyzed.

1) Bootloaders. This is the first risk point if an intruder has physical access to the machine. One of the problems is whether the attacker can boot from removable devices (for example USB) as it could access the data of a GNU/ Linux partition (or also in Windows environments) just by mounting the file system and could be placed as a root user without needing to know any passwords. Or he/she could also access to modify the kernel boot form so that it doesn't ask for the password. For the first case, it is necessary to protect the system boot from the BIOS, for example, by protecting password access, so that boot from USB or other external connections is not allowed.

The next step is to protect the bootloader, (typically Grub2) so that the attacker cannot modify the core boot options or directly modify the boot. In the case of Grub 2, it is possible to do this by using *menuentry*, available in the configuration, by defining passwords for specific users, which can then be added to each defined menuentry, in addition to the possibility of specifying a super user who has access to all the entries. Grub2 uses passwords in clear, so we must make sure that the file does not have read access for other users, or instead use alternative methods for password creation, for example, by using *grub-mkpasswd-pbkdf2*, which will allow us to generate hashes of the password to be used. Configuration details can be seen in Grub2/Passwd.

2) Passwords and shadows. We must remember to use hash passwords (usually sha512) in */etc/shadow*.

3) Bits sticky and setuid. Another important issue is some special permissions that are used over executables or scripts.

The sticky bit is used primarily in temporary directories, where we want any user to be able to write, but only the owner of the directory or the owner of the file in the directory (e.g., `/tmp`) can delete it . Care should be taken that there are no directories of this type, as they can allow anyone to write on them, so it should be checked that there are no more than those purely necessary as temporary. The bit is placed by (`chmod +t dir`) and can be removed with the option `-t`. In a `ls` it will appear as a directory with `drwxrwxrwt` permissions where the last letter is a `t`.

The *setuid bit (or setgid)* allows a user to run (either an executable binary program or a shell script) with the permissions of another user. This may be useful in some cases, but it is potentially dangerous especially in the case of scripts, as they could be edited and modified to perform any task. These programs need to be controlled and if *setuid* is not needed, it should be deleted. The bit is placed by `chmod +s`, either by applying it to the owner (then called *suid*) or the group (it is called bit *sgid*); it can be removed with `-s`. In the case of viewing with `ls`, the file will appear with `-rwSrw-rw` (an S instead of an x), if it is only *suid*, in *sgid* the `S` would appear in the second `w`.

It is also possible to detect programs that are running with suid permissions, using `ps ax -or pid,euser,ruser,comm` where if the effective user (euser) and the real user (ruser) do not match, it will surely be an executable or script with suid permissions.

4) Hosts. There are a number of special configuration files that enable access to a series of hosts for some network services, but whose errors can allow local security to be attacked later. Examples of this are `.rhosts` in the user directory, `/etc/hosts.equiv, /etc/hosts.lpd`, among others.

5) PAM. PAM modules are a method that allows the administrator to control how the user authentication process is performed for certain applications. Applications must have been created and linked to PAM libraries. PAM configuration is present in `/etc/pam.conf` (maintained for compatibility) and in the `/etc/pam.d` directory, where one PAM configuration file appears for each application that is using PAM modules. For its configuration, see How to Configure and Use PAM in Linux.

6) System alterations. Another problem may be altering basic system commands or configurations, by introducing trojans or backdoors into the system, by simply introducing software that replaces or slightly alters the behaviour of the system software. A typical case is the ability to force the root user to run false system commands, for example, if the root includes the "." in its PATH variable, this would allow for command execution from its current directory, which would enable the placement of files that replace system commands that would be executed first before system commands.

The same process can be done with a user, although because their permissions are more limited, it may not affect the system as much, but the user's own security instead. Another typical case is the fake login screens, which can replace the typical `login`, `passwd`, process, with a fake program that stores the entered passwords.

If these alterations occur, it will be essential to use control tools such as `tripwire`. For Trojans, tools such as `chkrootkit` or `rkhunter` can be used to detect known rootkit or `clamv` for detecting viruses.

7) Limited resources, cgroups and chroot. Common management of existing machine processes, whether perfectly valid or harmful (intentional or careless), can lead to resource saturation situations (CPU, memory, network resources, or simply concurrent user sessions and/or running processes).

To control this problem, limits can be entered to some of the resources used with the `ulimit` command (current values can be seen with `ulimit -a`), although the global configuration of limits is maintained in the */etc/security/limits.conf* file. In fact, these limits, which can be imposed on CPU time, maximum number of processes, amount of memory, are read by PAM modules during the *user* login process, and set by default from the limits imposed on each user. Not setting limits and using default behaviour can make the system 'fall' with some ease: a user can easily create a script called a *Fork Bomb* whose code is available in various languages and crash the system if there are no set limits (for example, put the `limits.conf on` a line with `* hard nproc 128` to limit the number of processes to 128 for all users and it can be checked with `limit -a` in the user's account) .

For the establishment of limits, it should be noted that there is also a functionality at the kernel level, introduced in the latest versions and used mostly by the *systemd* system, called *cgroups*, that allows controlling and requesting system resources of different types: CPU, memory, or network access. The list of subsystems supported with `cat /proc/cgroups` and with a set of administrative utilities (*libcgroup-tools*) that can be used to control these groups can be examined.

Another way to control resources to a given application, service or user is through the creation of *chroot* environments, (a kind of prison-environment -jail-, where specific limits will be placed on what can be done without affecting the rest of the system. It is based on a call to the *chroot()* system that basically redefines what the associated child process perceives as root directory (/), which, for the created environment (or jail), can be associated in a certain file system position, perceived by the process as root (/), and will search from there for all configurations, system components, libraries, etc. from that false root. Since the structure of the file system is changed to a new one, the process will only work if it has all its data in the new zone and all the necessary data, files and commands must be replicated (for example, the `bash` and its libraries, to mention one). It should be noted that a chroot environment protects against access to files outside the (jail) area, but does not protect against other limits, such as system use, memory access or other similar ones. This operation, in Debian for example, can be done from `limits.conf` as mentioned previously.

8) Hardening-Wrapper. They are a series of techniques used to mitigate various security issues related to attacks carried out by executables and stack overflow, heap and protections against access to data memory zones and executable code.

These techniques are used in conjunction with the compiler (e.g., GNU `gcc`), using a series of parameters and options (flags) passed in compile time, whether for the compilation of user applications, service clients or the service server part (daemons), which from their code sources and the compile process are protected against various attack techniques on the executables.

In Debian, the complementary utilities and files can be installed for the compilation process, in addition to having the *`hardening-check`* utility, which allows checking the protections that a certain command, application or daemon has. We simply need to install the `apt-get install hardening-wrapper hardening-includes` packages and then check the daemon/service, for example, to check *sshd* `hardening-check /usr/sbin/sshd`.

These techniques provide a good foundation for protecting system executables from attacks while they are running, as well as protecting the data and code from dynamic changes an attacker might introduce.

However, it is worth noting some defects, such as that these techniques constantly vary, that certain problems in the implementation of protection techniques may be exploited, or that in certain cases these protections may give false positives.

12.1. Protection using wrappers

TCP wrappers is a software that runs prior to the service that manages and verifies an access control rule, typically listed at */etc/hosts.allow* and */etc/hosts.deny*. This wrapper can function as a previous service (`tcpd`) and if it verifies the access rules, it calls the corresponding service (option used in the already obsolete `inetd`) or its libraries can be integrated into the *daemon*/service code itself (for example, `xinetd` or `sshd`).

To know if the corresponding binary has the `libwrap` reference included, for example the sshd daemon, `ldd /usr/sbin/sshd | grep libwrap` can be run which will show whether it has the library or not (in Debian branches it is included).

The wrapper is controlled from the `/etc/hosts.deny` files, where it specifies which services are denied and to whom and `/etc/hosts.allow`, where it indicates which service and to whom its entry is enabled (see `man hosts_access` and `hosts_options` for detailed syntax).

12.2. Protection using firewalls

A firewall is a system or group of systems that enforces inter-network access control policies. The firewall may be deployed in software, such as a specialized application running on a single computer, or it may be a special device dedicated to protecting one or more computers.

Technically, the best solution is to have a computer with two or more network cards that isolate the different connected networks (or network segments), so that the firewall software on the machine (or if it is a special hardware) is responsible for connecting the network packets and determining which ones may or may not happen, and to which network.

12.3. Netfilter: iptables

The Linux core provides a filtering subsystem called Netfilter which is a set of functionalities for intercepting and managing network packets by providing packet filtering and NAT (in addition to other options). The main control command is `iptable` that allows us to perform the different rule configuration tasks that affect the filtering system, whether it is record generation, actions of pre- and post "routing"of packets, NAT, and port forwarding.

The `iptables` system manages different tables where the most used/ important ones are NAT and FILTER, which in turn contains different CHAINS, where the rules are associated. The FILTER table is for the filtering standards themselves, and the NAT table for address translation.

The `iptables -L` command lists the active rules (if *-t filter* or *-t nat* are not indicated, it lists the rules of the filter table by default) in each chain. The chains by default in FILTER are INPUT (for packets intended for local sockets), FORWARD (for packets being forwarded through this machine), and OUTPUT (for packets generated locally). For the NAT table, the default chains are PREROUTING (to modify packets before entering the machine), INPUT (to modify packets intended for local sockets), OUTPUT (to modify packets generated locally after routing), and POSTROUTING (to modify packets that will leave).

The typical configuration of the FILTER table is a series of rules that specify what is done within a given chain through an `iptables -A string -parameters -j action` command. The -A option adds the rule to the existing ones and -j indicates what to do with the package that can be *accept* (accept it), *reject* (reject it) or *drop* (delete it). The difference between *reject* and *drop* is that the latter sends a connection error not allowed while the first (*drop*)

does not, so if the first one is used, an attacker already knows that there is a service but that cannot access while the second one is considered safer since an attacker does not know if there is a service or not.

It should be noted that the order matters so the least restrictive rules must be placed at the beginning, since if a rule is first put that deletes the packages it will no longer be available for the next ones. There are a set of default rules (public rules) that apply when there is no specific rule for a given package and they can be ACCEPT or DENY, that is, if none of the package rules can be applied to the package and the default rule is ACCEPT, the package will be accepted and if the default rule is DENY, the package will be deleted. A default rule of DENY all would be:

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

which means that no package that does not meet any of the active rules will pass and that there must be one from both INPUT and OUTPUT for the same package. A policy of ACCEPTING all would be:

```
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -t nat -P PREROUTING ACCEPT
iptables -t nat -P POSTROUTING ACCEPT
```

In which if the package does not satisfy any of the subsequent active rules, it will end up accepting, so this type of configuration (less paranoid) ends with a last `iptable rule -A INPUT -s 0.0.0.0/0 -j DROP` (that is, when all the rules are passed and there is none that satisfies the package, it is deleted).

A typical strategy in environments that are not very hostile is a default policy of ACCEPT all, a set of rules that filter packets to services and end with a DROP rule to the rest of the packages. Examples of these Filter table rules could be:

```
iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP
iptables -A INPUT -s 10.0.0.0/8 -p tcp -dport 113 -j ACCEPT
iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset
iptables -A FORWARD -s 192.168.3.2 -d 192.168.10.5 -p tcp --dport 5432 -j ACCEPT
iptables -A FORWARD -s 192.168.10.5 -d 192.168.3.2 -p tcp --sport 5432 -j ACCEPTt
```

Where:

- Packages coming from 10.x.x.x to 192.168.1.2 are deleted.

- Tcp packages to port 113 from 10.x.x.x are accepted
- Reject tcp packages to port 113, issuing a tcp-reset response.
- A package in transit from 192.168.3.2 to 192.168.10.5 and to port 5432 is accepted (and in the opposite direction).

Regarding protocol and port names, the `iptable` system uses the information provided by the `/etc/services` and `/etc/protocols` files, and the information (port and protocol) can be specified numerically or by name (we must be careful, in this case, that the information in the files is correct and that they have not been modified, for example, by an attacker).

It is always recommended to install the `iptables-persistent` package (if available) that is responsible for saving/restoring the rules when the system is shut down/rebooted and if not, it can be done through the specific `iptables-save` and `iptables-restore` commands. Below is an example to load with `iptables-restore < name _file` (# indicates comments):

```
*filter
# Allow all loopback traffic (lo0) and deny the rest of 127/8
-A INPUT -i lo -j ACCEPT
-A INPUT ! -i lo -d 127.0.0.0/8 -j REJECT
# Accept all pre-established incoming connections
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# Accept all output traffic
-A OUTPUT -j ACCEPT
# Allow HTTP and HTTPS from anywhere
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
Allow SSH connections
# It normally uses port 22, check in file /etc/ssh/sshd_config.
-A INPUT -p tcp -m state --state NEW --dport 22 -j ACCEPT
# Reply to ping icmp
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
# Remove remaining incoming traffic.
-A INPUT -j DROP
-A FORWARD -j REJECT
COMMIT
```

In this example, ping, http, htpps, ssh incoming traffic is accepted and the remaining incoming traffic is blocked. When it comes to output traffic, everything is allowed out without restrictions. To test its functionality from another machine, run for example `nmap`, which will show us the open ports on the machine configured with iptables.

In the RHEL branch and its derivatives, the firewall by default is `firewallD` (although it is possible to install `iptables`). It provides dynamic control of the firewall and allows multiple zones to be defined to manage the various networks to which the system is connected. It allows us to simultaneously maintain permanent, runtime configurations and apply new rules without the need to restart the entire firewall. Some useful commands in this case that allow us to get status, supported services, enable a service, get zones and zones with enabled services:

```
firewall-cmd --state
firewall-cmd --get-services
firewall-cmd --add-service=http
firewall-cmd --get-zones
firewall-cmd --list-all-zones
```

In addition to command configuration, the *firewall-config* graphical tool is also available for simpler/guided configuration.

Other tools for configuring firewalls include:

- **fwbuilder** is a tool that allows building firewall rules graphically. It can be used in various operating systems (Fedora, Debian, OpenBSD, MacOS), with management of different types of firewalls (including `iptables`).

- **ufw** (Uncomplicated Firewall): as the name implies, it is intended to be a simple-to-use firewall, which can be found in the distributions of the Debian branch, based on CLI and which allows for a fairly simple use (but withstanding much of the possibilities of `iptables`). A small example session:

```
ufw allow ssh/tcp
ufw logging on
ufw enable
ufw status
```

A graphic package called Gufw is also available for its use.

12.4. Netfilter: nftables

NFtables is the evolution of `iptables` to define the next generation of firewalls for Linux (included in the kernel space, starting with version 3.13) with a new `nft` command (in user space) and an `iptable` support layer. Some notable differences are:

- Different, clearer syntax.

- Tables and chains are fully user configurable, unlike `iptables`, which is based on a predefined table structure.

- Greater control of protocol correspondence.

- Multiple actions in a single rule.

- Supported network protocols can be updated by making modifications to client applications, without the need to update the kernel, thus avoiding problems with frozen versions in certain distributions.

As for the paths that the packages follow, they continue to have the same structure (of hooks, in Netfilter terminology): when a package arrives at the system, it goes through a prerouting, then accessing a routing decision (is it an entry package for the system or forwarding to others?); if the system is forwarded (because it works as a router), the package will arrive at postrouting and exit the system, depending on the routing options. If it is the opposite case, and the package was for the system, it will move to `INPUT`, it will be processed by the system, which can generate output packages, `OUTPUT`, which after a *routing* decision (is it for the system itself or for external network?), will arrive at postrouting to exit the system.

The first step in *ntables* is to create the chains as they are not defined:

```
Create Table:
nft add table ip filter
List tables:
nft list tables ip
List chains associated with a table (initially they will not be created):
nft list table ip filter
Delete a table
nft delete table filter
Flush from a table (release table rules):
nft flush table ip filter
Add a base chain (related to a Netfilter Hook discussed above), this chain will see the Linux
TCP/IP stack packets, input and output examples, and third case a non-base chain.
With delete or flush instead of add, the rules are deleted (or flush):
nft add chain ip filter input { type filter hook input priority 0 \; }
nft add chain ip filter output { type filter hook output priority 0 \; }
nft add chain ip filter test

Rules: list them (options -n disable resolution names, -nn service resolution), a rule for a
particular port, a rule to a particular previous position within the chain, save rules of a
chain in a file, load them from filter-tble file
nft add rule filter output tcp dport ssh counter
nft add rule filter output position 8 ip daddr 127.0.0.8 drop
nft list table filter > filter-table
```

```
nft -f filter-table
```

See more information on nftables HOWTO documentation page.

13. Configuring servers

In this section, some of the typical GNU/Linux services will be installed, such as an HTTP/HTTPS server and proxy servers. The range of services that can be provided with GNU/Linux is very wide, but for space reasons it will only be limited to the installation and configuration of the most interesting options of these two services. Interested readers can consult a chapter dedicated only to servers, which although it is outdated, the main concepts and methodology have not changed:

CAT: https://openaccess.uoc.edu/handle/10609/60685.

SPA: https://openaccess.uoc.edu/handle/10609/60686

13.1. World Wide Web (httpd)

Apache is one of the most popular servers with HTTP (*HyperText Transfer Protocol*) capabilities. Apache has a modular design and supports dynamic extensions of modules during execution. It is highly configurable in terms of the number of servers and modules available and supports various authentication mechanisms, access control, metafiles, caching proxy, virtual servers, etc. With the inclusion of modules it is possible to have PHP, Perl, Java Servlets, SSL and other extensions that can be consulted on the developer website.

Apache is designed to run as a standalone daemon process, creating a set of child processes that will handle input requests. It can also run as the daemon Internet via `inetd` or `xinetd`, so it will start every time a request is received, but it is not recommended. Server configuration can be extremely complex depending on needs (see documentation), however, a minimum acceptable configuration will be seen here. Its installation is simple, for example in Debian or derivatives: `apt-get install apache2 apache2-doc apache2-utils`

The server configuration will be set to */etc/apache2* and by default the *RootDirectory* to */var/www/html* (directory where html files that Apache will serve will be stored). After installation, it will be started and its operation can be checked by calling through a browser (it will show the known **It works!** page). One of the common commands for managing the service is `apache2ctl` (although it can also be managed with `systemctl` but with fewer options). This command accepts typical *start|stop|restart|graceful|graceful-*

stop|configtest|status|fullstatus|help service management options and a very interesting one is *configtest*, which validates the configuration that was made before the service was started in case of errors.

While all default parameters have functional values, it should be noted that the `ServerName` variable is not defined in the default installation and should be configured in */etc/apache2/apache2.conf* or in the site configuration files within the *Virtualhost* tag as *ServerName srv.nteum.org*.

The Apache2 configuration in Debian (and derivatives) is slightly different from the overall distribution as it tries to make it as easy as possible to configure the server for modules, virtual hosts, and configuration policies (however, equivalencies with other distributions can be quickly found). The main files found in the */etc/apache2/* directory are *apache2.conf, ports.conf*, and six *mods-available|mods-enabled, sites-available|sites-enabled*, and *conf-available|conf-enabled* directories. For additional information please see the */usr/share/doc/apache2* directory. The most important configuration files are:

- *apache2.conf* is the main configuration file where the server features are defined functionally and the corresponding configuration files (*ports, conf.d, sites-enabled*) are called.

- *ports.conf* defines the ports where the incoming connections will be served, and which of these are used on *virtual* hosts (80 for http and 443 for https by default).

- The configuration files in *mods-enabled/* and *sites-enabled/* are for the active sites and modules that will be uploaded to the server. These settings are enabled by creating a symbolic link from the respective **-available/* directories using the `2enmod/a2dismod`, `a2ensite/a2dissite` commands.

- The *conf-available* and *conf-enabled* directories are configurations for certain aspects supported by the server or other administrator-added packets that work in conjunction with Apache (they all have *.conf extension*).

- For the default settings in these directories to be effective, Apache2 will need to be managed via `systemctl` or `apache2ctl`.

- The *envvars* file is the one that contains the definition of the environment variables and it is necessary to basically modify USER/GROUP that will be the one used for execution and for obtaining the permissions. The user *www-data* and the group *www-data* are created by default (although they can be changed).

Apache may also need to integrate various modules depending on the technology it supports and therefore the corresponding libraries/packages should be added, for example:

- Perl: `apt-get install libapache2-mod-perl2`
- Python: `apt-get install libapache2-mod-python`
- PHP: `apt-get install php7.4 php7.4-cgi libapache2-mod-php7.4 php7.4-common`
- PHP with MySQL: `apt-get install php7.4-mysql`

13.2. Virtual servers

Virtual servers are isolated sites that will each be served independently of each other with their own files and configuration. First, the default site must be disabled with `a2dissite 000-default.conf`. The sites we will create will be remix.world and lucix.world which will have two configuration files in the directory */etc/apache2/sites-available/*.

Contents of file /etc/apache2/sites-available/remix.world.conf

```
<VirtualHost remix.world:80>
            ServerAdmin adminp@localhost
            ServerName remix.world
            ServerAlias www.remix.world
            DocumentRoot /var/www/html/remix/
            ErrorLog ${APACHE_LOG_DIR}/remix-error.log
            CustomLog ${APACHE_LOG_DIR}/remix-access.log combined
</VirtualHost>
```

Content of file /etc/apache2/sites-available/lucix.world.conf

```
<VirtualHost lucix.world:80>
            ServerAdmin adminp@localhost
            ServerName lucix.world
            ServerAlias www.lucix.world
            DocumentRoot /var/www/html/lucix/
            ErrorLog ${APACHE_LOG_DIR}/lucix-error.log
            CustomLog ${APACHE_LOG_DIR}/lucix-access.log combined
</VirtualHost>
```

This setting is basic and the student should refer to the developer web page for detailed information. As can be seen, the root directories for each domain will be in */var/www/remix|lucix* and the log files in /errors/accesses in

/var/log/apache2/mmm-error.log and *var/log/apache2/nnnn-access.log/*. To create the directories `mkdir -p /var/www/html/remix`; `mkdir -p /var/www/html/lucix` and into which a *index.html* could be placed with some identification that would show which domain is being loaded, for example for remix.world:

```
<html><body><h1>REMIX: It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

And the same for lucix.world but changing the line at <h1></h1>. No action should be taken for the logs as the directory */var/log/apache2* already exists and the files will be created by the server. Finally, the sites must be activated (creating the link from *sites-available* to *sites-enabled*) with

`a2ensite remix.world.conf`; `a2ensite lucix.world.conf` and restart Apache2 with `systemctl restart apache2`. Since domains are not available in a primary DNS, we can edit */etc/hosts* and add two lines for the server IP (e.g., 192.168.1.37) such as:

```
192.168.1.37 remix.world
192.168.1.37 lucix.world
```

Then from a browser on the same machine (as otherwise no domain/IP will be visible) or from another machine on the same network to which the */etc/hosts* has been modified, the URL remix.world can be entered and the result will be the *index.html* display that will show: **REMIX: It works!**

One of the advantages of Apache is that functionality can be added through specialized modules and found at */etc/apache2/mods-available/*.

For example, `apt-cache search libapache2*` can be done to get the list of modules available for Apache, and to install `apt-get install [module-name]` which will be available for use (remember that some additional configuration in the site files may be necessary). With `ls -al /etc/apache2/mods-available/` we can look at the available ones and install them with `a2enmod [module-name]`. To list the loaded modules, we can execute `apache2ctl -M` which will list the dynamically loaded ones with shared and those that are compiled with the server with static (these can also be obtained with `apache2 -l`). Modules in the *mods-available* directory have *.load* (indicates the library to load) and *.conf* (additional module configuration)

extensions but when using the `a2enmod` command, only the name of the module without extension should be indicated. To disable a `a2dismod module[module-name]`.

As a sample of these properties, a secure site will be configured with encrypted communications over HTTPS and under the remix.world domain, but it will be redirected to the */var/www/remix.ssl* directory. First, a certificate (self-signed) will be created for the site with:

```
make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/ssl/private/remix.crt
```

indicating the domain to be validated (remix.world in this case) – just enter the domain – and as a second IP validator:IP_SERVER. Then the SSL module must be activated with `a2enmod ssl`, create the */var/www/remix.ssl* directory and modify the index.html as it was done with the previous ones. The site settings are then modified (the default setting can be used by modifying it):

```
cd /etc/apache2/sites-available; cp default-ssl remix.world.ssl.conf
```

The *remix.world.ssl.conf* file is then edited (only the main/modified lines are displayed):

```
<VirtualHost _default_:443>
            ...
            DocumentRoot /var/www/remix.ssl
            ...
            ErrorLog ${APACHE_LOG_DIR}/remix.world.ssl_error.log
            CustomLog ${APACHE_LOG_DIR}/remix.world.ssl_access.log combined
            ...
            SSLEngine on
            SSLCertificateFile /etc/ssl/private/remix.crt
            #SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
            ...
</VirtualHost>
```

Finally, the site (`a2ensite remix.world.ssl.conf`) has to be activated, apache2 (`systemctl restart apache2`) restarted, and from the browser, *https://remix.world* should be made which, as the certificate is self-signed, will show a warning and after accepting the *Autosigned* option, the index.html page will be obtained, which shows **SSL - REMIX: It works!** If a site with DNS and domain is available, a free certificate can be obtained from Cerboot.

An interesting aspect is to allow authenticated access to certain directories through the Apache authentication module. To do this, in the VirtualHost configuration file we must include:

```
<Directory /var/www/html/auth>
```

```
            AuthType Basic
            AuthName "Basic Authentication"
            AuthUserFile /etc/apache2/.htpasswd
            require valid-user
</Directory>
```

To create the user, `htpasswd -c /etc/apache2/htpasswd adminp` is run which will request the passwd for this user and store it in the indicated file. It can then be put as a URL *http://remix.world/auth/* and it will ask for the user (adminp) and passwd that has been entered previously and the index.html which is located at */var/www/html/auth*. will be visible. If the passwd is erroneous or if Cancel is done, it will indicate with an *Authorization Required* message, preventing access.

There is another mode of authentication through the server but assigning it to the Operating System (when the user is a valid user of it). If we want to enable these users, we need to link them to Apache through the PAM authentication system. To do this, the following will be installed: `apt-get install libapache2-mod-authnz-external pwauth`

A */etc/apache2/sites-available/auth-pam.conf* configuration file similar to the above will then be created with the following content:

```
<VirtualHost authpam.nteum.org:80>
            ServerAdmin webmaster@localhost
            DocumentRoot /var/www/html/auth-pam

            ErrorLog ${APACHE_LOG_DIR}/error.log
            CustomLog ${APACHE_LOG_DIR}/access.log combined

            AddExternalAuth pwauth /usr/sbin/pwauth
            SetExternalAuthMethod pwauth pipe
            <Directory /var/www/html/auth-pam>
                        AuthType Basic
                        AuthName "PAM Authentication"
                        AuthBasicProvider external
                        AuthExternal pwauth
                        require valid-user
            </Directory>

</VirtualHost>
```

A `mkdir /var/www/html/auth-pam` directory will be created and among which there can be an *index.html* file with an identifier that is accessing PAM. Finally, the `a2ensite auth-pam` site will be enabled, the `a2enmod authnz` *`module`* will be enabled, and the `systemctl restart apache2` server is restarted. If a line is inserted into */etc/hosts* with the *authpam.nteum.org*

domain with the server IP, http://authpam.nteum.org/*auth-pam/* will already be accessible after entering a local user and the *passwd*. Authentication is performed with the integration of the *authnz-external* module and the `pwauth` command.

13.3. Proxies

The function of a proxy is to play the role of intermediary in the requests that the client requests from another server, that is, the proxy server knows both resources and puts them in contact without one knowing the other. As a link between requests and services, it allows different actions such as access control, traffic registration/control (including blocking), improved transaction performance (intermediate storage), anonymity in communication, among others. As an intermediary, there are various opinions/controversies regarding the use of proxies in terms of safety and anonymity. That is why it is necessary to take good care of its configuration and the service it provides so that it cannot be used for any purpose other than that for which it has been designed.

Although different types of proxies can be found, usually differentiated by the protocol/application they manage (web, ftp, ARP, dns, . . .), the one used in web services is probably the most common. If the proxy is connected to and from the Internet, it is considered an open proxy and its function is to forward all packets it receives allowing the client IP to be hidden to the server, which is a form of anonymity (weak). There are extensive lists of open proxies (we just need to consult the *https://www.google.es/search?q=open+proxies+list* query) but no one can attest to the anonymity they allow. If it is desired to have real anonymity, networks such as Tor (The Onion Router) must be used, which allows guarantees of anonymity using encrypted communications (multiple times) and that pass through a worldwide network of servers (voluntary) obtaining the anonymity of the communication and preventing it from being monitored or supervised. Another similar network that allows for anonymity is the I2P network of the project Invisible Internet Project with similar objectives to Tor but which is not as widespread.

A common question is the difference between a proxy and a firewall (acting as NAT). Generally, when a proxy is specified it is referring to a layer 7 application of the OSI model while NAT refers to layer 3 of that model. Because NAT operates in layer 3, it uses fewer resources, but is also less flexible than in layer 7, as it only acts on packet addresses and not as the proxy might do on content (in layer 7). It is common in GNU/Linux systems for NAT to be performed with IPtables (firewall) while different servers are used as a proxy; for example, Apache, Squid, Nginx, Varnishm, among others, are used for http/https/ftp.

13.4. Apache as reverse proxy and with load balancing

Apache is a very versatile and efficient http server that has a large number of modules that extend its functionality, including the proxy module. In this section, the configuration will be analyzed first as a reverse proxy to an internal server. This reverse proxy service will then be expanded to balance the load to more than one server by redirecting requests based on different policies. Apache supports different proxy modules, among which we can mention: mod_proxy (multi-protocol proxy), mod_http (http support for proxy), mod_cache, mod_proxy_html (rewrite HTML links to ensure they work outside the proxy), mod_proxy_balancer (balance for reverse proxy), lbmethod_bytraffic (for traffic load balancing) lbmethod_byrequests (request load balancing), among others. The steps to be followed will be:

1) Verify that the modules described above are available in */etc/apache2/mods-available* (in Debian and derivatives they are usually already installed and if not, something such as: `apt-get install libapache2-mod-proxy-*)` must be executed

2) Enable modules: `a2enmod proxy proxy_http` (can be verified with `apache2ctl -M`).

3) A new host is created in */etc/hosts* (e.g. *192.168.1.37 proxy.nteum.org proxy*).

4) A virtualhost is created at */etc/apache2/sites-avalaible/proxyr.conf*:

```
<VirtualHost proxy.nteum.org:80>
            ServerName proxy.nteum.org
            ServerAdmin webmaster@localhost
            DocumentRoot /var/www/html
            ErrorLog ${APACHE_LOG_DIR}/error.log
            CustomLog ${APACHE_LOG_DIR}/access.log combined

            ProxyRequests Off
            ProxyPreserveHost On
            ProxyPass / http://mig.nteum.org/
            ProxyPassReverse / http://mig.nteum.org/
</VirtualHost>
```

Where:

- ServerName must be set to */etc/hosts*, and indicates how we will call the input server (it has also been put into VirtualHost).

- ProxyRequests Off prevents this from being used as an open proxy, that is, users can go to the proxy and from there to any other address and it is very important to leave it disabled to avoid security or even legal issues.

- *ProxyPreserveHost On* allows the jump from the proxy server to the backend server to be transparent to the user (if it was not enabled, the user would go to *http://proxy.nteum.org* but immediately would see how the address changes to *http://mig.nteum.org*, which is the internal server –backend–) and if the backend server is also not visible from the external network the client would see an error.

- *ProxyPass* and *ProxyPassReverse* manage the jump and return from frontend to backend server (mig.nteum.org) The domain of the backend server can be changed by the server lp if this domain is not defined in */etc/hosts* of the proxy server.

5) The configuration of the new site is enabled (`a2ensite proxyr.conf`) and the service restarts (`systemctl restart apache2`). We must also have an *apache2* server running on the backend with a different page than the proxy to verify that it is accessed when *http://proxy.nteum.org* is entered in a browser.

One of the interesting aspects of the web service is being able to load balance requests on different servers to avoid the "bottleneck" effect on the service and improve response time and increase the number of requests handled per unit of time. This can be done using specific hardware or a reverse proxy (frontend) that distributes the load to a set of servers (backends) in accordance with a given policy. Apache has an additional module to the proxy (mod_proxy_balance) that allows load balancing on a web server set and different modules to implement policies (lbmethod_byrequests, lbmethod_bytraffic lbmethod_bybusyness, lbmethod_heartbeat).

To configure this module, we must:

- Load the modules:

proxy, proxy_balancer proxy_connect proxy_html proxy_http lbmethod_byrequests lbmethod_bytraffic lbmethod_bybusyness lbmethod_heartbeat status

(to view `apache2ctl -M` loaded modules)

- Create a *virtualhost*: `vi /etc/apache2/sites-available/proxy-bal.conf`:

```
<VirtualHost proxy.nteum.org:80>
            ProxyRequests off
            ServerName proxy.nteum.org
```

```
            DocumentRoot /var/www/html

            <Proxy balancer://mycluster>
                        BalancerMember http://172.16.1.2:80
                        BalancerMember http://172.16.1.3:80
                        Options Indexes FollowSymlinks Multiviews
                        AllowOverride None
                        Order Allow,Deny
                        Allow from all
                        #ProxySet lbmethod=bytraffic
                        ProxySet lbmethod=byrequests
            </Proxy>
            # Enable Balancer Manager
            <Location /balancer-manager>
                        SetHandler balancer-manager
                        Order deny,allow
                        Allow from all
            </Location>
            ProxyPass /balancer-manager !
            ProxyPass / balance://mycluster/
            ProxyPassReverse / balancer://mycluster
            ProxyPass / http://172.16.1.2
            ProxyPassReverse / http://172.16.1.2
            ProxyPass / http://172.16.1.3
            ProxyPassReverse / http://172.16.1.3
</VirtualHost>
```

Here the IPs of the nodes that will act as Backend have been put but they could be given as names in */etc/hosts*.

The balance manager is a tool that integrates the module and will allow us to easily see the statistics of the module activity and some modifications (simple as well). That is why requests to *http://proxy.nteum.org/balancer-manager* should not be redirected and addressed by the proxy.

The configuration includes the following elements:

- The Proxy Balancer section: where the balancer is identified.
- *BalancerMember*: each one of the backend IPs.
- *ProxySet lbmethod=byrequests|bytraffic*: the balancing policy.

To enable the configuration of the new site (`a2ensite proxy-bal.conf`) and restart the service (`systemctl restart apache2`). We also need to have the two servers (172.16.1.2 and 172.16.1.3) running on the backend with Apache2 executing and with a different page than the proxy to verify

that it is accessed when *http://proxy.nteum.org* is entered in a browser and the page is reloaded repeatedly (we will see how the page changes depending on the *backend* server that *serves* it). For more information, we can view the balancer statistics at *http://proxy.nteum.org/balancer-manager* and change the parameters to suit the load needs (we can use the load tools mentioned to analyze Apache).

13.5. Apache as Forward Proxy and Proxy cache

To configure Apache as Forward Proxy we must:

- Load the modules (if they are not loaded, and check `apache2ctl -M`): `a2enmod proxy proxy_http`

- Add to the configuration of a site */etc/apache2/sites-available/proxy-f.conf* with the following configuration:

```
<VirtualHost *:80>
           ServerName www.nteum.org
           ServerAdmin webmaster@localhost
           DocumentRoot /var/www/html
           ErrorLog ${APACHE_LOG_DIR}/error.log
           CustomLog ${APACHE_LOG_DIR}/access.log combined

           ProxyRequests On
           <Proxy "*">
                      Requires ip 172.16.
           </Proxy>
           ProxyBlock "www.site1.com" "www.site2.com"
           #ProxyBlock *

</VirtualHost>
```

- Enable the site (`a2ensite proxy-f.conf`) and restart the service (`systemctl restart apache2`). Clients must then be modified to solicit the requests from the *proxy* server, for example, in Firefox, we must put *Preferences->Network->Settings* and in *Proxy*, put the server IP, for example, 192.168.1.37 and port 80.

- To enable the proxy cache capability, we should add to the *virtualhost*:

```
CacheEnable disk /
CacheRoot "/tmp/cache"
CacheQuickHandler off
CacheLock on
CacheLockPath /tmp/mod_cache-lock
CacheLockMaxAge 5
```

```
CacheHeader On
```

- The ProxyBlock sentence will allow an access control policy and prevent clients from accessing the indicated websites (blocking).

We must then add the `a2enmod cache cache_disk modules`, enable the site if it is not, and restart Apache. From the browser and with the Web Developer options we can see how page load times are reduced when they are reloaded for a second time (Network tab).

There are other aspects of cache over Apache that have not been treated as File Caching to accelerate access to the files served by Apache and Key-Value Caching used by SSL and authentication caching that can be consulted on the developer's web.

It is also important to mention that at the enterprise level there are well-known proxies servers such as *Squid* or *HAProxy*. Squid improves the performance of business and private Internet connections by caching recurring requests to web servers and DNS, speeding up access to a given web server, or adding security by filtering traffic and although it is mainly oriented towards HTTP and HTTPS, it also supports other protocols.

HAProxy is a very fast and reliable reverse proxy that offers high availability, load balancing and proxy for TCP and HTTP based applications and is well suited for very high traffic websites. This server has become the de facto proxy balancer and is included in the repositories of most Linux distributions with very active development (2 versions per year) and new functionalities in each major revision.

Activities

1. For RPM packages, how would some of the following tasks be done?

- Know which package installed a command.
- Get the description of the package that a command installed.
- Delete a package whose full name we do not know.
- Show all files that were in the same package as a given file.

2. Perform the same task as in the previous activity, but for Debian packages, using the APT tools.

3. Update a Debian (or Fedora) distribution.

4. Install in the available distribution some generic administration tool, such as *Webmin*. What does it offer? Is it easy to understand the executed tasks and the effects they cause?

5. The swap space allows to complement the physical memory to have more virtual memory. Depending on the sizes of physical memory and swap, can memory run out? Can it be solved otherwise, other than by adding more physical memory?

6. Assume we have a system with two Linux partitions, one is '/' and the other is swap. How would we fix the fact that users' accounts ran out of disk space? And if we had an isolated partition /home that was running low, how would we fix it?

7. Analyze the systemd system. What services and groupings does it have predefined? Is there compatibility with the *SystemV* init (*sysvinit*)? How are services managed? How do we change target?

8. Examine default GNU/Linux system settings for non-interactive tasks by *cron* (or systemd-cron). What tasks are they and when are they going to be executed? How are new user tasks added?

9. Define the following network scenarios and their configurations:

- Isolated machine.
- Small local network (4 machines, 1 gateway).
- 2 segmented local networks (2 sets of 2 machines, one router each and one general gateway).
- 2 interconnected local networks (two sets of 2 machines + gateway each).
- A machine with 2 interfaces, one connected to the Internet with NAT to a router and another to a private network1, a second machine with two interfaces connected to a private network1 and the other to a private network2, a third machine connected to a private network2.
- 2 machines connected over a virtual private network.

Indicate the advantages/disadvantages of each configuration, the type of infrastructure they are suitable for, and what relevant parameters are needed (for both IPv4 and IPv6).

10. Using virtual machines, perform the configuration and monitoring and connection test (for example, `ping`, `dig` and `apt-get update`) of the proposals from the previous point and on each machine of the proposed architecture.

11. Perform the above experiments on IPv6 using `ping6` and one of the tunnels mentioned in the Network/IPV6 section to show connectivity to the Internet.

12. Configure a DNS server with `dnsmasq` and a proprietary domain.

13. Configure a NIS server/client with two machines by exporting the server user directories by NFS.

14. Configure an SSH server to access from another machine without a password.

Bibliography

All links were last accessed in May 2022.

[Apa] Apache Server. https://www.apache.org/

[Apachessl] SSL/TLS Strong Encryption: An Introduction. https://httpd.apache.org/docs/2.4/ssl/ssl_intro.html

[Com] Comer, Douglas. Internetworking with TCP/IP Addison-Wesley. 2013.

[Deb] Debian.org. Debian Home. http://www.debian.org

[Coo] Cooper, M. (2014). "Advanced Bash Scripting Guide".*The Linux Documentation Project*(guides). https://tldp.org/LDP/abs/html/

[Deb] Debian Package. https://wiki.debian.org/deb

[DNSMQ] DNSMasq. DNS forwarder and DHCP server. https://wiki.debian.org/HowTo/dnsmasq

[Fri] Frisch, A. (2002). *Essential System Administration*. O'Reilly. UOC Library.

[FHS] Filesystem Hierarchy Standard. https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard, http://www.pathname.com/fhs

[Gnu] GnuPG. https://gnupg.org/

[GRD] Debian. "Debian Chapter 5 Reference Guide. Network configuration". https://www.debian.org/doc/manuals/debian-reference/index.es.html

[HAP] HAProxy. http://www.haproxy.org/

[HeMa] Hertzog, R.; More, R. "The Debian Administrator's Handbook". https://debian-handbook.info/browse/stable/

[IET] IETF. Request For Comment repository developed by Internet Engineering Task Force (IETF). https://www.rfc-editor.org/rfc/

[LSB] Linux Standard Base specification. https://en.wikipedia.org/wiki/Linux_Standard_Base

[Mik] BASH Programming - Introduction HOW-TO. Mike G. https://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

[Mik] Mike, G. "BASH Programming - Introduction HOWTO".*The Linux Documentation Project*. https://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

[Mou01] Mourani, Gerhard. Securing and Optimizing Linux: The Ultimate Solution. Open Network Architecture, Inc. 2001. https://tldp.org/LDP/solrhe/Securing-Optimizing-Linux-The-Ultimate-Solution-v2.0.pdf

[Muh] Transparent Multi-hop SSH. http://sshmenu.sourceforge.net/articles/transparent-mulithop.html

[Nem] Nemeth, Evi.; Snyder, Garth, author; Hein, Trent, Unix and Linux System Administration Handbook. O'Reilly. 2010. UOC Library.

[OVPN] OpenVPN. "OpenVPN: 2x How To". https://openvpn.net/community-resources/how-to/

[Qui] Quigley, E. (2001). Linux shells by Example. Prentice Hall. UOC Library.

[RPM] RPM Package Manager. https://rpm.org/index.html

[Squ] Squid. http://www.squid-cache.org/

[Soy] Linux administration: a beginner's guide. Soyinka, Wale. McGraw Hills/O'Reilly. 2016. UOC Library.

[Tor]. Tor Project. https://www.torproject.org/