

Practice: Basic Commands

In this practice we are going to carry out a series of exercises to check how the command lines work. At first we will start with the GNU / Linux command line and then we will see the Windows command line (PowerShell).

Shell on GNU / Linux

When we open a terminal in graphical mode or log in to one of the ttys (as we have seen previously), what is shown to the user is a command interpreter, or also called *shell*. The shell that has a default user we have it in the file `/etc/passwd`.

The shell always shows us true *Prompt*, which are the characters that are displayed on a command line to indicate that it is waiting for orders. This can vary depending on the shell and is usually configurable.

When we log in with the user *fiddle* In a terminal, what we are shown is something like

this: `smx @ xubuntu : ~ $`

What information do you offer us?

- User is *smx*
- You are on a machine whose name is *xubuntu*
- Your current directory is: `~`
- This is a user other than **root**

In the event that we have logged in with the user **root**, the prompt would have changed to look like this:

`root @ xubuntu : ~ #`

What information do you offer us?

- User is **root** (id = 0)

- You are on a machine whose name is *xubuntu*
- Your current directory is: ~
- It's about the user **root**, since it has a "#" just before the order entry.

This text environment where we are and that allows us to enter commands is commonly known as Shell.

This Shell is capable of interpreting a wide range of commands and sentences. It is also a powerful scripting programming language.

GNU-Linux has the philosophy of not forcing the user to use a specific program for each action, but rather that it always gives the freedom to choose the program that we want to use. The same happens with the Shell that we are going to use to access the system. The most used Shell is known as bash, although there are a great variety of them, such as csh, ksh, etc.

BASH Features

Some features worth knowing about bash are:

- Autocomplete** during writing. When typing one or more characters you can press *TAB* With the aim that in case a command, file name or variable can be completed univocally (depending on the context), it will be completed automatically (the rest of the word is written). If there are several possibilities to complete the word, a sound will be heard and pressing *TAB* again will show all the existing possibilities on the screen. If there are many possibilities (by default more than 100), it asks if you want to show them all or not. PowerShell also includes similar behavior but it doesn't work in the same way.
- Command history**. This is a feature of many other shells that allows movement through the last commands executed, either in the current session or in the previous ones. By default, the last 1000 commands are accessed, but can be modified. To move up and down the cursors are usually used.
- Powerful control structures for scripting**. (Batch processes / Scripts). If, for, while, select, case, etc. statements can be used.
- Definition of functions and aliases for commands**. Functions allow you to define programmed subroutines using the bash language. Aliases allow you to

associate

names called commands with certain options and arguments in a more mnemonic or abbreviated way.

Moving through the File System

We are going to begin our journey by using the Terminal with the commands that allow us to navigate through the system folders (from now on also known as *File System*). And execute the basic operations on the files.

Remember that *EVERYTHING IN GNU \ / LINUX is a FILE*, so if we dominate the files we can dominate the Operating Systems.

A terminal at all times is taking a Directory as *Working directory*, that is, as a Working Directory.

To know what it is at all times, we can execute the order `pwd` which will show us the directory where we are at that moment. From now on we will take that *Work Directory* as the basic place from which we will execute the orders and when we see that we are told "Go to *home*" Means that we change our *Work Directory* to that, or more commonly: "Go to that directory."

Cd command

The command `cd` takes care of this task, changes the active working directory of the Shell to the one indicated as *argument*.

Example:

```
cd / tmp /
```

It will change our working directory to `/ tmp /`. If we now run `pwd` We will see what that change of route indicates to us.

Some common uses of `cd`

```
# Lines beginning with '#' are considered  
# in the world of ShellScripts as "comments"  
# that is, lines that are written for beings  
# Humans, not for the machine.
```

```
# They are used a lot in the computer world.
```

```
# Change directory to / var  
cd / var /
```

Go back to the directory where it was (last known).

cd -

Go to the "parent" directory

cd ..

Goes to the current directory (doesn't seem very useful ^ _

^) cd.

Go to the user's home directory

cd ~

Goes to a folder with a path relative to the current directory cd Desktop /

Ls command

From the man page (man ls) we obtain this information:

```
NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

We have already seen that ls displays information about the content of the directory indicated to it (in case no directory is passed as an argument, it will use ./).

Let's see the columns shown:

```
abernlanas@moria:~$ ls -l /var/
total 52
drwxr-xr-x  2 root root    4096 oct 18 07:21 backups
drwxr-xr-x 19 root root    4096 sep  4 19:28 cache
drwxrwsrwt  2 root whoopsie 4096 oct 17 19:28 crash
drwxr-xr-x 82 root root    4096 oct 15 18:21 lib
drwxrwsr-x  2 root staff    4096 abr 15  2020 local
lrwxrwxrwx  1 root root       9 jul 20 20:04 lock -> /run/lock
drwxrwxr-x 15 root syslog   4096 oct 18 09:03 log
drwxrwsr-x  2 root mail     4096 jul 30 08:45 mail
drwxrwsrwt  2 root whoopsie 4096 abr 23 09:38 metrics
drwxr-xr-x  2 root root     4096 abr 23 09:32 opt
lrwxrwxrwx  1 root root       4 jul 20 20:04 run -> /run
drwxr-xr-x  7 root root     4096 sep 23 17:28 snap
drwxr-xr-x  8 root root     4096 jul 21 06:33 spool
drwxrwxrwt  9 root root     4096 oct 18 12:33 tmp
drwxr-xr-x  3 root root     4096 jul 21 06:32 www
```

Task 02

Indicate for each of the following requirements which arguments we should indicate to the command `ls`.

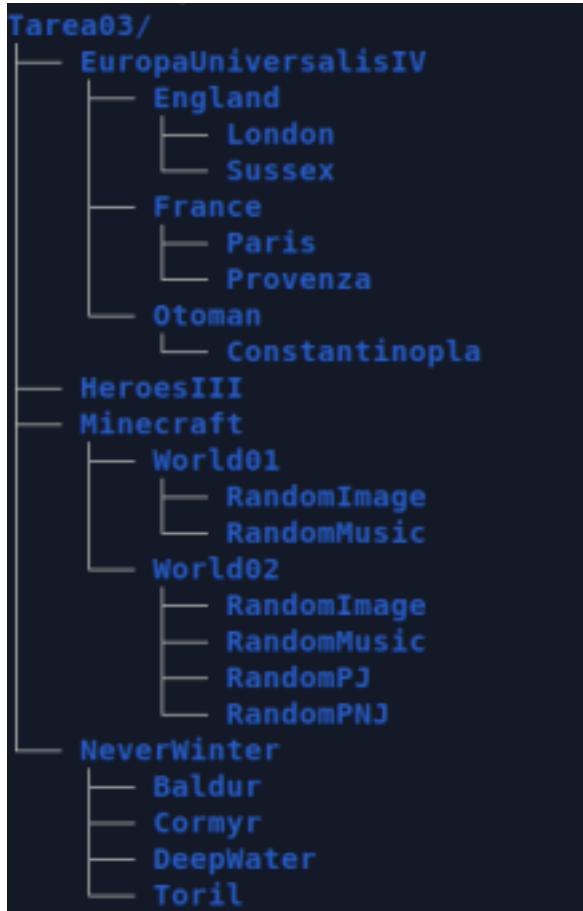
Requirement	Command + Arguments
List the directory <code>/etc</code>	<code>ls /etc</code>
List the directory <code>/var</code> with the options seen before	<code>ls /var</code>
List the directory <code>/tmp</code> no group information	<code>ls /tmp</code>
List the files in the current directory with the extension <code>.txt</code>	<code>ls *.txt</code>
List the files in the current directory with the extension <code>.pdf</code>	<code>ls *.pdf</code>
List the files in the <code>/etc</code> directory that start with "a"	<code>ls /etc/a*</code>
List the files in the <code>/etc</code> directory with the extension <code>.conf</code>	<code>ls /etc/*.conf</code>
List the directory <code>/var</code> by date in reverse order	<code>ls /var -r</code>
List the directory <code>/etc/</code> by size in reverse order	<code>ls /etc -r -s</code>
List the files in the directory <code>/etc/</code> that contain a "u"	<code>ls /etc/*u*</code>
List the files in the directory <code>/etc/</code> containing an "n"	<code>ls /etc/*n*</code>
List the files in the directory <code>/etc/</code> that start with a vowel	<code>ls /etc/vowel*</code>

Mkdir command

This command creates directories.

Task 03

Create the following directory structure in your personal folder.



Attach a screenshot of the command execution `ls -R ~ / Task03 /`, where it should show all the folders created.

Rm command

The command `rm` (ReMove) deletes the files and directories passed to it as arguments. As it is a command that can cause (and does) significant loss of information, it is one of the commands that we have to be most careful when using it.

With great power comes great responsibility

This command is worth an exercise (or at least some time spent on it).

We are going to carry out small steps and we are testing (always in our virtual machine ^ _ ^).

In each of the sections that we will see below there is always a small command / s that

prepare the "test environment", which is where we will then execute the command `rm` so you can see how it works.

Example:

```
# TEST00
# Building an Awesome Workspace ^ _ ^
mkdir -p / tmp / test00 / ThisIsATest
```

```
#Execution
rm / tmp / test00
```

This pattern is repeated in all the examples, you @ s will have to run the section of preparation and then the execution environment and describe what the command is doing`rm`. Some of the executions will have *success* and others not. It does not happen, nothing you must execute and understand *what do you do* and what is failing and write it down after each section.

TRACK: You can use the manual `rm` (`man rm`).

Delivery example after running the example (forgive the redundancy).

Execution of the command `rm / tmp / test00` ~ fails, because it is a directory and `rm` ~ by default only deletes files, unless instructed to do so with an argument.

Complete the following sections.

Task 04

```
# TEST01
# Building an Awesome Workspace ^ _ ^
mkdir -p / tmp / test01 / This
touch / tmp / test01 / isATest
```

```
#Execution
rm / tmp / test01 / *
```

Does not let you delete because it is a directory, besides that the command is badly written, that is to say, before `rm` there should be a `-r`.

```
# TEST02
# Building an Awesome Workspace ^ _ ^
mkdir -p / tmp / test02 / ThisIsA
touch / tmp / test02 / Test
```

```
#Execution
```

```
rm / tmp / test02 / T *
```

Does not let you delete because it is a directory, besides that the command is badly written, that is to say, before rm there should be a -r.

```
# TEST03
# Building an Awesome Workspace ^ _ ^
mkdir -p / tmp / test03 / ThisIsATest / And / this / and / This /

#Execution
rm -r / tmp / test03
```

In this test everything can be executed effectively.

```
# TEST04
# Building an Awesome Workspace ^ _
^ mkdir -p / tmp / test04 /
touch/tmp/test04/example1.txt
touch/tmp/test04/example2.txt
touch/tmp/test04/example3.txt
touch/tmp/test04/example4.txt
```

```
#Execution
rm -i / tmp / test04 / *
```

In this test everything can be executed effectively.

```
# TEST05
# Building an Awesome Workspace ^ _ ^
mkdir -p / tmp / test05 /
touch/tmp/test05/example1.txt
touch/tmp/test05/example2.txt
touch/tmp/test05/example3.txt
touch/tmp/test05/example4.txt
```

```
#Execution
rm -I / tmp / test05 / *
```

In this test everything can be executed effectively.


```
# TEST06  
# Building an Awesome Workspace ^ _^  
mkdir -p / tmp / test06 /  
touch/tmp/test06/example1.txt  
touch/tmp/test06/example2.txt  
touch/tmp/test06/example3.txt  
touch/tmp/test06/example4.txt  
  
#Execution  
rm -vi / tmp / test06 / *
```

In this test everything can be executed effectively.

Clear & history commands

Task 05

Using the man pages (`man clear` and `man history`). Translate into Spanish the / "Description" / of both commands.

Man clear:

Clear limpia su pantalla si esto es posible, incluyendo su búfer de desplazamiento (si la capacidad extendida "E3" está definida). **clear** busca en el entorno para el tipo de terminal dado por la variable de entorno **TERM**, y luego en la base de datos **terminfo** para determinar cómo borrar la pantalla.

Clear escribe en la salida estándar. Puede redirigir la salida estándar estándar a un archivo (lo que evita que **clear** borre la pantalla), y luego pantalla), y más tarde cat el archivo a la pantalla, limpiando en ese en ese momento.

Man history:

Muchos programas leen la entrada del usuario línea por línea. La librería GNU History es capaz de llevar la cuenta de esas líneas, asociar datos arbitrarios a cada línea, y utilizar la información de las líneas componiéndolas nuevas.

Task 06

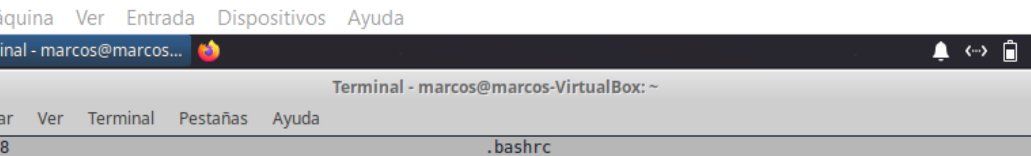
There is a more suitable file for collect all the user's custom aliases. This file is indicated in the file itself "`~ / .bashrc`".

- What file is it?

It's a shadow script

Create a custom alias file with the following aliases:

- An alias "pen" that changes the current directory to the one on the pendrive.



xubuntu [Corriendo] - Oracle VM VirtualBox

Archivo Máquina Ver Entrada Dispositivos Ayuda

Terminal - marcos@marcos...

Terminal - marcos@marcos-VirtualBox: ~

Archivo Editar Ver Terminal Pestañas Ayuda

GNU nano 4.8 .bashrc Modificado

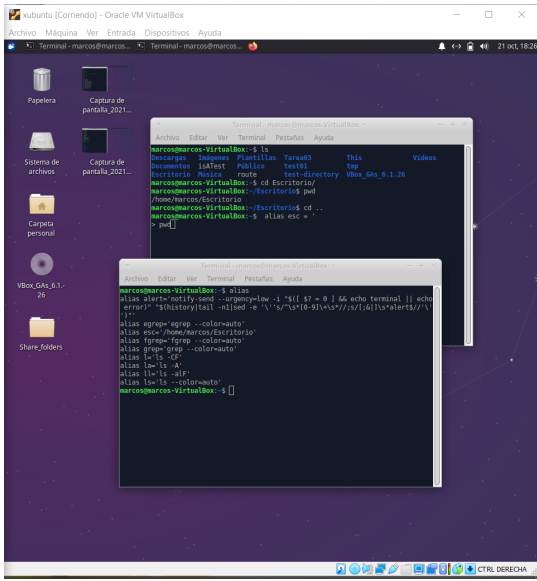
```
alias egrep='egrep --color=auto'

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'
alias pen='ls /media/'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "${ $? = 0 } && echo terminal || echo error" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\s*\>'
```

- An alias “esc” that changes the current directory to the one on the desktop.



- An alias "lh" that shows the files in the current directory in list mode, and with the size "human readable"

