Architecture et Système 2020-2021

Département Informatique, ENS Paris Saclay

TP 01 (part one of two) : The command line in Linux

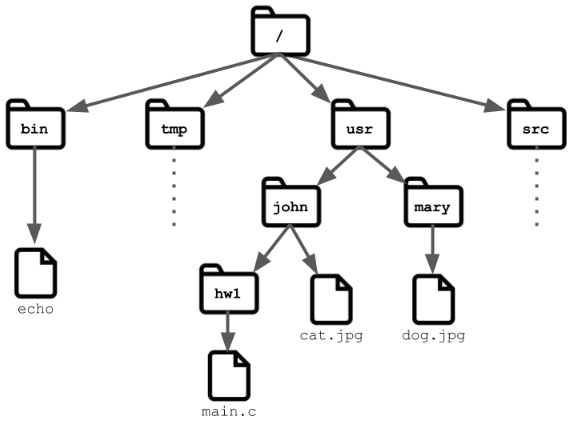
Introduction: This is the first of a two-part introduction to the GNU/Linux operating system installed on the machines of the Computer Science department. It gives you some basic knowledge that you will use throughout your courses. All notions discussed here will be considered as acquired and mastered for all future courses regardless of the module.
This first TP invites you to (re)discover the command line (shell).
Connecting to your session gives you access to your work environment. You will need to open a shell (Terminal) to enter the instructions of this TP.

1 First steps

1.1 File System

In Linux everything is a file, where directories are files that contain names of other files. In order to manage these files in an orderly fashion, we like to imagine them as an ordered tree. Each vertex is a file and folders(which are also files) are vertices that can have other children. The root of the file system is the "root" folder, denoted by "/". Under it you will find all the rest of the files. Let's go over few of the important(for this TP) sub-directories of the root :

— /**home** : Where you will find your users' personal directories.
— /**home/"username"** : Your private folder. In most shells(explained soon) the character "~" is interpreted as the path to your privet directory.
— /**bin :** Here you will find most of the commands(applications) that we'll use during this TP.



1.2 Shell and Terminal

A Terminal is a type of "text input/output environment", which means it can read your textual input and output some text on the screen. In it we'll input our commands to the computer, and get results. To open the terminal on you computer(one of the ways), you press "Ctrl+Alt+T".
The terminal doesn't perform your commands, but sends them to a Shell. The shell is a "command line interpreter", meaning it receives your textual commands and tries to interpret and preform your commands.

You can think of it as the steering wheel of you computer. There are multiple types of shells(Sh,Zsh,Ksh...), we are going to use Bash.

A few tricks that can be useful while using the shell :

— **Tab** : While writing you command you can click the Tab button, and the shell will try to auto-complete your command. If it has a multiple options for completion it will do nothing. But, clicking it twice will print out all of the options it found.

— **Ctrl+C & Ctrl+V** : You'll find out that these two shortcuts don't work 'normally'. This is partially since "Ctrl+C" is designated for stop running process. Instead you can use "Ctrl+Shift+C" and "Ctrl+Shift+V".

— **Special folder shortcuts :** As described earlier, "~" is understood as the address of your home directory. Other shortcuts are "**.**" (current folder), "**..**" (previous folder).

2 Command line : first steps

Since the shell is a command line interpreter, to make the shell do something you'll need to write a command and send it to the interpreter by pressing "Enter". For example enter the command "ls", what do you think it does ?

Some commands have "flags" or "option argument strings" usually beginning with "-". The flags modify the behavior of the program being invoked. For example enter the command "ls" with the flag "-l", what changed ?

2.1 Finding help

One of the ways of getting help with the commands on Linux, is to RTFM(read the fine manual). The manual can be accessed from the shell via the **man** command(short for manual).

2.1.1 Man pages

Run the command : **man man**. What do you see ?
To search for text, press the key "/" followed by your search query, use the keys **n** and **p** to navigate between different occurrences.
To quit the man page press "q."

2.1.2 Using the man pagesl

All pages in **man** are organized in the same way.

1. NAME : name of the command ;
2. SYNOPSIS : short summary of the syntax of the command ;
3. DESCRIPTION : long or short description of the command ;
4. OPTIONS(optional) : description of supported options ;
5. There is of course the possibility to put as many subsections as you want ;
6. AUTHOR : the people who developed the command ;
7. SEE ALSO : Cross references.

2.1.3 Manual of the command ls

Read the help page for the command "ls" and try to find information that answers the following questions :

1. In no more than 10 words, what is the purpose of this command ?

2. Which will print all the files (even those hidden : those whose name starts with a point) ?
3. Which option will print the files with a long listing format ?
4. Which option will print the size of the files in a human readable way ?
5. Which option will print the subdirectories of the given directory recursively ?

2.1.4 Other ways to get help

There are other option to find help :

1. Adding the options - -help in the end of a command gives you a short explanation about this command(mkdir –help).
2. The command "whatis" displays a one-line description of the command following it (whatis ls).
3. Using your favorite search engine on the internet, e.g. www.duckduckgo.com, www.bing.com(really ?), www.google.com,. . .
4. Asking a fellow human.

2.1.5 The command cd

— Find out what this command is for.
— Go to the **tmp** directory at the root of the system and list its contents
— Return to your home directory
— What does **cd -** do ?

2.1.6 Wildcards

A **wildcard** is a symbol that takes the place of an unknown character or set of characters. Some of the heavily used wildcards are :
— The asterisk "*", represents any number of unknown characters. For example "*cake" can stand for "carrotcake, chocolatcake,cake,..." and "cake*" can stand for "cakeParty, cakeWorld,cake,...".
— The question mark " ?", represents only one character. For example " ? ?ke" can stand for "cake,bake,take,. . ." and "ke ? ?" can stand for "keen,kept,keep,. . .".

Use the "wildcard" "*" to list all the files starting with "u" in the root of your system.

2.1.7 Manuel of the command mkdir

Consider that we want to create at the root of your account the following path :

```
ArchiSys/tp/01
```

Launch the following command and observe what is happening :

```
cd ; mkdir ArchiSys/tp/01
```

Look in the "mkdir" man page for the option to create the entire directory hierarchy in one command.

2.2 The command echo

What is the command "echo" used for ?
What does the following command do ? Why ?

```
echo -ne "\n\n *\tHello World $LOGNAME\t\t*\n\n"
```

Try the following commands, what is the difference ? Why ?

```
echo -e '\n\n *\tHello World $LOGNAME\t\t*\n\n'
```

2.3 Other commands

In the following, we will use the command "ps" which lists the programs running on the machine and "cat" which reads an input and prints it on the standard output. See the manual pages for these commands if they are not familiar to you.

3 Redirections

The commands that we have seen so far send their outputs to the so-called **"standard output"** (stdout). By default, that is the terminal. However, it is possible to change the destination of the standard output for a command and for example print the result of a command into a file (this practice is called redirection). Note that programs may also produce output in places other than standard output.

In general, most programs work as follows :
— if one or more files are specified, the program works on these files ;
— if no file is specified, standard input is used.

3.1 First steps in redirection

Execute each of the following commands(in order) in your terminal, examine the results and deduce what happens (it will be necessary to look at the contents of the files generated after each command) :

```
ls
ls > file1
pwd > file1
ps aux > file2.txt
cd > file3
cat file1 file2.txt > file4
cat file1 >> file1
pwd >> file1
cat file1 > /dev/null
```

What are the key words «>» and «>>» used in the previous examples ?

3.2 The pipe

The pipe is also a form of redirection, but this time instead of redirecting the standard output to a file as before, we will redirect it to a command. Note that, just like for standard output, any program has a **"standard input"**, **"stdin"**, which may or may not be used. In the terminal, the standard input is often the keyboard.

3.2.1 Cat

Run the cat command in a terminal. Write a few words on the screen then press "Enter". The line is duplicated. Why ? (to finish press Ctrl+d)

3.2.2 Examples of pipe

Execute each of the following commands(in order) on your device, examine the contents of the manipulated files and deduce what is happening :

```
ls -R > file1
less file1
cat file1
cat file1 | less
ps aux
ps aux | grep -v root
ps aux | grep root
ps aux > file2
cat file2 | grep -v root
```

Additional questions : what is the purpose of grep ? and its '-v' option ?

List the processes that do not belong to "root" but contain the root in their names and write them to a file with the name 'processwithroot' .

3.3 Input redirection

Just as it is possible to redirect the output of a command, it is sometimes interesting to redirect its input. Explain the behavior of the following command :

```
cat <<eob
a
b
c
eob
```

What is the difference between **<**, **<<** and **<<<** ?

Note : Pressing "Ctrl+D" tells the shell that it reached the end of the text.

4 Process Management

We call "process management" the manipulation of programs running from the terminal. There are also graphical interfaces, such as in Windows or in Mac OS, that can do what we will do in this section.

A running program is called a **process**. Processes are identified on the system by a number called "PID" (process identifier).

Under Unix on the command line, a process can be executed in two different ways :

— In the background : While the process is running in the background, you can launch other processes in the shell. The background process can print output to the terminal, but cannot receive input from the keyboard

— Foreground : This is the normal mode of operation. The foreground process receives input from the keyboard, and the shell waits until the process has finished before letting you start other command.

We will see that it is possible to switch a process from one state to another in the terminal, and to kill processes. We will also see that it is possible to list running processes in a terminal thanks to the **"jobs"** command.

4.1 Process in the background

Putting an "ampersand"(&) at the end of a command makes a process launch in the background. Run the following code :

```
xterm &
emacs monfichier &
```

In this case, the Shell executes the xterm command and returns to the user the control after displaying the PID numbered by the shell, then it runs emacs in the same way.

Example of a possible output, after running xterm& :

```
[1] 1360
```

and after running emacs& :

```
[2] 1366
```

In this case, the xterm PID is "1360" and the process numbered "1" in the shell, while emacs (PID 1366) is numbered "2" in this shell.

4.2 Switching from one state to another

When a process is started in "foreground", you can not interact with the Shell that started it. To "get your control back" you have to switch the process to the "background". The combination of keys "CTRL + Z" silences the process running in the foreground, then the "bg" command (stands for "background") wakes up the process and runs it in the background.

Note : You'll find that the "Control" key has different notations. Two of the those are «CTRL» and «ˆ».

Conversely, the last process put in the background (for example with ending the command by "&"), can be switched to the foreground with the command "fg".

Preform and answer the following questions :

— In a terminal launch a new terminal "xterm";
— run the command ls in the new terminal;
— put the new terminal to sleep;
— to run the ls command in the new terminal, what's going on? Why?
— switch the terminal in the background
— relaunch "ls" what's going on? Why?

4.2.1 Application and process states

create a new file with the name "script1" and the following code in it (use cat and a redirection) :

```
while true ; do
 for i in {1..5} ; do
  echo bla
  sleep 1
 done
```

```
  echo -n ">> "
  read a
  echo $a
done
```

Run the following command "bash script1", when the line in terminal displays « >> » enter a character and confirm with "Enter".

After several executions, put the script to sleep. What is happening ?

Wake up the process in the foreground, what's going on ?

Toggle the process in the background. What happens if you type "ls" when ">>" is displayed ?

Re take control of the script process and stop it.

4.3 Kill a process in foreground

To kill a process running in the foreground, enter the sequence "CTRL + C" in the terminal running the process.

4.4 jobs

Using bash's help page in the "JOB CONTROL" section, run the following commands and keyboard sequences and analyze the behavior of the processes (you will of course need two terminals, one for the manpage, the other for exercise).

Note : Remember to reposition yourself in the terminal after launching applications.

```
emacs &
xterm &
firefox &
emacs
Ctrl + C
jobs
fg %3
Ctrl + Z
bg
fg %2
Ctrl + C
kill %1
jobs
```

What process remains after the execution of these commands ?

4.5 Chaining processes

When a program finishes executing, it returns a value related to how the process stopped. This value is usually interpreted as an indication of whether the command succeeded. It can be used to chain commands. To this end we will use the operators "&&" for "AND" and "||" for "OR".

"&&" will run the command on its right side only if the left side "succeeded"' and "||" will run the command on its right side only if the left side "failed". For example command

```
mkdir folder/anotherfolder 2> /dev/null || echo right side evaluated
```

will print "right side evaluated", while the following command :

```
mkdir folder/anotherfolder 2> /dev/null && echo right side evaluated
```

won't print anything.

4.5.1 First process chain

With the "grep" command, test the existence of the "root" account on your system and display the message : "root exists" if the account is found.

Note : The file "/etc/passwd" contains the list of existing accounts on the system.

4.5.2 More advanced chain

Using the operators and what has been seen previously, write a command displaying **ONLY** the text "Firefox is running" on the standard output if the firefox program is running on your machine and "No firefox currently" otherwise.

5 The GNU/Linux operating system

5.1 Overview

— The root of the Linux operating system is marked "/".
— On Linux everything is file.
— A user is a person or program registered on the system (with an account).
— Groups are used to classify users based on criteria defined by the system administrator.
— Users can belong to one or more groups.
— Users have privileges based on their membership groups.
— For the system, a user is a number called "UID"(User identifier) associated with a "login".
— For the system, a group is also a number.
— The user defaults to a group whose number is "GID"(Group Identifier).
— The rights, or permissions, that can be assigned to a file are : no rights noted "-", "read" denoted by "r", "writing" denoted by "w" and execution rights or right to traverse the folder "x".
— Rights can be granted to users, groups or others (people not belonging to the two previous sets).
— Rights are displayed as triplets "rwx" for each set to which they can be granted (in the end we get permissions from « $---------$ », no rights, and « rwxrwxrwx » everyone can read write and access the file).
— The "-l" option of "ls" shows the rights of to the files.
— The "chmod" (change mode) command allows you to modify the file rights.
— The "chown" (change owner) command is used to modify the owner and/or group owning a file.
— On a corporate system, a single user can not elevate his privileges without authorization.
— The home directory of a user is noted "~login". If "login" is omitted, the current user is implied.
— "groups" allows to list the membership groups of a user and "id" specifies their number.
— "pwd" Locates you on the system (displays the current directory you are in).

5.2 Questions

— What are the rights on your personal directory ?
— Which group does your homedir belong to ?
— Which group do you belong to ? and your neighbor ?
— Go to "/root". What is going on ? Why ?

— What is your gid ?
— "chmod" has two ways to represent these permissions with an octal number and with symbols. Give two commands, one with the octal notation, the other with alphabetical notation, to define the following rights on the directory "mydir" (starting rights "rwxr-xr-x") :
  — rwx−−−−−−
  — rwxr-x−−−
— Change the owner of the tmp directory. What is going on ? Explain.

6 The variables

On Unix, the system is able to exchange or retrieve information by using variables. These variables are made accessible by the Shell. Some are operating system variables (environment variables), others are Shell specific. We will study some of them, how to modify them, create them and display them.

6.1 Declaration

The following is a template for declaring a variable :

```
name_of_var=value
```

Where "value" is a string, and "name_of_var" is a string consisting of basic ASCII characters and not starting with numbers. For example :

```
name="Jean"
```

6.2 Viewing

Bash (which is a type of shell) has the command "echo". This command allows you to display the value associated with the name of a variable. Variables must be prefixed by the character "$". The following command will display the value associated with the variable named "name" :

```
echo $name
```

Although this syntax works, the recommended syntax by bash is :

```
echo ${name}
```

6.3 Modification

To changing a variable one needs to overwrite its previous declaration.

6.4 Exercices

1. View the contents of the HOME variable. What do you think this variable represents ?
2. View the contents of the PWD variable. What do you think this variable represents ?
3. View the contents of the PATH variable. What do you think this variable represents ?

4. View the contents of the PS1 variable. What do you think this variable represents ? Look in the help page of bash. Try changing it to something else (for example "Linux is not scary :"). Re-launch a new terminal. What do you notice ?
5. Modify the contents of the PATH variable, then try to run a command. What is going on ?

6.5 String of characters

In the shell you can find strings between double quotes ("foo"), single quote ('foo') or anti-quote (‘foo‘). Test the following commands, and explain what happens :

```
echo $HOME
echo "$HOME"
echo '$HOME'
echo `$HOME`
TEST=ls ; echo "`$TEST`"
test=$(ls) ; echo $test
```

7 Some practical shortcuts

If you use the text editor **emacs**, You will have the chance to reuse some keyboard shortcuts when entering a command [1].

— What does **Ctrl+R** do ?
— What does **Ctrl+A** do ?
— What does **Ctrl+E** do ?
— How do you copy/paste text to or from your device ?

8 Test your knowledge

To go further with the command line, I recommend you go to the following URL : `http://overthewire.org/wargames/bandit/`. Your goal is to go to level 34 ! You have to pass the level n to unlock the level $n+1$. Good luck !

9 Bonus questions

— What does the command **which** do ?
— What is the difference between **>** and **2>** ?
— How to mix these two redirects ?
— What result of the commands **[** an **]** command ? How do you explain it ?
— What does the command **pkill** do ?
— What does the commands **less** and **more** ? do ?
— What does the command **yes** do ? What do you think it can be used for ?
— Search the internet for an answer that explains why the names of hidden files start with a dot.

1. If you are a user of **vi** you have to change the editing mode : `https://sanctum.geek.nz/arabesque/vi-mode-in-bash/`