Redes1 - Lab

Jesús Blanco, Cleto Martín, José Luis Segura

Mar 22, 2026

CONTENTS

Esta documentación está disponible en español

In this documentation you will find the required material to the practices of the subject **Computer Networks 1**, for the academic year 2025/2026. The *labs* will be published as the sessions progress.

This documentation is also available in PDF format.

Teachers

Jesús Blanco A1 - Monday 5.00pm

A2 - Monday 6.30pm

Javier Romero B1 - Tuesday 1.00pm

Inocente Sánchez B2 - Thursday 10.00am

D1 - Thursday 1.00pm

Cristina Bolaños C1 - Tuesday 10.00am

C2 - Wednesday 10.00am

General Information

The practical sessions are designed to reinforce and extend the knowledge acquired in the theory lessons. Many of the materials and issues that will be covered during the sessions will require practice and further study.

The first sessions will be aimed at familiarization with the environment and will *not be considered* for your mark. Later on, we will focus on computer networking issues, which will *be considered* for the lab mark and will require the resolution of exercises (during the lab session).

The laboratory environment will be based on the GNU/Linux operating system. To learn more about about it, take a look at our *virtual machine* section.

CHAPTER

ONE

VIRTUAL MACHINE

The laboratory practices will be performed on an operating system called Debian, which is one of the most famous and oldest GNU/Linux distribution. It is used in a large number of environments and applications, mostly in network infrastructure.

> **Note**
>
> It is possible to use any other GNU/Linux distribution like Ubuntu o Fedora. We *recommend* you to use the provided virtual machine. So you can follow the examples in the same environment that they will be explained.

To make easier its distribution and installation, we have prepared a virtual machine (VM) for VirtualBox. So that it is possible to run a complete machine with our pre-configured Debian for the labs, without the need to install it natively.

This VM is available on the *lab computers*, and we will use them for the practical sessions. You can also install it on *your own PC*, so you can have the same environment to practice at home or use it in the lab, if your assigned computer is not working for any reason. If you already have some of GNU/Linux distribution installed on your PC, you can probably use it directly.

1.1 Lab Computers

As a general rule, the laboratory computers are the ones that will be used during the practical sessions.

To start the virtual machine, do the following steps:

- Turn on the computer and wait until the operating system is booted.
- Once started, a menu will appear with different virtual machines to boot.
- Select, **in this order**:
  1. `Vol O`
  2. `Red de Laboratorio`. Note that the prompt shows: *IP Pública*.
  3. `Ejecutar`
- Wait a few seconds and the virtual machine will start booting.
- Once booted, it will ask for the user name and password.

> **Note**
>
> User: `alumno`

Password: `alumno`

1.2 Installation on your PC

With these instructions you will boot the lab virtual machine on your PC, without having to do a native installation. They are aimed primarily at Windows or MacOS users who do not have a GNU/Linux distribution installed in some form on their PC.

1.2.1 Requirements

To be able to use the lab virtual machine you will need to:

- Have at least **10GB** of free disk space.
- Have VirtualBox installed.
- Have **virtualization support enabled**. It is possible that your PC may already have it, but if the virtual machine loading fails, you may need to activate it. It is usually activated in the BIOS of your PC and the way to do it depends on the PC manufacturer. We recommend you to look for your PC model and how to enable virtualization support to activate it (or make sure you already have it enabled).

Optionally, and if you want to use your PC during the lab sessions, your PC will need to have a *wired* network interface. If your PC only has a wireless interface, you can buy a network interface with USB connection.

1.2.2 Procedure

1. Download the OVA file that contains the virutal machine.
2. Open VirtualBox.
3. Go to `File` -> `Import`
4. Select the previous downloaded OVA file in the `File` box and press `Next`.
5. In the next screen, click on `Import`.
6. After a few seconds,the new virtual machine named `redes1` will appear in the left panel.

Important

Before launching the virtual machine for the first time, the network interface must be correctly configured:

1. Select the virtual machine from the left panel and press `Configuration`.
2. In the `Network` section, change the `NAT` value to `Bridge`, in the `Adapter 1` tab. And in the advanced options, change the MAC address to a randomly generated one by pressing the refresh button.
3. Finish by clicking `OK`.

This procedure only needs to be done before running the virtual machine for the first time.

The virtual machine is now ready to be used. To start it, just double click on it in the left panel and it will start running.

1.3 Language configuration

The virtual machine is in *Spanish* (default configuration). But you can change it to *English* by executing this command in the terminal:

```
$ sudo localectl set-locale en_GB.UTF-8
```

Also, you can configure the keyboard layout in the virtual machine to adjust it to your real keyboard. The keyboard layout in the virtual machine is configured in Spanis, so keep it if you work with a Spanish keyboard. You can change the virtual machine to have English keyboard layout with the following commands:

```
$ sudo localectl set-keymap en
$ sudo localectl set-x11-keymap en
```

Note that other keyboard layouts are also possible. Just use the one you want instead of `en`.

Finally, reeboot the virtual machine.

CHAPTER

TWO

LABS

The laboratories will be published as the lab sessions progress. Here you have a list of the currently available ones:

2.1 Lab 1 - Command Line I

A terminal is the operating system software that allows users to enter data into the computer and to obtain the output of the programs.

In every GNU/Linux installation (and in general of any operating system) there is a terminal application or a terminal graphical emulator.

In our reference operating system, we can use `LXTerminal`, available at: *Aplications -> System Tools*.

2.1.1 What is a command?

A command is a program that can be executed from the terminal. It accepts a series of arguments and either produces a specific output or causes a specific change in the system.

Command structure

A command is comprised of its own name, arguments and, optionally, options:

```
$ command [--option=value] [-f] [argument1 argument2 ...]
```

- `command` is the command name. We will see many examples later.
- `--option` and `-f` are the options. Options usually have 2 versions: the long version, which is usually prefixes with two hyphens; and the short version, which is only prefixes with one hyphen. Options modify the usual mode of operation of a command.
- `argument1` and `argument2` are the arguments. In many cases, it is not necessary to specify them because they take default values.

Usually, all commands have the `--help` and `-h` options, which allows to show the command help, where the different options and arguments the command accepts are explained.

Reading the Manual

Although as explained above, all commands can provide help on their own options and arguments, sometimes much more information is needed since some commands may require configuration files, use environment variables, produce output files and so on.

There is a specific command to request more information about other commands: `man`.

The terminal manual provides access to a full help page about the requested command, with the different options, arguments or their format. It also includes if the program uses environment variables, information on how to report bugs or the program's license.

Navigating through the pages of the manual can be counterintuitive at first. Here a few tricks:

- `Up Arrow`, `Down Arrow`, `Page Up`, `Page Down` allow to scroll up and scroll down the page.
- `q` ends the execution of the command `man`.
- `/` allows to search for a string in the current document. Once in search mode, you can use `n` or `N` to search down or up in the document, respectively.

For example, open the manual page for the `date` command and find how to specify the output format of the command:

1. Execute `man date`.
2. Press the `/` key.
3. Type in the string to search: *format*.
4. Press `n` until find the section where the format is explained

> **Note**
>
> These keyboard shortcuts explained for navigating through the manual pages are also valid for navigating through any file using the `less` command, which will be discussed later.

2.1.2 Getting familiar with the file system

The file system is the part of the operating system that allows to interact with the storage devices on our machine.

The first directory to know in GNU/Linux is the *root*: `/`. A storage device is usually mounted in this directory, usually a partition of a hard disk. Within this directory there are usually the following:

- `/boot`: contains the boot configuration as well as the kernel images.
- `/dev`: contains a complete virtual structure of directories and files that represent the different devices present on the machine.
- `/etc`: the configuration of the installed services in the system.
- `/home`: contains the personal directories of users. In the provided virtual machine you can find `/home/alumno`, which is where you will work.
- `/media` and `/mnt`: paths where, depending on the distribution, temporary devices such as USB sticks or external hard disks are usually mounted.
- `/opt`: path where third-party applications are installed.

- `/proc`: virtual structure of directories and files with information related to the running programs.
- `/root`: personal directory of the `root` user, which is the system administrator.
- `/run`: virtual structure of directories and files with temporary data from running processes.
- `/srv`: directory where files that will be served to other systems are stored (in disuse).
- `/sys`: virtual structure with files used by the system itself.
- `/tmp`: temporary directory. Depending on the configuration, it can be volatile (lost after each reboot) or persistent.
- `/usr`: directory where all operating system programs are installed.
- `/var`: path where all variable files, such as databases, log files or file caches, are stored.

> **Note**
>
> The term "virtual structure" refers to a whole hierarchy of files and directories that are created in the system at a logical level, but do not represent directories or files stored on any hard disk or other storage device.

Working with the file system

On many occasions we need to consult the content of files, modify them, visit other directories, execute commands in different paths… For this, we must know the basic commands to make queries to the file system, change the working directory, create, modify and delete files, copy or move them to different directories.

- `pwd`: shows the current working directory.
- `ls`: lists the files and directories. If we pass a path, it will show the list corresponding to that path.
- `cat`: shows the content of a given file.
- `cd PATH`: allows to change the current working directory. If we do not pass a path, the personal user directory will be set. If instead of a path, we use the `-` character, it allows to return to the previous working directory.
- `mkdir DIRECTORY`: creates a new directory.
- `rmdir DIRECTORY`: deletes a directory (as long as it is empty).
- `rm PATH`: deletes a file.
  - With `-r` it allows to delete a directory with its subdirectories and files recursively.
- `touch FILE`: creates an empty file or updates its access date if it already exits.
- `cp ORIGIN DESTINATION`: copy a file to another path or directory.
- `mv ORIGIN DESTINATION`: moves a file to another path or directory (can be used to rename files).

Absolute and relative paths

When we talk about file system paths, we can refer to them in an absolute way, giving the complete path to the file or directory, or in a relative way, where we give a part of the path only, relative to the current directory.

And what is the current directory? When we are in a terminal, we can query it using the `pwd` command. When we open a terminal, the initial working directory is the personal user directory (or "home"). For example, `/home/alumno`.

Whenever you specify a path whose first character is a `/` it is an absolute path. When it starts with any other character, it is a relative path to the current directory.

There are also some special directory specifiers:

- ~: refers to the personal user directory (or "home").
- .: refers to the current working directory.
- ..: refers to the "parent" directory of the current directory.

For example, if we open a terminal, our working directory will be `/home/alumno`. The absolute path `/var/log` and the relative paths `../../var/log` or `~/../../var/log` are equivalent and refer to the same directory.

> **Note**
>
> Pressing the `Tab`, the terminal will automatically complete the name of your command or path. In case more than one option exists, pressing the `Tab` twice all possible options for completion will be shown.

Permissions

In every operating system, files in the file system are assigned a series of permissions that indicate to the system which users can carry out the different operations on each file or directory.

In GNU/Linux, the permissions are divided into 3: **r**ead, **w**rite and e**x**ecution. And each permission can be defined differently for 3 different user "roles": the user who owns the file, users who are members of the group that owns the file, and all other users (neither the owner nor members of the group that owns the file).

To query the file permissions, you can use the `ls -l` command (see the manual for the full meaning of its output).

Run in your terminal `ls -l /etc/hosts`:

```
$ ls -l /etc/hosts
-rw-r--r--. 1 root root 185 Dec 16 17:00 /etc/hosts
```

In this case, the first character indicates that it is not a special type of file; the next 3 characters (`rw-`) indicate the permissions assigned to the user who owns the file; the next 3 (`r--`), the permissions of the owner group and the next 3 (also `r--`) the permissions for other users. After that, we can also see the name of the user and the owner group (user `root` and group `root`).

In summary, this file only has read and write permissions for the `root` user and read permissions for all other users, whether they are in the `root` group or not.

The way to modify the permissions of a file is by using the `chmod` command:

```
$ chmod MODE PATH
```

This command forces us to pass, first, the mode we want to set for the file, and second, its path.

The mode can be defined through characters or through its octal version, which is just a numerical representation of the 3 groups of permissions that we can configure for the file.

In the following image you have an explanation of how the permissions are translated between textual and octal modes:

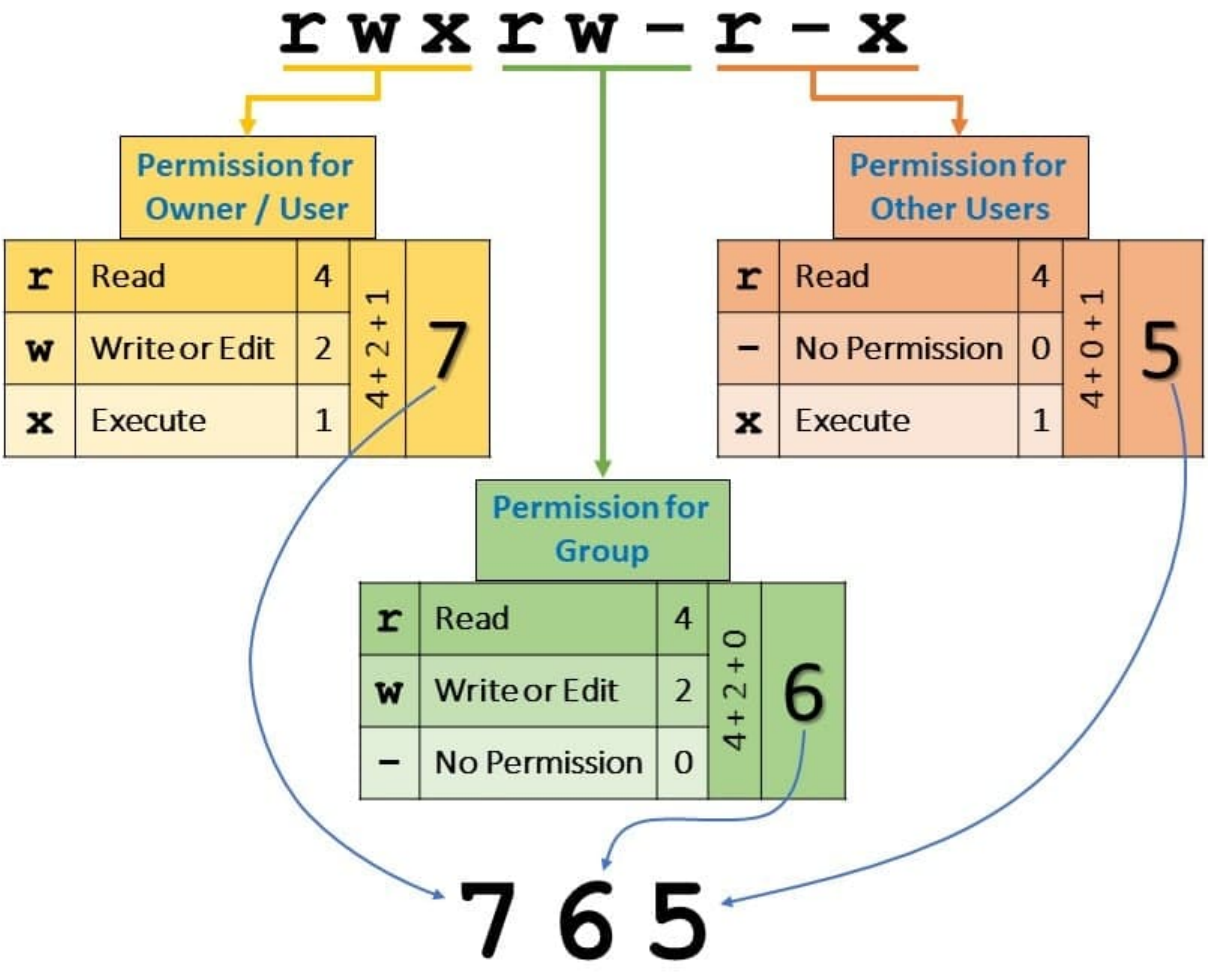

Fig. 2.1: Permission conversion.

2.1.3 Exercise

To consolidate what we have seen during this session, we are to carry out a series of practical exercises using a terminal, the command line and some of the commands seen previously:

1. Check if in the `/` directory of your machine there are any additional files or directories to the ones mentioned in this session.

   #### Solution

   The following items are shown:

   - `bin`
   - `initrd.img` e `initrd.img.old`
   - `lib`, `lib32` y `lib64`
   - `lost+found`
   - `sbin`
   - `vmlinuz` y `vmlinux.old`

   Most are symbolic links that are kept for compatibility with older versions. `initrd.img` and `vmlinuz` are links to the boot and kernel images. `lost+found` is the directory where corrupted files are recovered when doing checks on any 'ext'-type file system.

2. Look up in the help or the manual for the `ls` command how to show the details of each file and check the size of the `/etc/fstab` file.

   #### Solution

   Use `man ls` or `ls --help` to find that the appropiate solution is `ls -l` and run `ls -l /etc/fstab`

   ```
   $ ls -l /etc/fstab
   -rw-r--r-- 1 root root 826 feb  8 11:45 /etc/fstab
   ```

3. Create a `licenses` directory inside your user directory.

   #### Solution

   ```
   mkdir licenses
   ```

4. Look up in the help or the manual for the `cp` command how to make a recursive copy of a directory and copy the `/usr/share/common-licenses` directory to the one created in the previous step. Is the obtained result the expected one?

Solution

Look for in the `cp` manual or in `cp --help` the recursive mode and use it with `cp -r`:

```
$ cp -r /usr/share/common-licenses licenses
```

Doing so will create a `licenses/common-licenses` directory. Thin, about why this happens, what was the expected result and how to achieve it.

5. Using `ls` and the needed options, compare the permissions and ownership of the directories `/usr/share/common-licenses` and `licenses/common-licenses` in your personal user directory. Do you see any difference? Why are those differences happening?

Solution

Run `ls -l` on the 2 indicated directories and check that the main difference is the owner of the files and the dates.

6. Change the working directory to `licenses` and, from there, create a `GPLs` directory in your personal user directory.

Solution

`cd licenses` y `mkdir ~/GPLs` o `mkdir ../GPLs`

7. Copy the 4 files starting with `GPL` from the `licenses` directory to the `GPLs` directory.

Solution

```
cp common-licenses/GPL* ../GPLs/
```

8. Change the working directory again to `GPLs` and modify the files permissions to make them only readable and writable to your user (remove the permissions to the rest of the users of the system).

Solution

`cd ~/GPLs`, and `chmod 600 *` or `chmod go-rwx *`

9. Delete the `licenses` directory with all its subdirectories and files.

Solution

`cd` and `rm -fr licenses`. Try it without doing `cd` first, so that we can see that it is removed and that `pwd` still shows it as a working directory, but we can't do anything inside.

10. Delete the `GPLs` directory with all its subdirectories and files to keep your user directory clean.

Solución

`cd` and `rm -fr GPLs`

> **Note**
>
> Use the `clear` command to clear the content of a terminal. Also pressing `Ctrl l`.

2.2 Lab 2 - Command Line II

We continue with the work we have done in *lab 1*, deepening a little more in the handling of the command line. This time we will see how to group commands to create scripts and then, to automate tasks. Also how to use some very powerful techniques to optimize functionality between commands such as redirections and pipes. Finally, we will see how to install new programs using the Debian package manager.

2.2.1 Scripts

One of the most important features of the command line is that everything you do on it can be stored in a file and executed as many times as you want. The command line itself accepts as input an entire programming language (with *if* statements, *for* loops, etc.), which makes task automation possible at all levels. These programs that contain a set of commands to be executed at once are known as *scripts*.

Let's see how to create very simple *scripts* from the terminal.

Editing files with `nano`

To create a new file, just pass the path in you want to store it. For example:

```
$ nano my-file.txt
```

You can type some text and, when you want, save it to disk using`Ctrl o`. It will ask for confirmation of the file name, which will be the same as the one you have given, and you accept with `Enter`. The file will then be saved.

To exit the editor, just use `Ctrl x`. If you did not save, it will ask you if you want to exit without saving or to save the current changes.

> **Note**
>
> With `Ctrl g` you can see all the keyboard shortcuts that `nano` allows. In particular there are interesting `Ctrl k` and `Ctrl u`, which allow you to cut and paste a line, respectively.

Editing File with `gedit`

There are also graphical plain text editors such as `gedit`. You can launch it from the terminal by simply using the command `gedit`.

> **Warning**
>
> Although graphical editors are easier to use, we recommend that you get used to work with editors such as `nano`, because these editors are present in many network hardware such as routers, from where you will not have a graphical interface. In addition, it is usually much faster to edit small changes with such tools than with the graphical alternatives.

Exercise: creating a script

Now that you know how to create a text file, you can create scripts and run them. For example, create the file `/home/alumno/my-script.sh` with the following content:

```
#!/bin/bash

echo "-- Deployer v1.0 --"

# First, create all directories
mkdir dir1 dir2 dir3

# Then, copy myself into these directories
cp /home/alumno/my-script.sh dir1
cp /home/alumno/my-script.sh dir2
cp /home/alumno/my-script.sh dir3

echo "-- All done! --"
```

The script is quite simple to understand. There is some new stuff:

- The `#!/bin/bash` line that is always put at the beginning is known as *shebang*. That first line needs to start with `#!` followed by the interpreter you want to use. In this example, we use Bash.
- The command `echo` is used to print strings via the standard output.
- Lines beginning with `#` are considered comments, so the interpreter ignores them.

To execute the new script you can do:

```
$ bash /home/alumno/my-script.sh
```

Also is very common to give it execution permissions:

```
$ chmod +x /home/alumno/my-script.sh
```

And, afterwards, you can only launch it by giving the path to the script:

```
$ /home/alumno/my-script.sh
```

Or if you are in `/home/alumno`:

```
$ ./my-script.sh
```

> **Note**
>
> As an exercise, create a script that removes everything the previous script generates.

2.2.2 Searching with `grep` and `find`

One of the most recurrent tasks is to search within files and directories. `grep` is a very powerful command to look for patterns in file contents. On the othe hand, `find` is a command to look for files and directories whose name satisfies a given pattern.

The main structure of `grep` is:

```
$ grep [OPTIONS] PATTERN [FILE FILE ...]
```

The `PATTERN` argument is the string you want to look for. This string can simply be a word or even a *regular expression*, which allows us to define generic patterns. Let's see some simple examples:

```
$ grep auto /etc/network/interfaces
$ grep -i AuTo /etc/network/interfaces
$ grep -i AuTo --color /etc/network/interfaces
$ grep -i AuTo --color -H /etc/network/interfaces
$ grep -i AuTo --color -RH /etc/network
```

In turn, the main structure of `find` is:

```
$ find [STARTING-POINT] [OPTIONS]
```

Where `STARTING-POINT` is the path from where it starts searching for files. In `OPTIONS` you have all kinds of filters, so you can refine your search (by name, by size, by type, etc.). Here have some examples:

```
$ find
$ find /usr/bin
$ find /usr/bin -type f
$ find /usr/bin -type f -perm 755
$ find /usr/bin -type f -perm 755 -size +100k
$ find /usr/bin -type f -perm 755 -size +100k -mtime +100
```

2.2.3 Redirections

As we have said, commands are programs. And programs, when run, create execution *processes*. In a GNU/Linux system, a process has 3 points through which it can communicate with the outside. By default, every process has assigned these *file descriptors*:

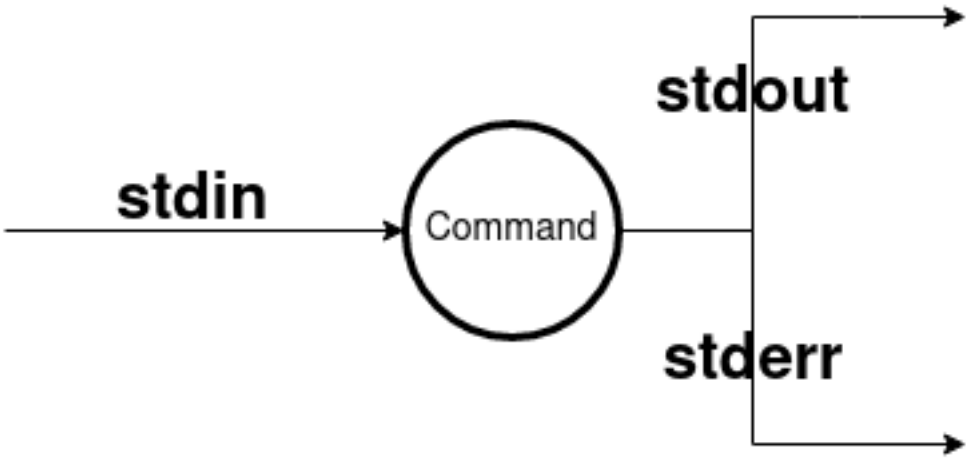

Fig. 2.2: Default file descriptors of a command.

- `stdin` or standard input, through which the command can receive information. By default, the standard input is the keyboard.

- `stdout` or standard output, where the command produces its output. By default, the standard output is the command line itself.
- `stderr` or standard error output, whereby the command can issue error or debugging messages. By default, it is also the command line itself.

> **Note**
>
> Standard output and standard error are separated, primarily to distinguish which parts should be taken as purely command output from error messages. This is very important when using redirection or *pipes*.

In GNU/Linux, it is possible to change the locations pointed by these file descriptors of the commands. To do this, the following operators are used:

- `cmd > file.txt`: redirects the `cmd` command output in the file `file.txt`.
  - If the file does not exist, it is cretated
  - If the file exists, it is completeley rewritten.

```
$ cat /etc/fstab > my-fstab.txt
$ echo "Hello!" > greetings
$ echo "World!" > greetings
```

> **Note**
>
> There is a special file where everything that you do not want to store goes: `/dev/null`. It is very common to redirect command output to this file to discard it.

- `cmd >> file.txt`: redirects the `cmd` command output in the file `file.txt`.
  - If the file does not exist, it is cretated
  - If the file exists, the output is appended to the end of the file.

```
$ cat /etc/fstab > my-fstab.txt
$ echo "# this goes at the end!" >> my-fstab.txt
```

- `cmd < file.txt`: redirects the `cmd` command standard input so that `file.txt` is the input and not the keyboard.

```
$ cat < /etc/fstab
```

Exercises

1. Stores in the `libs.txt` file the files (recursively) ending with `.so` from the `/usr/lib` directory.

Solution

```
$ find /usr/lib -type f -name *.so > libs.txt
$ cat libs.txt
```

2. Add to the end of the same file those ending in `.a`.

Solution

```
$ find /usr/lib -type f -name *.a >> libs.txt
$ cat libs.txt
```

> **Note**
>
> For simplicity, there is no mention of how to manipulate the error output, but it is also possible to redirect it like the other two.

2.2.4 Pipes

Pipelines are probably the most attractive feature of the command line environment in GNU/Linux. It provides the ability to perform non-trivial tasks in a easy, simple and clear way.

The concept is based on what has been seen in *redirections*, where all commands had an input and an output flows that could be manipulated. The pipe operator (denoted with the vertical bar `|`) connects the output of one command to the input of another. In this way, commands that perform different tasks can be joined together to perform, overall, a more complex task.

> **Note**
>
> The UNIX philosophy is behind the pipes: you provide many programs that do specific things and then compose them together to create advanced functionality.

For example:

```
$ find /usr/lib -name *.so -type f | less
$ cat /etc/passwd | grep 100 --color
$ grep -Hiw network /usr/share/common-licenses/* | cut -f 1 -d ":" | sort -u
```

Exercises

- Download the file Don Quixote in plain text and count its number of words. You can use the `wc` command for it.
- ¿Would you know how to do the whole proceses, including downloadking the file, with several chained commands?

Solution

```
$ $ wget -q https://uclm-esi.github.io/redes1-lab/assets/quijote.txt -O - | wc -w
```

2.2.5 Root Permissions

As you may have noticed, there are many items in the file system that you cannot access due to lack of permissions. For example:

```
$ cat /var/log/messages
cat: /var/log/messages: Permission denied
```

As we saw in the *permissions* section, you need to have the appropriate permissions to access a file in order to perform the write, read or execute operation. However, there are also other types of actions that, for security reasons, are reserved for users with management permissions. Some of these operations are:

- Add users or groups.
- Change passwords from other users.
- Install new programas on the system.
- Low-level access to network interfaces.

In our virtual machine, the `alumno` user is configured so that it can execute commands as a root user when needed. To do this, use the `sudo` command follow by the command you want to execute. For example:

```
$ sudo cat /var/log/messages
```

2.2.6 Packages Systems

If there is one aspect that characterizes a GNU/Linux distribution is the package system it uses. A distribution is based on collecting software that is available as free or open source code, grouping it together and configuring it so that users can access it quickly and easily. The package system is the component with which users can manage the programs they install.

In Debian, the different package systems are based on a very specific format: the *Debian packages*. These packages, packaged in `.deb` files, usually contain software already compiled and ready to use, plus extra information such as dependencies on other programs, program author, etc.

dpkg

Debian packages can be installed directly with the `dpkg` command. However, you cannot install packages directly as a normal user, you must have root permissions:

```
$ sudo dpkg -i <path/to/file.deb>
```

> **Warning**
>
> If the package has dependencies `dpkg` *no* will resolve them and will not try to install them. To do that, you must use `apt` as we will see later.

You can also remove a package such as:

```
$ sudo dpkg -r <package-name>
```

> **Note**
>
> The package name is *different* from the `.deb` file name. For example: the package `myapp-1.0.deb` will most likely have the package name as `myapp`.

And some interesting options to list the system packages and the files contained in a package are:

```
$ dpkg -l
$ dpkg -L <package-name>
```

apt

At a higher level of package management is `apt`. This tool, which uses `dpkg` underneath, provides greater functionality since:

- Resolves dependencies between packages, installing whatever is necessary.
- Access to remote repositories. `apt` can connect to an application store to download and install them automatically.
- Look for packages based on different criteria (content, description, type, etc.).

The most common ways to use `apt` are:

```
$ sudo apt update
$ sudo apt search <pattern>
$ sudo apt show <package-name>
$ sudo apt install <package-name>
$ sudo apt remove <package-name>
$ sudo apt purge <package-name>
$ sudo apt upgrade
$ sudo apt full-upgrade
```

Exercises

- Download the package `redes1` and install it using `dpkg`. What files does it have? Can you use any of them? Finally, remove it using `dpkg`.

  #### Solution

  ```
  $ dpkg -L redes1
  ```

  You can run `redes1` and get a message.

- Install the `sl` and `tree` packages using `apt`. What do they do according to their description? Uninstall the packages completely.

Solution

```
$ apt update
$ apt install tree sl
$ apt show tree
$ apt show sl
$ apt purge tree sl
```

2.3 Lab 3 - Client and Server

In this session, we will delve into the concepts of client and server from a practical point of view and, at the same time, learn the utility called *netcat* (`nc` command): a basic tool used to diagnose problems in networks, among other uses.

The client-server communication model is graphically described in the following figure:

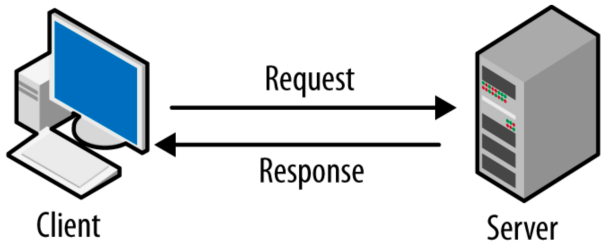

Fig. 2.3: Basic communication scheme between a client and a server.

The client makes requests to the server and the servers responds to these requests. Some considerations to take into account:

- The server has a *reactive* behavior, i.e., it waits for incoming requests to act on the communication.
- The client has a *proactive* behavior, i.e., it makes requests to the server when needed and *initiates* the communication.
- A node can have both roles at the same time. It is possible that, within a more complex environment, the same node can be the client of a server and the server of other different nodes. The separation of these two roles is *logical*, not phisical.

2.3.1 *netcat*

To create simple connections between nodes using the client-server model, one of the simplest tools is *netcad*. In the terminal, `nc` command is used. If you do not have it installed, you must install the `netcat` package:

```
$ sudo apt install netcat
```

2.3.2 Case Studies

The following case studies are intended to be performed in the lab in pairs (each member of the pair with his or her lab PC).

> **Note**
>
> It would also be possible to practice at home using 2 virtual machines launched at the same time. Each virtual machine will have its own IP address, and therefore one can act as a server and the other as a client. You only need to have 2 imported virtual machines.

One member of the pair will act as the client and the other will act as the server. It is recommended to exchange the roles in the different exercices so that both members play the role of client and server.

In order to stablish the communication between them, it is necessary to know the both IP addresses. We can know them using the `ip` command:

```
$ ip addr
```

Also it is possible to obtain IP addresses using the `ifconfig` command, although the first method is preferable:

```
$ sudo ifconfig
```

These commands show us the IP addresses available on our machine. It is in the form of 4 bytes separated by dots. For example: `192.168.1.120`. The IP addresses are associated with the *network interfaces* available in the node. As our virtual machine has only one network interface (simulated) we will only see one valid IP address.

> **Note**
>
> Both commands show a special network interface called `lo` whose IP is always `127.0.0.1`. This interface belongs to the local network of the machine itself, and should be ignored for the time being.

Once you have taken note of which IP address dcorresponds to the client and which to the server, you can start with the following practical exercises of communication between client and server.

Client-Server Chat

Server

Type in the server the command:

```
$ nc -l -p 5400
```

The server listens with the `-l` (*listen*) option on port `5400`. By default TCP is used, although it could be done with UDP by adding the `-u` option.

Client

Once the server is listening, type in the client:

```
$ nc <server-IP> 5400
```

Replace `<server-IP>` with the IP of the server you wish to connect to. This will cause `nc` to open a TCP connection to the server.

Once connected, the client communicates with the server as if they shared keyboard and screen: any text entered by either will be seen by both whenever it is confirmed with `Enter`.

To abort the communication just press `Ctrl c` on either one. This aborts the program and closes any communication in progress.

Sending and Saving Text

Server

Type in the server the command:

```
$ nc -l -p 5400 > output.txt
```

As before, the server listens on port `5400`. Now whatever it receives will be redirected to the `output.txt` file. As soon as the client connects and starts sending data, the file will acquire content.

Once the connection is closed, you can view the contents of the file with `cat`. Do not forget to remove the file with `rm`.

Client

Once the server is listening, type in the client:

```
$ nc <server-IP> 5400
```

Write a test sentence such as: `You'll be free, hackers, you'll be free.`

The `Enter` key does not close the connection, it simply enters a line break. When you have finished typing the text, you can close the connection with `Ctrl c`.

Viewing Received Content in the Server

Server

Type in the server the command:

```
$ nc -l -p 5400
```

The server linstens on port `5400`. Now what it received will be displayed on the screen.

Client

Create a file named `file.txt` with any content, for example: `At our call, hackers, at our call.`. You can use `nano` to create it.

```
$ nc <server-IP> 5400 < file.txt
```

Do not forget to remove the file with `rm` at the end.

Sending a File from the Client (Upload)

Server

Type in the server the command:

```
$ nc -l -p 5400 > book.pdf
```

The server linstens on port `5400` and what it receives will be saved in the file `book.pdf`.

Once finished, do not forget to remove the file with `rm`.

Client

Download the course workbook with `wget`. Replace `en` with `es` if you want the Spanish version:

```
$ wget https://uclm-esi.github.io/redes1-lab/en/redes1-lab-en.pdf
```

Now you can send it to the server using:

```
$ nc <server-IP> 5400 < redes1-lab-en.pdf
```

Once finished, do not forget to remove the PDF with `rm`.

> **Note**
>
> You can view the content of the PDF by opening the file with the `evince` command. It will open a graphical program to view the PDF passed as an argument.
>
> You must close the PDF in order to regain the control of the terminal, which is locked in the meantime. You can also use `Ctrl c` to end the program from the terminal.

The file name under which the PDF is saved on each side of the communication may be the same or different. Each communication actor assigns a name to it.

Saving Data from the Server

Server

Type in the server the command:

```
$ nc -l -p 5400
```

The server listens on port `5400`, but can also receive keyboard input. This input will be sent to the client when it connects.

Let's try this. Type a sentence such as: `We'll kick out those dirty licenses`.

Once the client connects, it will receive the text you have introduced. If you continue introducing text, the client will continue receiving it.

Client

Wait until the server has entered all text. When ready, run:

```
$ nc <server-IP> 5400 > file.txt
```

The file `file.txt` will be updated with the data sent by the server. You can open another terminal and view the contents of the file with `cat`.

Do not forget to remove the generated file with `rm`.

Viewing a Server File in the Client

Server

Create a file named `file.txt` with any content, for example: `Join us now and share the software`. You can use `nano` to create it.

```
$ nc -l -p 5400 < file.txt
```

The server listens on port `5400` and as soon as the client connects it will receive the content of the file.

Client

When the server will be ready, run:

```
$ nc <server-IP> 5400
```

The content of the file that the server has made available will be displayed on the screen.

Sending a File from the Server (Download)

Server

Download the course workbook with `wget`. Replace `en` with `es` if you want the Spanish version:

```
$ wget https://uclm-esi.github.io/redes1-lab/en/redes1-lab-en.pdf
```

This will create the file `networks1-lab-en.pdf` in the directory from where `wget` was launched. Now start the server with:

```
$ nc -l -p 5400 < redes1-lab-en.pdf
```

The server listens on port `5400` and as soon as the client connects it will receive the PDF.

Do not forget to remove the PDF file with `rm` at the end.

Client

When the server will be ready, run:

```
$ nc <server-IP> 5400 > book.pdf
```

When connected, the server will send the PDF and the client will save it in the file called *book.pdf* .

2.4 Lab 4 - Capture of network traffic

A computer sends or receives network traffic through its network interfaces. Depending on the network configuration and your own equipment, you can capture network traffic in order to study it, analyse it or even save it to a file for later use.

We need a specific software to carry out the traffic capture. In this practice we are going to learn how to use the `tshark` program that will allow us to do the mentioned actions.

It is also possible to use `wireshark`, a graphical environment version of `tshark`. Everything explained for the latter is also valid for the former.

2.4.1 Preparing the environment

Network configuration

First of all, you have to make sure you have a network connection. For this, open the terminal and run the following:

```
$ ip addr
```

The output should show all of your computer's network interfaces with their MAC addresses and their IP addresses if they have them.

Depending on the network you are on, the IP will have a different value. Remember that it must always be in the form of 4 bytes (values from 0 to 255) separated by dots, for example `192.168.1.120`.

In principle, if you use the *virtual machine*, both on the lab computers and on your PC, you should have no problem when capturing traffic.

Capture software installation

As mentioned above, we will use `tshark` and `wireshark` programs to capture traffic.

To check if you have these programs installed, use:

```
$ command -v tshark
$ command -v wireshark
```

If the path to any of the programs is not shown when execute the above commands, it means that it is not installed. So install it using:

```
$ sudo apt install tshark wireshark
```

2.4.2 Get started with `tshark`

`tshark` is a terminal tool that allows us to capture network traffic and save it to a file or display it on the standard output. To capture traffic, first find out on which network interface or interfaces of the computer we want to do it.

To do this, we will use the `ip addr` command. With it, we can view the IP addresses of the network interfaces, being able to find out on which of them the connection is configured.

> **Note**
>
> If you use the virtual machine, you should view 2 interfaces: one called `lo` or *loopback* interface, which gives IP connectivity to the machine itself, and a second interface whose name will depend on the environment, which will

be the one that gives connectivity to the outside of the machine. You must pay attention to the second one to make traffic captures.

Once you know the names of the network interface(s) you want to use, you can use the `tshark -D` command to view the correspondence between the interface name and the index to use for captures:

```
$ tshark -D
Running as user "root" and group "root". This could be dangerous.
1. enp0s3
2. any
3. lo (Loopback)
4. ciscodump (Cisco remote capture)
5. dpauxmon (DisplayPort AUX channel monitor capture)
6. sdjournal (systemd Journal Export)
7. sshdump (SSH remote capture)
8. udpdump (UDP Listener remote capture)
```

For the lab nodes, we will normally use the network interface of our virtual machine, so select the index corresponding to the interface listed with `ip addr` (**1** in this case).

The first capture

To create the first capture, open your terminal and run:

```
$ tshark -i 1
```

This command will start capturing traffic on interface number `1` (the network interface of the virtual machine as we saw before). The default behaviour is to display a brief summary of the captured packets in the program output. Press `Ctrl c` to stop capturing.

```
Capturing on 'enp0s3'
 ** (tshark:594982) 00:11:55.871189 [Main MESSAGE] -- Capture started.
 ** (tshark:594982) 00:11:55.871276 [Main MESSAGE] -- File: "/var/tmp/wireshark_
 ↪wlp0s20f35RkW4P.pcapng"
    1 0.000000000 192.168.1.39 → 255.255.255.255 UDP 909 54712 → 29810 Len=863
    2 3.684236270 fe80::9a97:d1ff:fe77:9368 → ff02::1      ICMPv6 82 Router␣
 ↪Advertisement from 98:97:d1:77:93:68
    3 4.094190431 192.168.1.91 → 255.255.255.255 UDP 214 49153 → 6667 Len=172
    4 4.094193127 192.168.1.91 → 255.255.255.255 UDP 218 49153 → 6667 Len=172
    5 2.247868733 MitraSta_77:93:68 →              ARP 62 Who has 192.168.1.95? Tell␣
 ↪192.168.1.1
    6 2.247899195 IntelCor_d1:62:2b →              ARP 44 192.168.1.95 is at␣
 ↪c8:09:a8:d1:62:2b
    7 5.119982488 192.168.1.39 → 255.255.255.255 UDP 905 54712 → 29810 Len=863
    8 5.119985457 192.168.1.39 → 255.255.255.255 UDP 909 54712 → 29810 Len=863
    9 9.009472416 192.168.1.91 → 255.255.255.255 UDP 214 49153 → 6667 Len=172
   10 9.009534174 192.168.1.91 → 255.255.255.255 UDP 218 49153 → 6667 Len=172
```

After a few seconds you will see an output similar to the previous one. Apart from a first message confirming that the capture is being performed on the `enp0s3` interface and a couple of log messages from the `tshark` itself, a summary of each captured packet per line is shown. Their structure is as follows:

1. Packet number (ordinal) in the capture.
2. Time elapsed relative to the first packet captured..
3. Source and destination IP addresses of the packet (or MAC address if lower layer traffic).

4. Protocol (TCP, UDP, ICMP, DNS…).
5. IP or Ethernet datagram size.
6. Source and destination ports (only in the case of TCP and UDP).
7. Short description of the packet.

Saving a capture in a file

When we are making a live capture, it might not be possible to inspect the packets in detail, if there is a too much traffic. If you need this, it is highly recommended that `tshark` saves the traffic capture in a file on disk:

```
$ tshark -i 1 -w /home/alumno/capture1.pcapng
```

In this way, the program will not show any information in the output, but it will save all the capture in the path that you have passed as argument. This allows us to inspect the traffic capture as many times as we need, open it with other programs, such as the aforementioned `wireshark` and even "inject" the capture again, simulating a traffic already passed.

To show the content of the saved file with `tshark`, use the following option:

```
$ tshark -r /home/alumno/capture1.pcapng
```

2.4.3 Filtering `tshark` captures

`tshark` allows you to define 2 types of filters:

- *Capture filter*: only the packets that meet the defined filter will be shown and/or saved in the capture.
- *Display filter*: only shows the packets that meet the condition defined by the filter, but the rest will still be saved in the capture.

Capture filters

This type of filters allows us to lighten the size of the resulting capture. It is suitable when we have a precise idea of which packets we want to be able to analyse in the capture.

To make a capture filtering we must use the following option:

```
$ tshark -f <capture filter>
```

The syntax of the capture filters is the so-called BPF. Allowing us to make filters such as the following:

Filter	Obtained Result
icmp	ICMP traffic
host 192.168.1.1	Traffic to or from the IP 192.168.1.1
tcp portrange 1-1024	TCP packets received or sent at any port in range 1 to 1024
tcp and not http	TCP packets, but do not contain HTTP traffic

Although capture filters are very useful, especially for lightening the size of the captures, they are actually less versatile than the display filters as we will see in the next section.

Exercise: filter the packet capture

1. In a terminal, start a traffic capture that only captures ICMP packets. Save the capture in a file named `icmp_cap_filter.pcapng`.
2. Open a second terminal and make a `ping` with 10 requests to `8.8.8.8`. Use the `-c 10` option to stop the `ping` command after sending the 10 requests.
3. When `ping` has finished, stop the capture with `Ctrl c`.

Display filters

The display filters allow `tshark` to capture all traffic (or all traffic defined in the *capture filters*) and display only those packets that we want. This is much more flexible, since with the same capture we can easily inspect different types of traffic using different display filters.

Furthermore, unlike capture filters that can only be based on the actual fields in the packets, display filters can take into account context information that `tshark` can extract from the analysis of the complete trace.

For example, with either of the 2 types of filter we could find ICMP packets, but with the display type, we could request to see only ICMP packets that have not received a response.

Although the syntax of the display filters is very similar to that of capture filters, viewed above, they do not follow exactly the same rules.

To define a display filter:

```
$ tshark -Y <display filter>
```

For example, to only view the ICMP packets sent from the IP `192.168.1.11`, use the following filter:

```
$ tshark -Y 'ip.src == 192.168.1.11 and icmp'
```

> **Note**
>
> It is recommended to use single quotes to delimit the filter using terminal commands to prevent the shell from splitting the filter or expanding variables or terminal wildcards.

For layer 2 to 4 protocols, `tshark` allows filtering by sending, receiving or either direction:

Protocol	Source	Destination	Anyone
Ethernet	eth.src	eth.dst	eth.addr
IP	ip.src	ip.dst	ip.addr
TCP	tcp.srcport	tcp.dstport	tcp.port
UDP	udp.srcport	udp.dstport	udp.port

Any of these fields can be matched with the equality or inequality operators o (== o `!=`).

Exercise: display filters

1. In a new terminal, prepare ready to execute the command to download the file phone_new.gif, using for example `wget` but without running the command for now.

   ```
   $ wget http://www.esi.uclm.es/www/isanchez/phone_new.gif
   ```

2. Start a capture of all traffic with `tshark` without using any capture filter. Save this capture in a file named `http_get_gif.pcapng`.
3. Run the command you prepare in step 1.
4. Stop `tshark` to end the capture with `Ctrl c`.
5. Load the capture back into `tshark` and apply a display filter to show only DNS protocol packets.

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'dns'
   ```

6. What IP (DNS server) did your machine connect to in order to make the DNS request for the domain `www.esi.uclm.es`?
7. Load the capture back into `tshark` and apply a display filter to show only HTTP protocol packets.

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'http'
   ```

8. What is the IP of the server hosting the image?

> **Note**
>
> In the display filters, protocols must be written in lowercase letters.

2.4.4 Modifiying `tshark` output

Sometimes the output provided by the `tshark` command does not include a field that may be relevant to us. Of course, the command allows us to modify its output to show the relevant data we need.

To modify the fields shown by default, we will have to activate the `-T` option, indicating in which format we want to show the output (for now use `fields`), and with `-e` we will indicate the columns we want to show. We can use `-e` as many times as we need:

```
$ tshark -r cap1.pcapng -T fields -e ip.addr -e tcp -e udp -e _ws.col.Info
```

The above command would show us the source and destination IP addresses, relevant information about TCP or UDP, depending on the transport protocol used by each packet, and finally a special column containing the brief summary of the packet payload.

> **Note**
>
> To see a complete list of all fields that `tshark` allows us to use with `-e`, you can list them with `tshark -G fields`. The list is quite large, so use the commands you learned in the first practices to filter or paginate the output.

Exercise:

1. Use the capture made in the previous block of exercises named `http_get_gif.pcapng`.
2. Load the capture with `tshark` and, filtering only DNS packets, modify the output to show the information related to the transport protocol.

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'dns' -T fields -e udp
   ```

3. What were the source and destination ports of the DNS requests?
4. Reloads the previous capture, but this time filters only HTTP packets. Modify the output to show information related to the transport protocol.

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'http' -T fields -e tcp
   ```

5. Which TCP ports did the HTTP request use?
6. Run the following commands. Say what you observe, what do you think it is due to?

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'dns' -T fields -e tcp
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'http' -T fields -e udp
   ```

7. Find out the response code of the HTTP request that downloaded the *gif* file in the example:

   ```
   $ tshark -r /home/alumno/http_get_gif.pcapng -Y 'http' -T fields -e _ws.col.Info
   ```

2.4.5 Using `wireshark`

So far we have seen the use of `tshark`, which is a command line tool. Its use is recommended for captures with simple filters or to analyse packets at very basic levels, such as physical addresses, network addresses or ports, as well as to filter by protocols quickly.

Its main advantage is that it allows us to redirect the output, using pipelines to other commands to filter information, colour it, or save it in a readable format without having to learn other tools.

However, for deeper analysis, `tshark` may be too complex, so for those situations it is fine to use `wireshark` instead.

Starting `wireshark`

Like `tshark`, its graphical version also allows us to make captures and visualise them: we can open `pcapng` files generated with `tshark` without any problem and also generate files to be able to analyse them with `tshark` or `wireshark` later.

From the initial screen we can select either to start a capture, defining a capture filter using the same syntax as above, or to open a previously saved capture file from the `File` menu.

By clicking on the blue button we can start capturing traffic, and in the text input box under `Capture` we can define a capture filter.

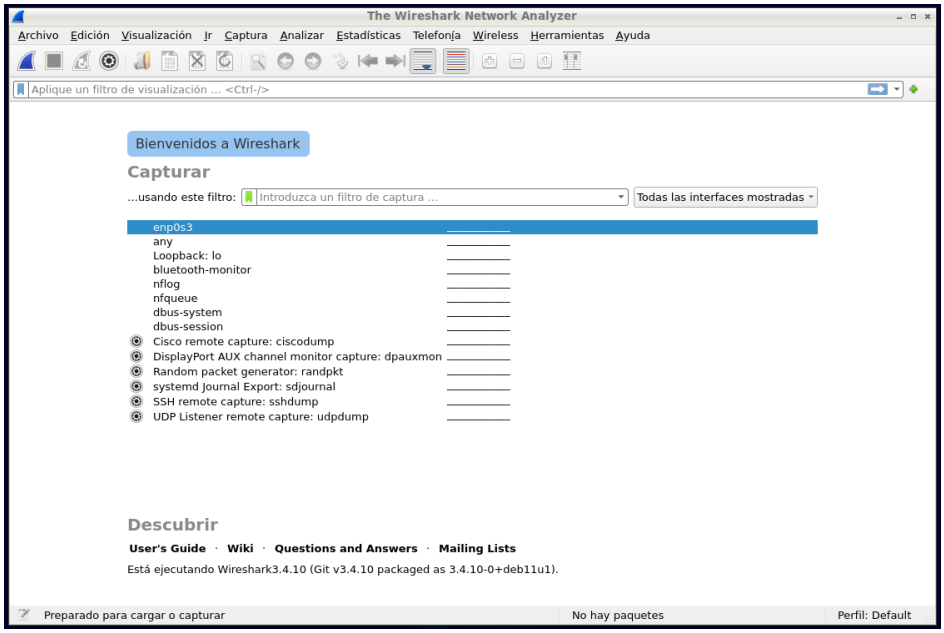

Fig. 2.4: Screenshot of Wireshark startup window.

Analysis display

While `wireshark` is performing a capture or we have loaded one from a file, you will see a window like the following:

This window is divided into 4 important parts, from top to bottom:

1. *Display filter* input: allows us to define a display filter, in the same way as with `tshark -Y`.
2. List of *packets*: the packets of the capture and a series of basic information about them are displayed, once the defined filter, if any, has been applied.
3. *Breakdown* of selected packet: when we have a packet selected, here we will be able to view internal information of the protocol of each layer.
4. Hexadecimal and ASCII *content* of the packets: it shows the information at the lowest possible level, being able to see at all times the correspondence between the hexadecimal value of each byte of the packet and its ASCII representation, if any.

Exercise: analysing a capture with `wireshark`

1. As in the previous exercise, open the `http_get_gif.pcapng` capture, but this time with `wireshark`.
2. Find the same values as requested in the previous exercise.
3. Find in the capture the HTTP request corresponding to the download of the GI file from the server and identify the HTTP headers that have been used such as the `User Agent`.
4. Repeat the same, but with the HTTP response.

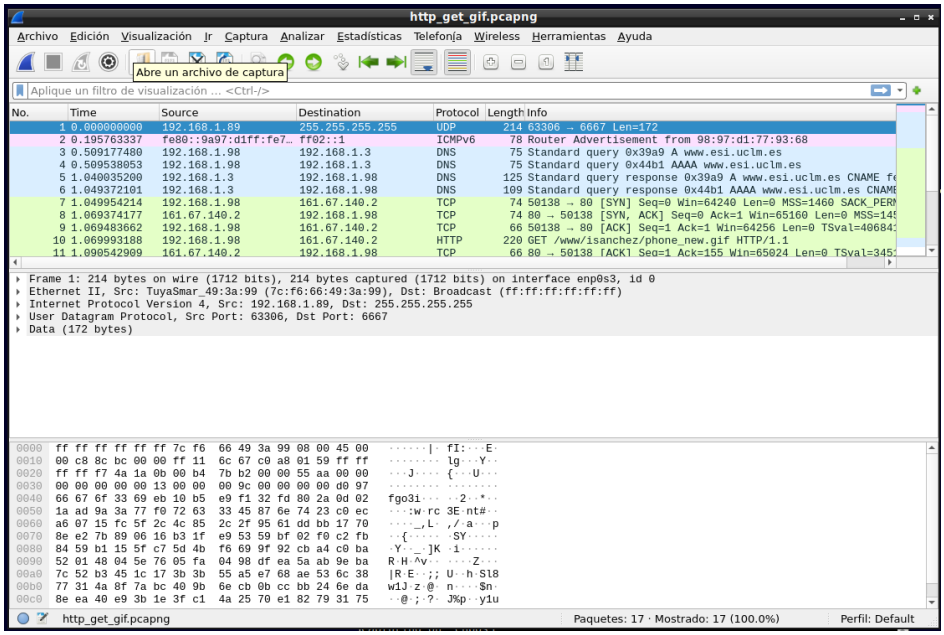

Fig. 2.5: Screenshot of the analysis display of packets in Wireshark

2.5 Lab 5 - Application layer analysis

The purpose of this lab is to capture network traffic related to the exchange of messages involving application-level protocols, their analysis and the identification of the different message fields. Specifically, this session will look at DNS and HTTP protocols.

2.5.1 Assessment

This session will be graded (**1/6** of the lab grade), so you must complete the corresponding questionnaire in Campus Virtual, answering the questions. The questions to be answered will be available during the practical session and must be submitted before it ends. Each wrong answer *will be penalized with 1/3* of the assigned value.

2.5.2 Setup

The network connection must be correctly configured to carried out the lab:

1. Aspects already seen in previous labs are assumed to be known, for example, the command to know the own IP and the usage of Wireshark.
2. Prepare your working directory:
   - Create the `p5` directory in `/home/alumno`. Assuming your current directory is `/home/alumno`, the command would be:

     ```
     $ mkdir p5
     ```

   - If it already exists, remove it and create it again:

     ```
     $ rm -rf p5
     $ mkdir p5
     ```

   - Go to this directory and do the tasks from there:

```
$ cd p5
```

- Check that you are in the correct directory with:

  ```
  $ pwd
  ```

  It should show:

  ```
  /home/alumno/p5
  ```

3. Be sure that the network interface of the virtual machine has an IP address of type `172.24.21x.xxx`.
4. Be sure that you can go out to the Internet using the virtual machine. For example, you can run:

   ```
   $ ping -c 3 8.8.8.8
   ```

   This should provide a result similar to:

   ```
   3 packets transmitted, 3 received, 0% packet loss, time XXXms
   ```

5. Open a browser *inside the virtual machine* and have these instructions and the quiz of Campus Virtual on hand. This will make it easier for you to follow them and download the needed files.

> **Important**
>
> **Do not save your Campus Virtual password in the vitual machine.**

2.5.3 DNS

dns1.pcapng

This capture was generated using the following command:

```
$ host www.uclm.es
```

This command resolves the name `www.uclm.es` to the corresponding IP address or addresses using the DNS protocol.

Download the capture `dns1.pcapng` and perform an analysis with `wireshark` or `tshark`. Answer the questions related to this capture.

dns2.pcapng

Download the capture `dns2.pcapng` and analyze it. Later, answer the related questions in the quiz. This capture was generated using the following command:

```
$ host redes1.com
```

2.5.4 HTTP

http1.pcapng

In this exercise the `curl` command will be used. This is a tool that allows you to make HTTP requests. By default, it uses the `GET` method which is the case of this example. Another method can be used through the `-X` option.

This time, you must create a capture named `http1.pcapng` that will contain HTTP traffic. To do this:

1. Run `wireshark`.
2. Open a terminal.
3. Start capturing traffic with `wireshark`.
4. Run the following in the terminal (you should view an output similar to the one shown):

   ```
   $ curl http://httpbin.org/get
   ```

   The output should be similar to:

   ```
   {
     "args": {},
     "headers": {
       "Accept": "*/*",
       "Host": "httpbin.org",
       "User-Agent": "curl/7.74.0",
       "X-Amzn-Trace-Id": "Root=1-621ff61f-71e17fc20c02922e7e26a8df"
     },
     "origin": "182.200.203.150",
     "url": "http://httpbin.org/get"
   }
   ```

5. Stop to capture traffic with `wireshark`.
6. Save the capture as `http1.pcapng`. Be sure that you save it in `/home/alumno`.
7. Use `http` as display filter.
8. Only 2 HTTP packets shoul be shown. If you do not get this output, you must repeat the capture.

Now, with the 2 shown HTTP packets, you can complete the part of the quiz assigned to this exercise.

http2.pcapng

To generate the network traffic in this exercise, the `wget` command will be used, which is used, among many other functionalities, to download files using HTTP. Unlike `curl`, this command saves server responses to files, as well as implementing higher level functionality such as following HTTP redirects automatically. In fact, this is exactly what it will do in this example.

In short, `wget` is a higher level tool than `curl` and allows you to download files over HTTP as a conventional browser would do, but without the need of a graphical environment.

Using the above procedure, you must create the capture named `http2.pcapng` as follows:

1. Run `wireshark`.
2. Open a terminal.
3. Start to capture traffic with `wireshark`.
4. Run the following in the terminal:

```
$ wget http://www.uclm.es
```

You should view an output similar to:

```
--2022-03-07 09:31:00--  http://www.uclm.es/
Resolving www.uclm.es (www.uclm.es)... 51.105.185.204
Connecting to www.uclm.es (www.uclm.es)|51.105.185.204|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://www.uclm.es/ [following]
--2022-03-07 09:31:00--  https://www.uclm.es/
Connecting to www.uclm.es (www.uclm.es)|51.105.185.204|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 113381 (111K) [text/html]
Saving to: 'index.html'
index.html              100%[=============================>] 110.72K  --.-KB/s  ␣
 ↪ in 0.03s
2022-03-07 09:31:00 (3.57 MB/s) - 'index.html' saved [113381/113381]
```

5. Stop to capture traffic with `wireshark`.
6. Save the capture as `http2.pcapng`. Be sure that you save it in `/home/alumno`.
7. Use `ip.addr == 51.105.185.204` as display filter.
8. You should view different types of traffic. Also `wget` must have downloaded a file named `index.html`. If any of this is missing, you should repeat the capture.

Now, with all the requirements fulfilled, you can analyze the traffic and answer the part of the quiz assigned to this exercise.

2.5.5 Cleanup

Finally, remove the created directory `/home/alumno/p5` with:

```
$ rm -rf /home/alumno/p5
```

2.6 Lab 6 - Transport layer I

The aim of this lab is to study the differences between UDP and TCP transport protocols, both in terms of performance and reliability. For this purpose, the `nc` tool will be used to transfer some files, as well as the `Wireshark` tool to make the corresponding captures.

More specifically, during this session you are going to send several files between two processes on your computer usiing the *loopback* (`lo`) interface. Text and sound files will be handled in this lab, using TCP and UDP transport protocols.

2.6.1 Assessment

This session will be graded (**1/6** of the lab grade), so you must complete the corresponding questionnaire in Campus Virtual, answering the questions. The questions will be available during the session and must be submitted before it ends. Each wrong answer *will be penalized with 1/3* of the assigned value.

2.6.2 Setup

The network connection must be correctly configured to carried out the lab:

1. Aspects already seen in previous labs are assumed to be known, for example, the command to know the own IP and the usage of Wireshark or Tshark.
2. Prepare your working directory:
   - Create the `p6` directory in `/home/alumno`. Assuming your current directory is `/home/alumno`, the command would be:

     ```
     $ mkdir p6
     ```

   - If it already exists, remove it and create it again:

     ```
     $ rm -rf p6
     $ mkdir p6
     ```

   - Go to this directory and do the tasks from there:

     ```
     $ cd p6
     ```

   - Check that you are in the correct directory with:

     ```
     $ pwd
     ```

     It should show:

     ```
     /home/alumno/p6
     ```

3. Create a directory to run the clients from there and another one for the servers:

   ```
   $ mkdir client server
   ```

4. Have two different terminals ready: go to the client directory in one of them and to the server directory in the other:

   ```
   cd ~/p6/client
   pwd
   ```

   ```
   $ cd ~/p6/server
   $ pwd
   ```

2.6.3 UDP

A text file will be transmitted between 2 programs on the same computer, using the local network interface or *loopback*, to analyze the operation of the UDP protocol. You will run the client and the server simultaneously, and you will capture the UDP packets with the tool of your choice: Wireshark or Tshark.

udp-uuid.pcapng

1. First, prepare the file to be sent from the client to the server. This file must contain a "Universally Unique IDentifier" (UUID). This type of identifier is composed of 32 hexadecimal characters grouped in 5 groups of different sizes, using the hyphen as a separator. To generate it, run the following on the client terminal:

   ```
   $ cat /proc/sys/kernel/random/uuid > uuid.txt
   ```

   The file must have a size of 37 bytes (36 bytes of the UUID and the newline character). You can check it with:

   ```
   $ ls -l uuid.txt
   -rw-rw-r--. 1 alumno alumno 37 Mar 20 16:58 uuid.txt
   ```

2. Run the capture with `Wireshark`. If you wish, save it as `udp-uuid.pcapng` for its later analysis.

   > **Note**
   >
   > You can specify to listen only in the "lo" interface (*loopback*) and use a "udp" capture filter, so that only the traffic that travels using UDP as transport protocol is captured.

3. In the server terminal, run the following command:

   ```
   $ nc -l -u -p 8888 > received-uuid.txt
   ```

   Te data received by `nc` will be redirected to the `received-uuid.txt` file.

4. In the client terminal, run the following command:

   ```
   $ nc -q 2 -u 127.0.0.1 8888 < uuid.txt
   ```

   In this exercise we use `127.0.0.1`, which is the IP address assigned to the local interface or *loopback*. That way we will communicate with another process inside our machine through the network protocol stack.

5. End the server execution by pressing `Ctrl c` in the corresponding window.
6. Stop the capture. Check that the transmission appears on it.
7. Answer the questions related to this capture.

udp-quijote.pcapng

1. In this exercise we are going to use the following capture: download the capture udp-quijote.pcapng. In it, the client sends the text file `quijote.txt` via the local interface or *loopback*, using the UDP transport protocol.
2. Open the capture with `wireshark`.
3. Answer the questions related to this capture.

2.6.4 TCP

Now, we are going to do the equivalent work of the previous section, but this time using TCP as transport protocol and using a text file and sound file instead of text files.

tcp-quijote.pcapng

1. For this exercise we are going to use the following capture: Download the capture tcp-quijote.pcapng. The trasmission of the previous exercise is replicated on it, but using TCP transport protocol.
2. Open the capture with `wireshark`.
3. Answer the questions related to this capture.

tcp-mp3.pcapng

1. Dowload the MP3 audio in the client directory. The file must be called `p6audio.mp3`:

   ```
   $ wget https://uclm-esi.github.io/redes1-lab/assets/p6audio.mp3
   ```

2. Run the capture with `Wireshark`. If you wish, save it as `tcp-mp3.pcapng` for its later analysis.

   > **Note**
   >
   > You can specify to listen only in the "lo" interface (*loopback*) and use a "tcp" capture filter, so that only the traffic that travels using TCP as transport protocol is captured.

3. In the server terminal, run the following command:

   ```
   $ nc -l -p 8888 > audio_tcp_received.mp3
   ```

4. In the client terminal, run the following command:

   ```
   $ nc -q 2 127.0.0.1 8888 < p6audio.mp3
   ```

   In this exercise, we use `127.0.0.1` again to use the local interface or *loopback*.

5. Stop the capture. Check that both the packets sent by the client and those sent by the server appear in the capture.

   > **Note**
   >
   > Look at the source and destinaton ports to know wich packets belong to each process.

6. Answer the questions related to this capture using as a display filter `tcp.port == 8888`.

2.6.5 Cleanup

Finally, remove the created directory `/home/alumno/p6` with:

```
$ rm -rf /home/alumno/p6
```