TP1: Commands and Shell scripting

1 Getting started

Once you have been connected to your account on a Linux machine, open a terminal window from the program menu.

1. In your command line, type **pwd**. It prints the absolute name of your current working directory (where you currently are), i.e. the path from the root of the file system to this directory. With which command you can always go to this directory from anywhere in the file system ?
2. Create a new directory **CUnix** using the command **mkdir**. You should not use neither the space character or any accents for the names. With which command you can verify that the directory has been created ?
3. Access the directory by using the command **cd**. If you have a long name to tape, you can type only the first letters and use the key "tab" to automatically complete the name. If more than one name exists with the same start, a double "tab" provides you the existing options.
4. Use the arrows "up" and "down" to go through the history of the commands already taped in the shell. Display again the absolute name of the current working directory.
5. In Linux, the mouse can be used for copy-paste. Double click the name of the directory that you have displayed on the screen. This selects a word, whereas a triple click selects a line. Then, click with the middle button of the mouse. You can also do a right click and select copy. For paste, you can use the key combination "shift"+"insert".
6. Create a new directory **tp1** inside **CUnix**, which will be the working directory of this lab.

2 The command man

The command **man name_command** provides you the information on the functionality and the parameters of the command **name_command**. The information is displayed page by page. To pass from one page to the other you can simply hit the space key and to go from one line to the next, you can hit the enter key.

Use man to search information about some of the basic commands, such as :
man, man -k,
cd, cd .., ls, ls -a, ls -l, ls -al, pwd,
mkdir, rmdir, cp, cp -R, mv, rm, rm -R,
touch, echo, history, history -c, who, passwd, cat

3 Environmental variables

In order to facilitate the use of files that exist in another directory, you can use a shell environmental variable that stores the path to that directory. This can be achieved through the command **export** :

export name_variable=path_directory

Then, the path to the file can be replaced by a reference to the variable **$name_variable**.

1. Initialize a shell environmental variable called **rcom** with the path **/share/l3info/CUnix/tp1**
2. Use this variable to display the content of the directory **/share/l3info/CUnix/tp1**
3. Use this variable to display the content of the file **/share/l3info/CUnix/tp1/prog1.c**

4 Redirections

1. Copy the contents of the directory **/share/l3info/CUnix/tp1** to your personal space ∼**/Cunix/tp1** using the command **cp**.
2. Open the file **prog1.c** with an editor. You can use the text editor **xemacs**, **gedit**, **vim** etc. You can use the **&** in order to put the editor process in the background and to be able to continue with the execution of the commands in the shell. Can you understand what **prog1.c** does ?
3. Execute the program **prog1** use the **./** inside your personal space for the following cases :
   (a) standard input : keyboard, standard output : screen
   (b) standard input : the file **d1**, standard output : screen
   (c) standard input : keyboard, standard output : the file **res1** inside your working directory
   (d) standard input : the file **d1**, standard output : the file **res2** inside your working directory

5 Extracting system information

The command **ps** reports the process status of the currently running processes of the system. To view all the running processes you have to use the option **-A**.
The command **find** helps you to search for information in the file system.
The command **2>/dev/null** can be used to redirect the error messages to the null device, which it takes any input and throws it away.

1. Create a file **aliveprocs.txt** which has as content the list of the names of all the currently running processes (column Command) of the system. Display the file on the screen.
2. Create an ordered list of all the currently running processes of the system with respect to their Virtual Memory (VM) size (column SZ) . Display this list on the screen.
3. Create an ordered list of the currently running processes of the system which do not belong to the **root**. Display this list on the screen.
4. Create an ordered list of all the currently running processes of the system with respect to the total CPU time. If a process is executed several times, its name should appear only once.

5. Display on the screen the name of the files (not directories) of your personal space which have been modified during the day.
6. Create a file **read-only.txt** which has as content a list with the names of all the files of the system for which all the users have the permission of reading. Do not display the error messages.
7. Create a file **mine.txt** which has as content a list with the name of all the files of the system that belong to the user that executes the command.

6 Script programs

1. Open the file **prog2.c** to understand what the program does. The program inputs are given by the standard input and they consist of an integer "i" and a sequence of characters in the following form :
   ".....%.....%.....%"
   The output is the i^{th} subsequence of characters.
2. Execute program **prog2** to better understand how it works.
3. Write a script **select.sh**, which executes **prog2** and it is called with three parameters :
   select.sh i f1 f2
   The parameter **i** is the integer value. The parameter **f1** is the input file that has the total sequence of the characters. You can use as an input file the already provided file **d2** The parameter **f2** is the output file that has the i^{th} subsequence of characters.
   You can use the command **cat** to implement this script :
   — **cat f1 f2** : sends to the standard output the concatenation of f1 and f2
   — **cat - f** : sends to the standard output the concatenation of the standard input and the f
   Your script should take care of all the verifications, such as the correct number of parameters is passed, the files exists, the permissions are correct etc.
4. Write a script called **selmult** which takes 1 parameter :
   selmult f
   The script continuously reads an integer "i" and copies at the standard output the i^{th} subsequence of characters of the file f (using **prog2**). It stops when the value of the integer "i" is equal to 0.