COMPUTER ARCHITECTURE and OPERATING SYSTEMS (CAOS)
Bioinformatics
DAC, FIB, UPC

ShellLab **Spring 2026**

1 Previous work

Objectives

The objective of this session is learning to get along with the laboratory work environment. We will see some operations that can be done either with the command-line or using the window manager. We will centre in the practice of some basic commands and the usage of the online manual (man) that you will find in all the Linux machines.

Skills

– Be able to use `man` pages
– Be able to use the basic system commands to modify/navigate around the file system: `cd`, `ls`, `mkdir`, `cp`, `rm`, `rmdir`, `mv`, `df`, `ln`, `namei`, `readlink`, `stat`, `mount`
– Know the special directories `.` i `..`
– Be able to use the basic commands and programs of the system to access files: `less`, `cat`, `grep`, `vim` (or another editor)
– Be able to modify the access permissions of a file
– Be able to consult/modify/define an environment variable
– Be able to use some special characters of the shell (command-line interpreter):
  `&` to execute a program in the background
  `>` to store the output of a program (redirecting the output)

Previous knowledge

This session doesn't require previous knowledge.

Guide for the previous work

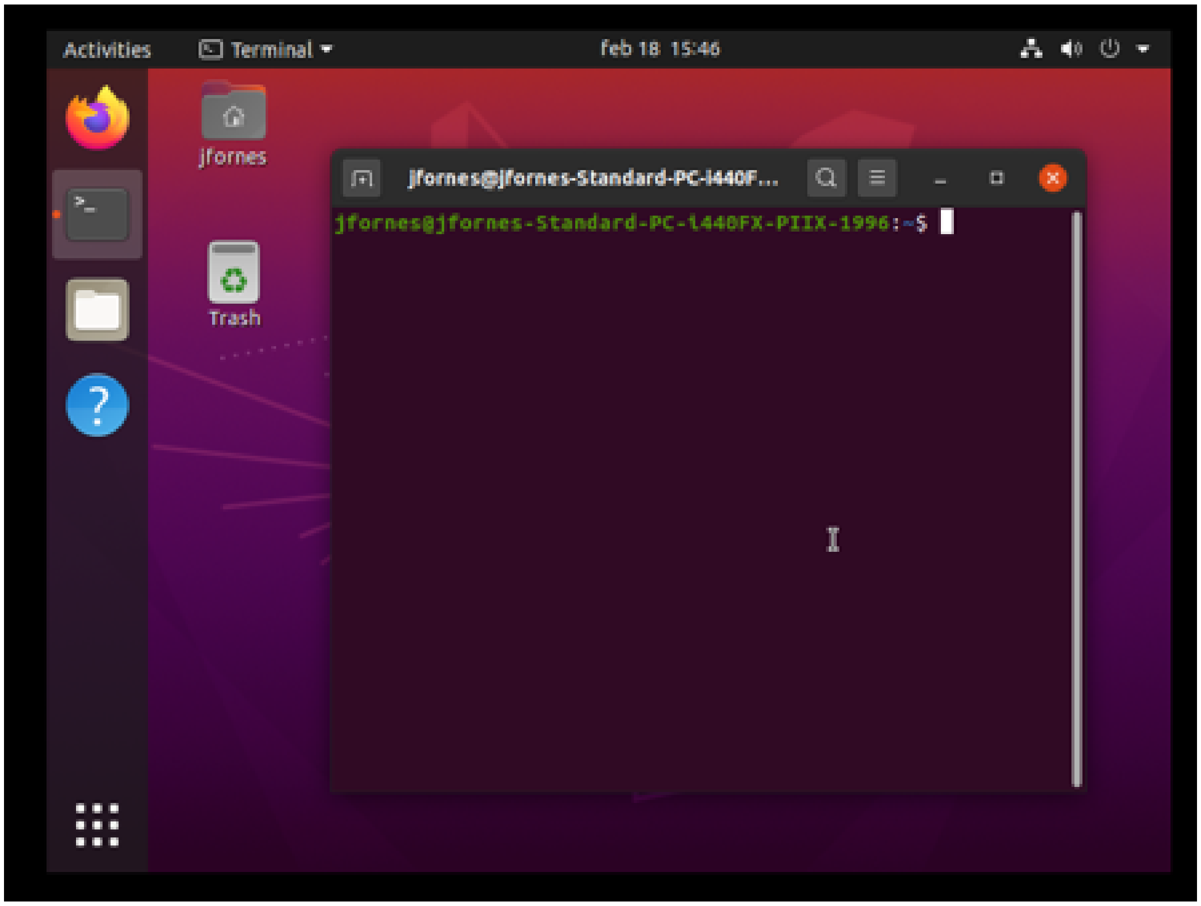
Access to the system

In the FIB laboratories, we have installed Linux OpenSuse (in most rooms) or Linux Ubuntu (only in the OS room). If you have OpenSuse, you can log in with your Racó username and password. In the OS classroom, you can log in with the username `alumne` and password `sistemes`.

To start, we will execute what we call a Shell or a command-line interpreter. A Shell is a program that the OS offers us to work in an interactive text mode. This environment could seem less intuitive than a graphical environment, but it is really simple and powerful.

There are several command-line interpreters in the market, in the lab you will use Bash (GNU-Bourne Again Shell), but in general we will refer to it as Shell. Most of the things that we will explain in this session can be consulted through the Bash manual (executing the command `man bash`).

To execute a Shell we only need to execute the “Terminal” application or the “Konsole” application. With these applications, it opens a new window (similar to the one in the image) where a new Shell is executed.



```
jfornes@jfornes-Standard-PC-i440FX-PIIX-1996:~$
```

Figure 1: **Shell window**

El text que apareix a l’esquerra juntament amb el cursor que parpelleja és el que es coneix com a `prompt` i serveix per indicar que la Shell està llesta per rebre noves ordres o comandes.

TIP

In the laboratory documentation we will use the `#` character to represent the prompt and indicate that what comes next is a command line (to try the line you DO NOT NEED TO WRITE `#`, only the command that appears next to it).

The first thing you should check is that the executing Shell is Bash. You can do this checking by typing in the window the following command:

```
# echo $SHELL
```

This will show you which is the executable launched in the Shell window. If it is not bash, you can launch the bash shell just typing the command:

```
# bash
```

The Shell code could be summarized as the following:

```
while(1) {
       cmd=getCmd();
       runCmd(cmd);
}
```

There are two types of commands: internal commands and external commands. The **external commands** are any program installed in the machine and the **internal commands** are functions implemented by the command-line interpreter (each interpreter implements its own, some of them common and some of them particular to the interpreter).

Commands to obtain help

In Linux, there are two commands that we can execute locally in the machine to obtain interactive help: the `man` command, which offers help about external commands (as part of the installation, the manual pages that we can consult using the `man` command are also installed), and the `help` command, which offers help about internal commands.

Read the guide about "How to use the Linux man" that you have at the end of this section ("Using the manual"). Afterwards, query the `man` (`man command_name`) of the following commands. Specifically, for each command you have to read and understand perfectly: the `SYNOPSIS`, the `DESCRIPTION` and the options that appear under the "Options" column of the table 1.

Use the help command to get help about the internal (built-in) commands in table 2.

Access the man page for bash (executing the "`man bash`" command) and search the meaning of the environment variables `PATH`, `HOME` and `PWD`

TIP

the "`/`" character is used to search patterns in `man` pages. Use it to find the descriptions of these variables).

Using the manual

Knowing how to use the manual is basic since, although some commands will be explicitly explained, you will have to search for the rest yourself in the manual. The manual is self-contained; you can see all its options executing:

```
# man man
```

The manual information is organized in sections. Section 2, for example, is called system calls. The sections that we can find are:

1. commands
2. system calls
3. calls to user or language libraries
4. etc.

The information provided when executing man is what is called a "man page". A "man page" is usually the name of a command, system call or function call. All the pages of the manual follow a similar format, organized in several parts. In figure 2 you have an example of the output of the man for the command ls (we have deleted some lines to be able to see all the principal parts). In the first part you can find the name of the command, its description and a schema (`SYNOPSIS`) of its usage. In this part you can observe if the command accepts options, or if it needs some fixed or optional parameters, etc.

```
LS(1)                                                       User Commands

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.

       Mandatory arguments to long options are mandatory for short options too.

 -a, --all
              do not ignore entries starting with .

SEE ALSO
       Full documentation <https://www.gnu.org/software/coreutils/ls>
       or available locally via: info '(coreutils) ls invocation'

GNU coreutils 8.32                                          February 2024
```

Figure 2: `man ls`, simplified.

The next part is the (`DESCRIPTION`) of the command. This part includes a more detailed description of its usage and the list of options it supports. Depending on the installation of the man pages you can also find here the (`EXIT STATUS`) of the command. Finally, there's usually several parts that include the authors of the manual, the way to report errors, examples, and related commands (`SEE ALSO`).

In figure 3 you have the result of executing `man 2 write`, which corresponds with the system call write. The number that we put before the page is the section where we want to search, and we include it here because there is more than one page with the name `write` in other sections. In this case the `SYNOPSIS` includes the files that need to be included in the C program to be able to use the concrete system call (in this case `unistd.h`). If it were necessary to link your program to some concrete library, different from the default libraries that the C compiler uses, it would be listed here as well. In addition to the `DESCRIPTION`, in the function calls in general (be it a system call or a language library call), we can find a `RETURN VALUE` section (with the values that returns the function) and a special section, `ERRORS`, with the list of possible errors. Finally, we can also find some more additional sections, `NOTES` (clarifications) and `SEE ALSO` (related calls).

The `man` is simply a system tool that interprets some markings in a file text and displays them following the instructions of these markings. The four basic things that you need to know are:

- Usually, a man page occupies several screens, to advance you just need to press the space bar.
- To go to a previous screen you can press the letter **b** (back).

```
WRITE(2)                   Linux Programmers Manual                  WRITE(2)
NAME
       write - write to a file descriptor
SYNOPSIS
       #include <unistd.h>
       ssize_t write(int fd, const void *buf, size_t count);
DESCRIPTION
       write()  writes  up  to  count bytes to the file referenced by the file
       descriptor fd from the buffer starting at buf.  POSIX requires  that  a
       read()  which  can  be  proved  to  occur  after a write() has returned
       returns the new data.  Note that not all file systems  are  POSIX  con-
       forming.
RETURN VALUE
              On  success,  the  number of bytes written are returned (zero indicates
       nothing was written).  On error, -1  is  returned,  and  errno  is  set
       appropriately.   If  count  is zero and the file descriptor refers to a
       regular file, 0 will be returned without causing any other effect.  For
       a special file, the results are not portable.
ERRORS
       EAGAIN Non-blocking  I/O  has  been  selected  using O_NONBLOCK and the
              write would block.
       EBADF  fd is not a valid file descriptor or is not open for writing.
        ….

       Other errors may occur, depending on the object connected to fd.
NOTES
       A successful return from write() does not make any guarantee that  data
       has been committed to disk.  In fact, on some buggy implementations, it
       does not even guarantee that space has successfully been  reserved  for
       the  data.   The  only way to be sure is to call fsync(2) after you are
       done writing all your data.

SEE ALSO
       close(2), fcntl(2), fsync(2),  ioctl(2),  lseek(2),  open(2),  read(2),
       select(2), fwrite(3), writev(3)
```

Figure 3: `man write`, simplified.

- To search a text and directly go to it you can use the character "/" followed by the text. For example "/SEE ALSO" will send you to the first appearance of the text "SEE ALSO". To go to the next appearance of the same text simply use the letter n (next).
- To exit the man page use the letter **q** (quit).

Bibliography

The documentation we give you in this booklet is usually enough to do the sessions, but in each session, we will give you some extra references.

- BASH shell guide: http://tldp.org/LDP/abs/html/index.html

2 Exercises to do in the laboratory

The practises will be done in a Ubuntu 20.04 LTS system or in a OpenSuse 15.4 64 bits system

You have at your disposal a system image equal to the one used in the laboratories to be able to prepare the sessions at home. The image can be used with VirtualBox:

Ubuntu image:
https://softdocencia.fib.upc.edu/software/Ubuntu22v3r1.ova
OpenSuse image
https://softdocencia.fib.upc.edu/software/suse156v1r3.ova
You can download VirtualBox from:

Windows:
https://download.virtualbox.org/virtualbox/7.0.20/VirtualBox-7.0.20-163906-Win.exe
Linux:
https://www.virtualbox.org/wiki/Linux_Downloads
macOS (based on Intel processors[2]):
https://download.virtualbox.org/virtualbox/7.0.20/VirtualBox-7.0.20-163906-OSX.dmg

You can find in the documentation section of the web page of the course a short guide about how to install the environment:
https://docencia.ac.upc.edu/FIB/grau/SO/

▷ Answer in a text file named "to_deliver.txt" all the questions that appear in this document, indicating for each question its number and answer. This document must be delivered through Racó (the FIB's intranet). The questions are highlighted in bold and marked with the following symbol: ⚠

▷ The lines of the instructions that start with the character # indicate commands that you have to try. You do NOT have to write the character #.

▷ **To deliver: file** `shelllab.tar.gz`

```
# tar zcfv shelllab.tar.gz to_deliver.txt
```

Access to the System and Execution of a Shell

You can find the information about how to access to the system and how to launch a Shell in the previous work section of this session 1.

[2]Mac Mx laptops (M1, M2, etc. based on ARM processors) are not compatible with this software. If you have this type of laptop, please email your instructor for alternatives.

Navigate through directories (folders in graphical environments)

You can observe that the vast majority of the basic Linux commands are 2 or 3 letters that synthesize the operation to realise. For example, to change the directory, you have to use the command `cd`. To see the contents of a directory (list directory), you have to use the `ls` command, etc.

In Unix based systems, the directories are organized in a hierarchical way. The base directory is the root (represented with the character `/`) and from there hang the rest of the directories of the system, where the directories and files common to all the users are situated. Moreover, inside this hierarchy, each user has a directory assigned (*home directory*), though it acts as a base for the rest of his directories and files. When a user initiates a terminal, its working directory is its home directory. To change the working directory, you can use the command `cd`, which lets you navigate through the file system hierarchy.

Next, do the following exercises using the commands that you think are more appropriate:

1. In the Documents folder of your home directory[3], create the directories for the first five sessions of the course with names S1, S2, S3, S4 and S5.

   ⚠ Question 1

   What commands have you used to create the directories $S1 \cdots S5$?

2. If you open the *File Browser* of Ubuntu, and go to the same "folder" where you are in the Shell, you should see an image showing the content of the folder (the five directories you just created).
3. In the Shell, change the current work directory to directory S1.
4. List the contents of the directory. Apparently, there's nothing there. However, there are two "hidden files". All the Unix files that start with the character "." are hidden files, and they are usually special. Look up what options do you need to add to the command to see all the files. The files that you can see now are:
   - Directory type file "`.`": It references the same directory where you are at the moment. If you execute (`cd .`) you will see that you are still in the same directory. We will see its utility later.
   - Directory type file "`..`": It references the directory of the immediate upper level from where we are. If you execute (`cd ..`) you will see that you change to the previous directory.
   - Note that these hidden files don't appear in the graphical environment: if you access the S1 folder it appears empty (in the graphical environment, it also exists a configuration option for folders that allows the display of hidden files).

   ⚠ Question 2

   What commands do you use to list the contents of a directory? What option do you have to add to display the hidden files?

[3]The computers in the teaching "general labs" (running OpenSUSE) are configured so that the only directory where data is preserved after shutdown is the dades directory; data stored elsewhere is not guaranteed to persist. The Documents and Downloads directories in your home folder are actually symbolic links to dades/Documents and dades/Downloads, so their contents will be preserved. On the other hand, the computers in the teaching "systems labs" (running Ubuntu) do not keep any data after shutdown, so if you want to keep your files, you must save them elsewhere.

5. The options of the commands can usually be accumulated. Look up in the manual which option do you have to use to see extended information about the files and try it out.

> ⚠ Question 3
>
> What option do you have to add to `ls` to see the extended information of the files? Which fields are displayed by default with this option? (If you cannot find the information in the manual, ask your instructor.)

6. 5. When we use really often a specific configuration of a command, it is usual to use what is known as "alias". It consists in defining a pseudo-command that the Shell can recognize. For example, if we see that we always execute the `ls` command with the options "`-la`", we can define "`ls`" as an alias in the following way:

   ```
   # alias ls='ls -la'
   ```

   Execute this `alias` command and then execute `ls` without options. Check that the output is the output of the `ls` command with the options `-la`.

7. We can configure the *file browser* in the graphical environment to get an information per file similar to the `ls -la` command. In the file browser, look for the configuration option that selects and formats the information shown per file.

> ⚠ Question 4
>
> Which options of the menu have you activated to extend the information displayed by the *File Browser*?

8. From the Shell, delete some of the directories that you have created, check that they don't appear and create them again. Look up how to do that in the graphical environment as well.

> ⚠ Question 5
>
> Which command sequence have you executed to delete a directory, check that it's not there and create it again?

Basic file system commands to access files

1. 1. Create a file. To create a file that contains any text we have several options, for example opening the editor and writing anything:[4]:

   ```
   # gedit test
   ```

2. 2. To be able to execute any other command you will see that you need to open another Shell because the one you had is blocked by the editor (this only happens when opening the first file, not if you had the editor already opened). This is so because the Shell, by default, waits until the current command finishes before showing the prompt and processing the next command. To avoid needing an opened Shell for each command that we want to execute simultaneously, we can ask the Shell to execute the commands in background. When we use this method, the Shell executes the command and immediately shows the prompt and

[4]There are many editors. One of the favorites of programmers, due to its efficiency and versatility, is `vi`; or its improved version, `vim` (*vi improved*). If you don't have an editor at hand, you can always use `cat`, redirecting its output to a file and pressing Ctrl+D when you're done editing.

starts to wait for the next command (without waiting for the previous one to finish). To execute a command in the background you have to add at the end of the line the special character "&". For example:

```
# gedit test&
```

3. To see in a quick way the contents of the file, without opening again the editor, we have several commands. We will mention two here: `cat` and `less`. Add to the test file 3 or 4 pages [5] of text (anything). Try the commands `cat` and `less`.

⚠ Question 6

What difference exists between `cat` and `less`?

4. Copy the test file several times (adding a different number at the end of the name of each destination file, e.g. "test2"). What would happen if the source and destination files had the same name? Look in the man the option "`-i`" of the command `cp`. What does it do? Create an alias of the command `cp` (call it `cp`) that includes the option `-i`.

⚠ Question 7

What's the function of the `-i` option in the `cp` command? Which is the command to do an alias for the cp command that includes the `-i` option?

5. Try to delete some files that you have just created and change the name of others. Make an alias for the `-i` option of the rm command (call it `rmi`). Check also the `-i` option of the `mv` command.

⚠ Question 8

What does the `-i` option of the `rm` command? And the `-i` option of `mv`? Write the command to do an alias of the `rm` command that includes the `-i` option.

6. Another really useful command is the `grep` command. The `grep` command allows the search of a text (explicit or through a pattern) in one or more files. Add a word in one of the files that you have copied and try the grep command to search that word. For example, add the word "hello" to one of the files and we do this test:

```
# grep hola test test1 test2 test3 test4
```

7. The `ls -l` command also allows the visualisation of the permissions of a file. In UNIX, the permissions are applied in three levels: the owner of the file (u), the users in the same group (g), and the rest of users (o). And they reference three operations or access modes: read (r), write (w) and execution (x). For example, if in the current directory there's only file f1, and this file has read and write permissions for the owner of the file, read only for the members of the group and read only for the rest of the users of the machine, the execution of the command would give the following output:

```
# ls -la
drwxr-xr-x  26 alumne  alumne884  2011-09-15 14:31 .
drwxr-xr-x   3 alumne alumne102  2011-09-15 12:10 ..
-rw-r--r--      1  alumnealumne300 2011-09-15 12:20 f1
```

[5] if the file you have created is not big enough you won't see any difference.

The first column of the output indicates the type of file and its access permissions. The first character encodes the type of file (the character 'd' means directory and the character '-' means data file). Next, the first group of 3 characters represent, in order, if the owner has read permission (using the 'r' character) or doesn't have it (then the character '-' shows up), if the owner has write permission (character 'w') or he cannot write (character '-') and if he has or not permission to execute it (character 'x' or character '-'). The second group of three characters are the permissions that the members of the owner's group have and the last group of 3 characters are the permissions of the rest of users of the machine.

These permissions can be modified using the `chmod` command. The `chmod` command offers several ways to specify the access permissions, a very simple way consists in indicating first the users that are going to be affected by the permission change, how do we want to change these permissions (adding, removing or directly assigning) and the affected operation. For example the command:

```
# chmod ugo+x f1
```

Would modify the permissions of f1, activating the execution permission over f1 for all the users of the machine.

The command:

```
# chmod o-x f1
```

Would modify the permissions of f1 removing the execution permission for users that are not the owner of the file nor belong to its group.

And the command:

```
# chmod ug=rwx f1
```

Would change the f1 permissions to the given ones: read, write and execute for the owner and its group members.

TIP

If you are working in the teaching "general labs" (running OpenSUSE), check in which directory you have the test file. If the test file is in the dades folder (or in the `$HOME/Documents` folder or in the `$HOME/Downloads`), copy this test file to your home directory to execute these commands[a].

[a] The reason for making this copy is that in these labs, the dades directory is managed using a Windows file system, which is incompatible with the `chmod` command. If you `chmod` to change permissions on files within that folder, it will have no effect. However, your home directory on those machines is part of a Linux file system.

Modify the permissions of the test file to have only write permission for the owner, its group, and the rest of the users, and try to do a `cat`. Modify again the permissions of the test, leaving only read permissions for the owner of the file, its group, and the rest of the users, and try to delete it.

Question 9

What options of the `chmod` have you used to leave only write permissions? What was the result of cat when trying to see the test file? What options of the `chmod` have you used to leave only read permissions? Have you succeeded in deleting it?

File system

For each file in a UNIX-like system (Linux, FreeBSD, macOS, Android, etc.), a data structure called inode is maintained. You can view the contents of an inode with the `stat` command. Each inode is uniquely identified by a number (the index in the system's inode table). The inode contains the file data in bytes, but the minimum allocation unit is the block. The block size is decided by the creator of the file system. For example, 512 bytes. In the case of a directory, its data are pairs (file name, inode number) for each file in the directory. The inode also contains the file's metadata (management information): owner, size, permissions, last modification date, etc.

8. Run a command line to display information about the test file. In particular, you should know how many bytes it occupies, how many data blocks it has allocated, the size of a data block, its inode number, and the number of different pathnames to get to this file.

> ⚠ Question 10
>
> Which command line did you run to display the inode information in the test file?

9. The inode number also appears in the output of the `ls` command, with the appropriate option. Run `ls -lia` and make sure you understand each piece of data that appears.

> ⚠ Question 11
>
> How many links does the S1 directory have?

10. Now we can reinterpret the `mkdir` command. When creating a directory, two new files are automatically created. The '`.`' is a new path to the new directory. And the '`..`' is a new path from the new directory to its parent directory. Therefore, the new directory will have two links, one the '`.`', inside the new directory. The other is the name of the new directory, which is stored inside the parent directory. Such links are called hard links and can also be created between regular files (those with a '`-`' in the first column of an `ls -l`) with the `ln` command.

11. Run the following commands:

```
# echo "this is a test" > pr.txt
# ln -s pr.txt sl_pr
# ln pr.txt hl_pr
```

> ⚠ Question 12
>
> What type is each of the links created: `sl_pr` and `hl_pr`?

12. Run the `stat` command on `pr.txt`, on `sl_pr` and on `hl_pr`. Search the `stat` output for information about the inode, the file type, and the number of links.

> ⚠ Question 13
>
> What is the number of links in each file? What does this value mean? What inode does each file have?

13. Run the commands `cat`, `namei` and `readlink` on `sl_pr` and `hl_pr`.

⚠ Question 14

Do you notice any difference in the result of the commands when executed on `sl_pr` and when executed on `hl_pr`? If there is a difference, explain why.

14. 13. Now delete `pr.txt` and rerun the `stat`, `cat`, `namei` and `readlink` on both `sl_pr` and `hl_pr`.

⚠ Question 15

Do you notice any difference in the result of the commands when executed on `sl_pr` and when executed on `hl_pr`? If there is a difference, explain why.

⚠ Question 16

Do you notice any difference in the execution of these commands before and after deleting the `pr.txt` file? If there is a difference, explain why

UNIX, unlike MS Windows, maintains a single file system tree, where you can root (`mount`) other file systems. Each file system is associated with a logical partition (`/dev/sda1`, for example). Therefore, to find out how much free space you have on your filesystem, you will have to take a look at what you have *mounted* and where.

15. Try the following commands and consult their `man` page carefully.

```
# df -h
# mount
```

⚠ Question 17

How can you find out which file systems are mounted on your system and what type they are? Also indicate in which directories they are mounted.

⚠ Question 18

How can you find out the number of free inodes on a file system? Which command did you use, and with which options?

⚠ Question 19

How can you find out the free space on a file system? Which command did you use, and with which options?

Environment Variables

The programs are executed in a given environment or context: they belong to a user and a group, from a concrete directory, with a system configuration regarding limitation, etc. More details about the context or environment of a program will be explained in the processes' unit.

In this session we will introduce the environment variables. The environment variables are similar to the constants that can be defined in a program, but they are defined before starting the program, and they usually reference system aspects (e.g. default directories), and they mark some important aspects of its execution, since some of them are used by the shell to define its functioning. They are usually defined in capital letters, but it's not mandatory. These variables can be consulted during the execution of a program through functions of the C library. To consult the meaning of the variables defined by the Shell, you can consult the manual of the Shell you are using, in this case `bash` (`man bash`, section `Shell Variables`).

16. Execute the `env` command to see the list of the variables defined in the current environment and their values.

    To indicate to the Shell that we want to consult an environment variable, we have to use the `$` character before the name of the variable, with the purpose of not confusing it with some text chain. To see the value of a concrete environment variable, we use the command `echo`:

    ```
    #echo $USERNAME
    #echo $PWD
    ```

17. Some variables are dynamically updated by the shell; for example, change the directory and consult again the value of `PWD`. What do you think is the meaning of this variable?
18. Check the values of the variables `PATH` and `HOME`.

    ⚠ Question 20

    What's the meaning of the environment variables `PATH`, `HOME` and `PWD`?

    ⚠ Question 21

    The `PATH` variable is a list of directories. What character acts as a separator between directories?

    We can also define or modify an environment variable using the following command (to modify it, the `$` character is not used):

    ```
    # export VARIABLE_NAME=valor
    ```

19. Define two new environment variables with the value you want and check their value.

    ⚠ Question 22

    Which command have you used to define and check the values of the new variables you have defined?

20. Download the `shelllab-files.tar.gz` and copy it to the folder `S1`. Unzip it using the command `tar xvfz shelllab-files.tar.gz` to obtain the "`ls`" program we will use next.
21. Place yourself in the `S1` folder and execute the following commands:

    ```
    # ls
    # ./ls
    ```

    Note that in the first option, the system command is executed instead of the `ls` command that resides in your directory. However, with the second option, you have executed the `ls` program you just downloaded instead of using the `ls` system command.
22. Add the "`.`" Directory at the beginning of the `PATH` variable using the command (note the directory separator character):

```
# export PATH=.:$PATH
```

Mostra el nou valor de la variable `PATH` i comprova que, a banda del directori "`.`", encara conté els directoris que tenia originalment (no volem perdre'ls). Executa ara la comanda:

```
# ls
```

⚠ Question 23

Which version of `ls` has been executed? The system `ls` or the one you have just downloaded? Execute the "`which ls`" command to check it.

⚠ Question 24

Is the directory where you are in defined in the `PATH` variable? What are the implications of that?

23. Modify the `PATH` variable to eliminate the "`.`" directory. You cannot partially modify a variable; thus, you will have to define it again. Show the current content of `PATH` and redefine it using all the directories that you had except for the "`.`".
    Execute now the command:

    ```
    # ls
    ```

⚠ Question 25

Which program has been executed? The system `ls` or the one you just downloaded? Execute the "`which ls`" command to check it.

24. Add the "." directory at the end of the `PATH` variable (note the directory separator character):

    ```
    #export PATH=$PATH:.
    ```

    Check that, besides the "`.`" directory, the `PATH` variable still contains the directories that originally had (we don't want to lose them).
    Execute now the command:

    ```
    # ls
    ```

⚠ Question 26

Which program has been executed? The system `ls` or the one you just downloaded? Execute the "`which ls`" command to check it.

Keeping the changes: `.bashrc` file

The changes that we have done during this session (except the ones that reference the file system) will be lost when the session ends (alias definition, `PATH` changes, etc.). To not lose them, we have to insert these changes in the session configuration file that the shell uses. The name of the file depends on the shell that we are using. In the case of Bash, the file is `$HOME/.bashrc`. Each time we initiate a session, the shell is configured, executing the commands that are found in that file.

25. Edit the `$HOME/.bashrc` file and add the `PATH` modification that we have asked for in the last section. Also add the definition of an alias to make the execution of the `ls` command using the `-m` option. To change that, you have correctly modified the `.bashrc` and execute the following command:

```
# source $HOME/.bashrc
# ls
```

And check that the output of the ls corresponds with the -m option. The source command causes the execution of all the commands provided as a parameter (it's a way of not having to reboot the session to make those changes effective).

TIP

In the work environment on the FIB's "system labs" (running Ubuntu), the system boots using REMBO. Meaning, it loads a new system image, and therefore all your files are lost and the configuration files are reinitialized with a remote copy. This means that if you reinitiate the session, you will start working with the original `.bashrc` file, and your changes will not be maintained.

Some useful special characters of the Shell

In previous sections we have introduced the `&` character, which is used to execute a command in the background. Other useful characters that we will introduce in this session are

- `*` : The shell substitutes it for any group of characters (except the "."). For example, if we execute (`#grep test t*`) we will see that the shell substitutes the `t*` pattern for the list of all the files that start with the string "t". The special characters of the shell can be used with any command.
- `>` : The data output by default of the commands is associated with the screen. If we want to change this association and make the output redirect to a file, we can do this with the "`>`" character. This action is called "output redirection." For example, `ls > output_ls`, stores the output of the `ls` in the file `output_ls`. Try to execute the previous command. Next, try it with another command but with the same output file and check the content of the file `output_ls`.
- `>>` Redirection of the data output of a command to a file, but instead of removing the content of the file, the output is appended at the end of the file. Repeat the previous example with "`>>`" instead of "`>`" to check the difference.

Question 27

What's the difference between using `>` and `>>`?

Make a backup for the next session

If you are working on the FIB's "system labs", each time a computer is started, a new system image is loaded, and the previous content is erased (they are configured this way). Therefore, it is necessary to back up any changes you make if you want to keep them for the next session. To save your changes, you can use the tar command. For example, if you want to create an archive containing all the files in the S1 directory along with the `.bashrc` file, you can run the following command from your `HOME` directory:

```
# tar zcvf folderS1.tar.gz S1/* .bashrc
```

Finally, you should store this archive in a safe location, such as a USB drive, by emailing it to yourself, or uploading it to cloud storage.

If you are working in the FIB's "general labs", and all the files and directories you created are inside the dades directory, then there is no need to save them elsewhere—they will be preserved for your next login. The same applies to the `$HOME/.bashrc` file or any files or directories created inside `$HOME/Documents` or `$HOME/Downloads`, since all three are actually symbolic links to locations inside dades, and are stored there. However, if you created files elsewhere (such as in `$HOME` or `$HOME/Desktop`), then you should move them to a location within the `dades` directory if you want to preserve them.

If you are working on a virtual machine on your computer, there is no need to make any backups—your data will remain on the image even after rebooting the machine.

To read with `man`	Basic description	Options
`man`	Accesses the on-line manuals	`-k`
`ls`	Shows the contents of a directory	`-l -a`
`alias`	Defines an alternative name for a command	
`mkdir`	Creates a directory	
`rmdir`	Removes an empty directory	
`mv`	Changes the name of a file or moves it to another directory	`-i`
`cp`	Copies files and directories	`-i`
`rm`	Deletes files or directories	`-i`
`echo`	Visualize a text (can be an environment variable)	
`less`	Shows files in a format suitable for the terminal	
`cat`	Concatenates files and shows them through the standard output	
`grep`	Searches text (or patterns of text) in files[1]	
`vim`	Editor for programmers	
`env`	Executes a command in a modified environment, if no command is specified, it shows the environment	
`chmod`	Modifies the access permissions of a file	
`df`	Report information about the file system	`-h -T -l -i`
`ln`	Make links between files	`-s`
`namei`	Follow a pathname until a terminal point is found	
`readlink`	print resolved symbolic links or canonical file names	
`stat`	display file or file system status	`-f`
`mount`	mount a filesystem	

Table 1: Basic commands.

To consult with `help`	Basic description	Options
`help`	Offers information about the internal commands of the Shell	
`export`	Defines an environment variable	
`cd`	Changes the current directory (folder)	
`alias`	Defines an alternative name for a command	

Table 2: Built-in commands