Linux Operating System Environment

Computadors – Grau en Ciència i Enginyeria de Dades

Facultat d'Informàtica de Barcelona – Departament d'Arquitectura de Computadors

This practical session is focused on getting experience in working on the Linux distribution you have downloaded from the FIB. That is, **Ubuntu 20.04 LTS 64 bits**. We will browse over the different tools provided by the system, to obtain information from the operating System (OS), understand the environment, execute commands, and build small scripts to do simple tasks.

Linux environment

The home directory of the Linux session is where the usual Unix configuration files are created and saved (.login, .emacs…). When you open a new session you will be in this directory and you may access the rest of your account by moving upwards in the directory tree (cd ..). Do not remove this directory or you will have problems accessing Linux!

If you are working with the OpenSUSE version provided by FIB, all accounts have a "dades" folder, that has the nice property that it is backed-up every night. Save there (in a subdirectory of "dades") the work that you want to keep. Be aware, anyway, that it is better to do the lab sessions in a folder outside "dades", as the "dades" filesystem is provided by a Windows machine, and it does not work properly for some tasks. For example, configuring and compiling applications on it may fail for unknown reasons.

Accounts are limited in the amount of data that can be stored in each filesystem area. This is known as the account quota. You can check the quota available at your RACÓ account. If you need temporary space you can use the "/tmp" directory during the current session, but its contents will be cleared when you restart the PC.

(https://www.fib.upc.edu/ca/la-fib/serveis-tic/entorn-de-treball-als-ordinadors)

Working with the Shell

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 more powerful.

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

To execute a Shell we only need to open a terminal, for example using the "konsole" application (an icon in the desktop). If the icon is not there, you can write "konsole" after click on the SUSE's icon (bottom-left of the desktop). This application opens a new window where a new Shell is executed. By

default, it opens the "tcsh" Shell. We suggest to use "bash". Thus you can launch it by running "/bin/bash".

You can also try to change your default shell to bash, which is currently the most used one in Linux environments, using the *chsh* command. Unfortunately, in the FIB environment *chsh* is not working because users information is provided by the distributed service LDAP, and not the traditional /etc/passwd file:

```
$ chsh
Password:       <enter your password here>
Changing the login shell for <your username>
Enter the new value, or press ENTER for the default
      Login Shell [/usr/bin/tcsh]:    /bin/bash
Unable to modify the user's shell, because the user is not in the
passwd file.
```

If it would work, the new shell will be started at the new login, so you will need to logout and login again, in order to have the change in effect.

All shells have 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 their own, some of them common and some of them particular to the interpreter).

Getting help

In Linux, there are several common 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), the **info** command, provided by GNU, and the (bash) **help** command, which offers help about bash internal commands.

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 chapters. The information provided when executing man is what is called a "man page". A "man page" is usually the name of a **command** (**chapter 1**), **system call** (**2**), **function/library call** (**3**), **system driver** (**4**), **configuration file** (**5**), **game** (**6**), **system component** (**7**), or **administrative command** (**8**). There may be chapters named "1p", "3p", indicating that the specific commands, tools, functions follow the POSIX standard (https://pubs.opengroup.org/onlinepubs/9699919799/). All the pages of the manual follow a similar format, organized in several sections.

In the first section 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. 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 find also 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.

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).
- 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).

Browsing folders

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), thought to act as a base for the rest of the user's directories and files. When a user initiates a terminal, its working directory is the home directory. To change the working directory you can use the (internal) command **cd**, which lets you navigate through the file system hierarchy.

Exercise 1

Create a tree-based folder hierarchy to save exercises performed in every session throughout the course. Do it with the command "mkdir" and with the file browser (graphical environment). Practice also removing folders with "rmdir".

List the contents of the directory with the list (*ls*) command. 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).

Exercise 2

Select the proper icon in your desktop to browse folders, and go to the folders created above. Modify the options to change the information displayed in the browser.

Basic file edition

To create a file that contains any text we have several options, for example opening the editor and writing something: #gedit test

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 the background**. When we use this method, the Shell executes the command and immediately shows the prompt and 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 &

To see in a quick way the contents of the file, without opening again the editor, we have several commands. We will mention three here: *more*, *cat*, and *less*. Add to the test file 3 or 4 pages of text (you can copy&paste from this document, for example). With it, try the commands *more*, *cat* and *less*.

Copy (*cp*) 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? What does it do? Create an alias of the command *cp* (call it *cp*) that includes the option –i.

Try to delete (rm) 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 move (*mv*) command.

File permissions

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 for user), the users in the same group (g), and the rest of users (o for other). 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, for example, the following output: # ls –la

FTYPE/PERM	NLINKS	OWNER	GROUP	SIZE	DATE	FILENAME
-rw-r--r--	1	xavi	users	17410312	Feb 3 21:12	f1

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.

The permissions can also be interpreted as a bitmap, in such a way there is a 1 if access mode is granted, and 0 if not. Thus, each permission level can present values from 0 to 7, both inclusive, and "chmod" can assign permission sets based on this numeric combination.

Exercise 3

*Create a file called "**answers.txt**" and explain how to modify the permissions of the test file to have only write permission for the owner, its group and the rest of users and explain what happens when you try to do a cat.*

Special characters for the Shell

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

- *: The Shell substitutes the '*' for any group of characters (except an initial "."), matching a file name in the directory where the shell is executing. 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 default output of the commands is associated with the screen. If we want to change this association, and make the output go into a file, we can use 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*. What has happened?
- >>: 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.
- |: also known as "pipe". This character allows to communicate the output of a given command execution (left side) to the input of another command execution (right side).
- ^: this character is used to especify the following character is at the very beginning of a line.

Filtering/Searching

The *grep* command allows the search of a text (constant text or through a pattern) in one or more files. You can check it by the following example. Add a word in one of the files that you have copied, and try the grep command to search for that word. For example, add the word "hello" to one of the files and do this test: #grep hello test test1 test2 test3 test4

Exercise 4

*Answer in the file "**answers.txt**" how to filter the list of the folder contents to show only the directories.*

Filesystems

Files are organized in filesystems. A file system is a collection of directories and files, starting at a top level directory. Compare them to the different partitions known from the Windows operating system: c:, d:, ...

The *mount* command lists the set of filesystems available in your environment, and the directory where they are mounted. For example:

```
proc        on   /proc  type   proc ...   # /proc is the filesystem containing processes information
sysfs       on   /sys   type   sysfs ...   # /sys is the filesystem containing system information
/dev/sda1   on   /      type   ext4 ...    # / is the absolute top (root) of all filesystems
//pax/dades  on   ...                      # the dades filesystem is mounted over the network
libra.fib.upc.edu:/... on /home2/...       # our home directory is mounted over the network
```

Exercise 5

*Look at the content of the root (/) filesystem. Try to understand the use of the different directories you find there (only those at the top level). Explain your findings in the "**answers.txt**" file.*

Configure an abbreviation for a frequent command

When we use really often a specific command with the same arguments, we can simplify our typing by using "aliases". They consist of the definition of a pseudo-command that the Shell recognizes. 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.

You can make this definition permanent by adding it in your **$HOME/.bashrc** file (using bash).

Environment variables

Programs are executed in a given environment or context: they belong to a user, a group, they are executed from a particular directory (current directory), with a system configuration regarding resource limits, etc. More details about the context or environment of a program will be explained in the operating systems and processes session.

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, particular values to be used) and they mark some important aspects of the execution. Even some of them are used by the Shell to define the way it works. They are usually defined in capital letters, but it's not mandatory. The values assigned to these variables can be accessed during the execution of a program through functions of the C library (*getenv*). To learn about the meaning of the variables used by the Shell, you can look for them in the manual pages of the Shell you are using. For the bash shell: #man bash, and look for the section Shell Variables.

Execute the "env" command to see the list of the variables defined in the current environment and their values. To indicate the Shell that we want to get the value associated to 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 constant. To see the value of a specific environment variable we use the command echo: # echo $USERNAME or # echo $PWD

We can also define or modify an environment variable using the following command (to modify the variable the $ character is not used): "# export VARIABLE_NAME=value" (without spaces around the '=' sign).

Exercise 6

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?

*Check also the values of the variables PATH and HOME. Finally, edit the "**answers.txt**" file and briefly explain the purpose of these variables.*

Sometimes it is interesting to look in detail onto the format of a specific files. A first option is to use the "file" command, that informs the user about the estimated content of the file provided:

```
# file  *.c
program.c:              C source, ASCII text
extractsymbols.c:       C source, UTF-8 Unicode text
```

A second option is to observe the exact content of files in octal format:

```
# od    -t    cd1x1        program.c        # displays the contents in characters(c),
                                             # decimal bytes(d1) and hex bytes(x1)
0000000    #    i    n    c    l    u    d    e         <    s    t    d    i    o    .
          35  105  110   99  108  117  100  101   32   60  115  116  100  105  111   46
          23   69   6e   63   6c   75   64   65   20   3c   73   74   64   69   6f   2e
0000020    h    >   \n    #    i    n    c    l    u    d    e         <    s    t    d
         104   62   10   35  105  110   99  108  117  100  101   32   60  115  116  100
          68   3e   0a   23   69   6e   63   6c   75   64   65   20   3c   73   74   64
...
```

A third option is to use "xxd" to show the content in hexadecimal format.

Exercise 7

Execute the command line "echo hello world > file.txt" to create a file called "file.txt" with the content "hello world". Then compare the output of commands "file", "od", and "xxd" to check the contents of "file.txt" and explain the conclusions you draw out of it in the "answers.txt" file.

Upload the Deliverable

To save the changes you can use the tar command as follows:

```
#tar czvf session1.tar.gz answers.txt
```

Now go to RACÓ and upload this recently created file to the corresponding session slot.