SysAdmin - Part 1 - CLI & FS

Michel FACERIAS

22 octobre 2024

Polytech Montpellier

Université de Montpellier

Table des matières

1 Understanding CLI

A **command line interface** (CLI) processes commands via a computer program in the form of lines of text. The program which handles the interface is called a command line **interpreter** or sometimes a **console**, improperly a **shell**.

 See more : https://en.wikipedia.org/wiki/Command-line_interface

1.1 Shell command

1.1.1 Concept : What is a command ?

 A **command** is an order given to the operating system through a **command line interface** in a **text console** :

```
$ arg0 [arg1] [arg2] [arg3] ... [argx]
```

It consists of :

— a list of arguments :
  — a **command name** (here **arg0**) ;
  — facultative **arguments** (here **arg1 arg2 arg3 ... argx**) ;
  — separated by **spaces** (or blank characters) ;
— written after a **shell prompt** (here **$**, and # for the *root* user) ;
— executed by pressing the ***enter*** key.

Computer scientists are great because they are lazy (and vice-versa). So, the **shorter** a command name is, the **more** it is used.

Some commands are not **accessible to normal users**.

1.1.2 To do : Kick-off

 Open a console window and type the command below, then press *enter* :

```
$ whoami
```

The system answer by giving your **login name**.

1.1.3 To do : A little more difficult

 In the opened console window, type the command below, then press *enter* :

```
$ whoami --version
```

The system answers by giving some information about the *whoami* program.

1.1.4 Question : Analyse the command line

 In the previous examples :

— what is `whoami` ?
— what is `--version` ?

1.1.5 Concept : Result of a command

 The **result** of the command **may be directly visible**, when it is displayed on the screen.

 But beware :

— a command that displays nothing is not a command that doesn't have an effect ;

— a command that displays something did not necessarily produce the exact effect expected.

Therefore **always check the result produced** ... by using another command !

1.1.6 Concept : Uniline

 It is possible to pass two (or more) commands in the same command line. You must separate them using **;** (semicolon).

It is called **uniline commands**.

1.1.7 To do : Trying uniline

 Try separately `date` and `whoami` commands.

Then type the following dual command and execute it :

```
$ date ; whoami
```

1.1.8 Question : Analyse the result

 From which command does each part of the answer come ?

1.2 Using the console

1.2.1 Concept : Console helpers

 **Don't be afraid** ! Linux console is **user friendly** :

— use BACK, DEL, LEFT, RIGHT keys to **edit** a CLI ;

— use UP and DOWN keys to **recall** already used commands ;

— use TAB key to **autocomplete** a command (2 times if it's ambiguous) ;

— use CTRL+R keys to search in and **recall the last commands** you use ;

— use CTRL+D keys or `exit` command to close a console (and logout if you are using a pure text one).

The Linux console has a high integration level into the graphic interface :

— a graphic environment should only be used to open several consoles :-) ;

— you can use the **mouse to cut/paste**, simply by selecting words and center-click on the CLI to copy/paste ;

— **dragging icons** from a graphic shell to a console **input the object path** in the CLI ;

— you can also **close** a console by clicking its *window close button*

Keep in mind these shortcuts to be more efficient later.

1.2.2 To do : Ask for help

 There is a **manual** that details possible **command operations and options**. Use the next command :

```
$ man whoami
```

This will give you access to the `whoami` manual :
— in this *synopsis* paragraph, `[xxx]` means that `xxx` is an optional argument ;
— use UP, DOWN, PAGE UP and PAGE DOWN keys or the mouse wheel to crawl ;
— use H key for more help about the help ;
— use Q key to quit.

2 Understanding FS

The main role of a **File System** (FS) is to organize data belonging to users and the system itself.


See more : https://en.wikipedia.org/wiki/File_system

2.1 File-system

2.1.1 Concept : Human view


To ease document (and other data) handling, it is better to have access to an abstract view.

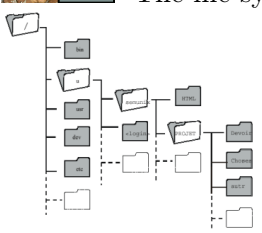
It can be seen as a **hierarchical** sequence :
- of "boxes" (**folders**, or directories) ;
- containing **files** (the documents themselves).

With Linux, we talk about the **Virtual File System** (VFS). It is a **unified view** of folders (which are not compulsorily on the same medium).


See more : https://fr.wikipedia.org/wiki/Virtual_File_System

2.1.2 Concept : Tree structure


The file-system looks like a **tree** :



- Each node is called a folder (or directory) ;
- In each folder you can find :
  1. other folders ;
  2. files.

There is no independent drive like Windows `C:`, so the directory tree is unique !

There is a special folder, the one at the top and called ***root***. It is the **starting point** of the whole structure.

Do not confuse the **root folder** with the **eponymous user** and/or his home directory.

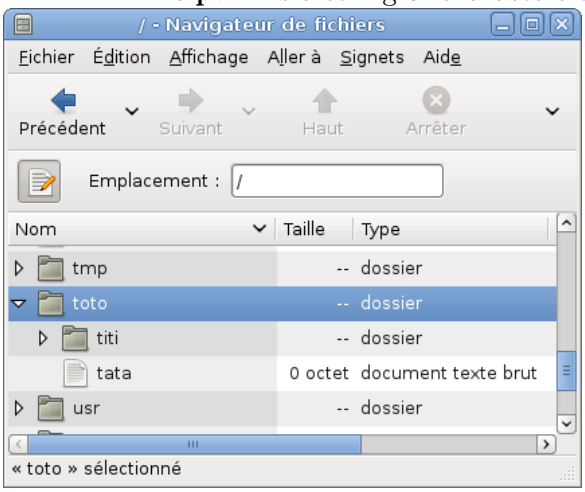
2.1.3 Concept : Files


A **file** constitutes the leaves of the tree :
- it's a **sequence of data** written in binary code ;
- it can contain **different types** of data :
  - **documents** (image, text, formatted text, etc.) ;
  - **programs** (or program elements) ;
  - **links**, which are pointers to another file or folder ;
  - **others**, not detailed here.

With Linux, a **folder** is, in fact, a **special file** whose data is a list, i.e. the files or folders it contains...

2.1.4 Concept : Path

The **path** is a string of characters that identifies a resource in the file-system.



We use the / character to parse each level in the path.
There are 2 ways to identify paths :

— **Absolute** :

1. it starts from the root of the file-system ;
2. it always starts with / ;
3. it is never equivocal ;

— **Relative** :

1. it starts "from where you are" ;
2. it never starts with / ;
3. it can be equivocal.

2.1.5 Question : Find the path (easy)

This is a partial view of a file-system. Labels #A to #I are used to identify files and folders.

```
/
|
|---home
    |---toto
    |   |---tata
    |   |   |---fic1 #A
    |   |   |---fic2 #B
    |   |
    |   |---titi
    |       |---fic3 #C
    |
    |---tata
    |   |---tata
    |       |---fic1 #D
    |       |---fic7 #E
    |       |---fic5 #I
    |---titi
        |---tata     #F
        |   |---fic1 #G
        |---fic5     #H
```

Find the absolute paths :

1. What is the absolute path for #C ?
2. What is the absolute path for #B ?
3. What is the absolute path for #F ?

2.1.6 Concept : Special paths

There are **3 special paths** :

— **..** means *one folder up* ;
— **.** means *here* ;
— **~** means *at home*, i.e. in my personal folder.

There is a command to know **where you are**. This particular place is called the **working directory**.

2.1.7 Question : Find the path (less easy)

This is a partial view of a file-system. Labels #A to #I are used to identify files and folders.

```
/
|
|---home
    |---toto
    |   |---tata
    |   |   |---fic1 #A
    |   |   |---fic2 #B
    |   |
    |   |---titi
    |       |---fic3 #C
    |
    |---tata
    |   |---tata
    |       |---fic1 #D
    |       |---fic7 #E
    |       |---fic5 #I
    |---titi
        |---tata     #F
        |   |---fic1 #G
        |---fic5     #H
```

The current directory is `/home/titi`, find the relative paths :

1. What is the relative path for #H ?
2. What is the relative path for #G ?
3. What is the relative path for #D ?
4. What is the relative path for #A ?
5. What is the relative path for #E ?
6. What is the relative path for #I ?

2.2 File and folder handling

2.2.1 Concept : File and folder naming

Files and folders names are not spelled randomly :

— avoid spaces, replace them by _ or - (You will thank me when you will make scripts !) ;
— you should not use accents (Only Windows is not capable of dealing with UTF8 encoding, but it's a good practice) ;
— make distinction between lowercase and uppercase (Only Windows falsifies letter case) ;
— avoid meta-characters that have a special meaning (seen in next part) ;
— never use . (dot) as the first character of the name (because it makes an hidden file/folder).

With Linux, there is **no extension** like *.jpg* or *.htm*. Files are identified by their firsts bytes. But, we choose to use fake extensions for the sake of human readability.

2.2.2 To do : Just a kind of magic... number

We are going to identify a file type using the magic numbers. Open a text console and a graphic file manager. Choose a *pdf* (or download one).

In the console, type `file` followed by a space. Then drag the *pdf* file to the text console to complete the command line with the absolute file path. You should see something like this :

```
$ file '/home/mfacerias/Desktop/tps23750.pdf'
```

Then, execute the command, and look at the result.

In the same console, recall the last command and edit it to obtain something like this :

```
$ hexdump -C '/home/mfacerias/Desktop/tps23750.pdf' | less
```

Then, execute the command, and look at the result, at the left of the first line. End the command using `Q` key.

 See more : https://en.wikipedia.org/wiki/Magic_number_(programming)

2.2.3 Question : Analyse the result

 How could you explain the result of the last two commands ?

2.2.4 To do : More than one kind of magic

 Try the following commands, to understand more cases :

```
$ file /etc

$ file /etc/hosts

$ file /bin/ls
```

2.2.5 Question : Is there something wrong in the last "todo" ?

 Looking at the last "todo", you can imagine that a directory (here `/etc`) has magic numbers. Above in that course, it is said that a directory is a file, describing a list of the files it contains.

So try `hexdump -C /etc` to show his magic number, as we have done before for a *pdf*. What happened ?

Could you explain that result ?

2.2.6 Concept : Folder commands

 There are commands for **folder handling** :

— `pwd` : print working directory, to know which folder is the current directory ;
— `ls` : list, to show the folder's content ;
— `mkdir` : create directory, to create a folder ;
— `cd` : change directory, to move from the current folder to a target folder ;
— `rmdir` : remove directory, to delete an empty folder ;
— `tree` : show the folder tree.

Keep in mind that you can use `man` to get more information on these commands' syntax.

2.2.7 To do : Folder handling

 Open a text console and :

1. use `pwd` to know what is your current directory and remember it ;
2. use `ls` command to see all the files and folders in the current directory ;

3. then create a directory named `test` using the `mkdir` command ;
4. verify that the last command worked properly using the `ls` command ;
5. move in the new created directory using the `cd` command ;
6. verify that you are in the correct directory using the `pwd` again ;
7. here, create a directory named `subdir` ;
8. move to the directory you were first and verify your location with the `pwd` command ;
9. try the `tree` command, with the proper argument to only show the two folders you created. Try without any argument to see the difference ;
10. using `rmdir` and absolute paths twice, delete the two folders you created ;
11. verify that the job is done using `ls test`.

2.2.8 Question : `cd` shortcut

 Open a console and :

1. use `pwd` to show the current directory. What is its path ?
2. go to the file-system root using `cd`. Which argument must you give to do that ?
3. use the `cd` command without any argument. What is the current directory ?
4. use the `cd` command with the special path to go to your home directory. What is the current directory ?

Conclude on what is `cd` shortcut.

2.2.9 Concept : File commands

 There are commands for **file handling** :

— `cp` : copy, to copy a file ;
— `mv` : move, to move (or rename a file) ;
— `rm` : remove, to delete a file ;
— `cat` : concatenate, mainly used to dump the content of a file on the screen ;
— `less / more` : to to dump the file's content smartly (with the ability to navigate inside) ;
— `touch` : to create an empty file (or modify the attribute of a nonempty file).
— `vi / nano / emacs` : to edit the content of a text file.

2.2.10 To do : File handling

 Open a text console and :

1. use `touch` to create a file named `myfile` ;
2. verify that the last command worked properly using `ls` without any argument ;
3. verify that the last command worked properly using `ls` with the absolute path of `myfile` as argument ;
4. verify that the last command worked properly using `ls` with the relative path of `myfile` as argument ;
5. read the `cp` command's manual, then copy `myfile` in `/tmp` ;

6. verify that the last command worked properly using `ls` with `/tmp` as argument ;
7. copy `myfile` as `mysecondfile` in the same folder ;
8. verify that the last command worked properly using `ls` without any argument ;
9. read the `mv` command's manual, then rename `myfile` as `myfirstfile` in the same folder ;
10. verify that the last command worked properly using `ls` without any argument ;
11. move `myfirstfile` to `/tmp` ;
12. verify that the last command worked properly using `ls` with `/tmp` as argument ;
13. in one command, list all the files you placed in `/tmp` ;
14. in one command, delete all the files you placed in `/tmp` ;
15. verify that the last command worked properly using `ls` with `/tmp` as argument.

2.2.11 Question : Let's play cups and balls

 First, `pwd` answer `/home/toto/Desktop`.
Then I do `cp thefile /tmp`.
After that, I do `mv thefile newfile`.
Then, I do `cp /tmp/thefile /opt`.
So :

1. Where is the file `thefile` ?
2. Is there an other file with the same content ? Where ?

2.2.12 Concept : Finding files

 The `find` command allows to look for files. It can be used with a lot of criteria, and it is possible to do actions on the found files. Here is an example :

```
find ~ -name '*jpg' -size +1M -exec rm {} \;
```

In this example, we are looking for files, starting in ~ directory (user homedir), with a name ending with jpg (you may use ' ' to protect meta characters), and which are bigger than 1 MB (1024x1024 bytes).

We remove these files, {} is automatically replaced by the file path for each file found and `rm` is then done using it as an argument.

2.2.13 Question : How to invoke `find`

 First, you have to read the manual for the `find` command, and then, answer the following questions.

What is the correct command line to search in your homedir :

1. all pdf files ?
2. all files modified (changed) at least 2 days ago ?
3. all regular files ?
4. all directories ?
5. only hidden directories ?

2.2.14 To do : Final practical work

To practice, you will build the following tree structure in your homedir and play with the files in it. To be more efficient, you will do the job using less commands as possible (I did it in only 14 steps), using absolute or relative paths (it's as you want) but you must not use the `cd` command.

```
test
|---xyz
    |---programs
    |   |---java
    |   |   |---TP.java *
    |   |---langage_c
    |   |   |---a.c     *
    |   |---php
    |---public_html
        |---docs
        |---pictures
        |---index.html   *
```

1. create the tree structure and the empty files (indicated by *) ;
2. modify `index.html` to write `<h1>Hello world !!<\h1>` inside it ;
3. show `index.html` content ;
4. look at `index.html` rendering using *firefox* **from the command line** ;
5. copy all folders and files in `programs` in `pictures` folder ;
6. move all folders and files in `programs` in `docs` folder ;
7. show all the content of `docs` as a tree.