Introduction to Linux and CLI

Part 2

Francesco Ambrogi[1]

[1]**Assistant Professor**
Department of Mechanical and Materials Engineering
Queen's University

June 12, 2025

Table of Contents

How to follow this lecture

1. DO NOT take notes!
2. Open a terminal window and try to run the commands we will learn.
   - **Windows**: open a WSL terminal window.
   - **MacOS**: after starting the multipass instance, run **multipass shell openfoam**.
3. Download the u_velocity file from GitHub.
4. ASK QUESTIONS!

The move (MV) command

MV is very powerful command and it stands for "move". Think of it as your digital moving van, able to perform the following tasks:

- Rename files.
- Move files.
- Tidy up your directories.

The basic syntax of the MV command is:

mv [OPTIONS] source_file(s) destination_file(s)

- **source_file(s)**: name of the files we want to rename or move.
- **destination_file(s)**: name of the destination directory or new file name.

Example using MV

In HOME, let's create a couple of random .txt files (file1.txt, file2.txt) and a directory called "my_dir".

```
francesco@rafaela : (02_cli_part2) -> ls
file1.txt file2.txt my_dir
francesco@rafaela : (02_cli_part2) ->
```

1. **mv file1.txt 01_file1.txt**
2. **mv file2.txt 02_file2.txt**
3. **mv 0* my_dir**
4. **mv my_dir 01_my_dir**

Example using MV

In HOME, let's create a couple of random .txt files (file1.txt, file2.txt) and a directory called "my_dir".

```
francesco@rafaela : (02_cli_part2) -> ls
file1.txt file2.txt my_dir
francesco@rafaela : (02_cli_part2) ->
```

1. **mv file1.txt 01_file1.txt** - rename file 1
2. **mv file2.txt 02_file2.txt** - rename file 2
3. **mv 0* my_dir** - move EVERY file that begins with 0 into my_dir
4. **mv my_dir 01_my_dir** - rename the directory

WILDCARDS

Commands can use **wildcards** to perform actions on more than one file at a time, or to **find part of a phrase** in a text file.

- ***** (asterisk): this can represent any number of characters (including zero, in other words, zero or more characters). Example: **ls *.pdf** will list ALL files ending with **.pdf**
- **?** (question mark): this can represent any single character. Example: ls file?.txt will list **ONLY** file1.txt and file2.txt
- **[]** (square brackets): unlike the previous 2 which specify ANY character, [] lets you specify the range of characters. Example ls [fm]* will list ALL files beginning with f OR m.

The copy (CP) command

In Linux we use CP to copy files. The syntax, as usual, is:

cp [OPTIONS] source_file destination_file

A couple of important options:

- **-i** (interactive): this command will copy with a **warning** before overwriting the destination file.
- **-r or R** (recursive): this will copy the **directory structure** recursively.
- **-p** (preserve): this will copy while **preserving** file characteristics (creation date, access time, ownership, etc.)

Heavy duty copy (SCP and RSYNC)

The CP is good for copying small amount of data, but CFD simulations (as some will learn!) can generate TB of data in just a couple of simulations. Moreover, results might be **sensitive**. If that's the case, your options are:

- **scp**: the very important feature of the secure copy protocol **scp** is that all files and passwords that are being transferred are encrypted. **scp** is therefore highly recommended when dealing with sensitive or proprietary data.
- **rsync**: the remote sync or simply **rsync** is a command line, remote and local file synchronization tool. **rsync** is very fast and the most important feature is that it uses an algorithm that minimizes the amount of data copied by **only moving the portion of files that are different between the source directory and the destination directory**. Therefore, rsync is extremely useful for a periodic backup of data

The remove (RM) command

In Linux we use RM to remove files. The syntax, as usual, is:

rm [OPTIONS] file_name(s)

Linux DOES NOT have a recycle bin!!!

A couple of important options:

- **-i** (interactive): this command ensures that before anything is deleted it needs to be confirmed.
- **-f** (force): this will force the file(s) to be deleted. **You should generally AVOID this option, especially at the beginning.**
- **-r** (recursive): along with the current directory, this command will delete all subdirectories and files within it.
- **-v** (verbose): this will show on terminal what the command is currently doing.

The FIND command

It will come a day when you will have several thousands of files, especially those of you who will run heavy CFD simulations. The **find** command is a very powerful tool that will help you navigate and **search** what you are looking for. Its adaptability allows users to search by name, size, modification time, or content, providing a flexible and potent solution.

find [path] [options] [expression]

```
francesco@rafaela : (Teaching) -> find 07_MECH444/01_lectures/02_cli_part2 -type f -name "*.txt"
07_MECH444/01_lectures/02_cli_part2/my_file.txt
francesco@rafaela : (Teaching) ->
```

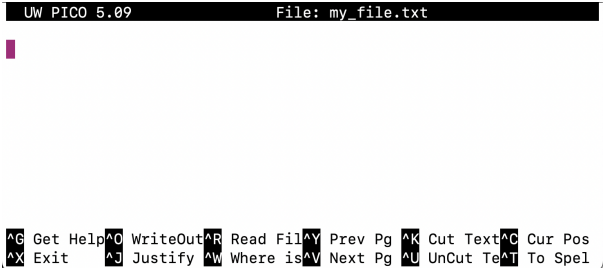
How to visualize text files in CLI

When using Linux, in many cases, you will not have a graphical user interface, e.g. when you are connected to a computer cluster or server. **Your very nice graphical text editors (VScode, Atom, Gedit, ...) won't work!**. Linux offers a selection of commands and tools to visualize and edit text files directly on the terminal window.

NANO text editor

GNU nano is an easy-to-use **command line text editor** for Unix and Linux operating systems. It includes all the basic functionality, like syntax highlighting, multiple buffers, search and replace with regular expression support, spellchecking, UTF-8 encoding, and more.

nano filename



Example using NANO

Let's open one of the .txt we created earlier, and let's add some text to it. The NANO commands you will use the most in this course are:

- **ctrl+O**: this will save whatever text you've written but leave the file open.
- **Directional arrows**: to move around the file just like a regular document.
- **ctrl+X**: to close the file after the modifications have been saved.

The CAT command

A quick way to preview the contents of a text file without having to open the file in a large application, is to use the **cat** command. The **cat** command on Linux **concatenates files together**.

```
francesco@rafaela : (02_cli_part2) -> ls
my_file.txt u_velocity
francesco@rafaela : (02_cli_part2) -> cat my_file.txt
Ciao, this is a test file.
Just showing how the cat command works.
Cheers
francesco@rafaela : (02_cli_part2) ->
```

The HEAD and TAIL commands

CAT is not the only way of efficiently visualizing text files. Sometimes we may only be interested in the BEGINNING of the text file, or in the very last lines. The **head** and **tail** command will help in this scenario. Let's test them on the **u_velocity** file you can download from GitHub.

- **head -n filename**: this command will show the first **n** lines of filename. n=10 by default.
- **tail -n filename**: this command will show the last **n** lines of filename. n=10 by default

Even better: the LESS command

The **less** command is a Linux terminal pager that shows a file's content one screen at a time. **It is useful when dealing with large text files** because it does not load the entire file but accesses it page by page, resulting in fast loading speeds.

less [OPTIONS] filename

Useful options of the **less** command are:

- **-E**: less automatically exits after reaching the end of the file.
- **-N**: less displays line numbers at the beginning of each line.
- **–help**: gives a complete list of options available.

The GREP command

The **grep** command in Linux is a powerful tool used for searching and manipulating text patterns within files. **grep** is widely used by programmers, system administrators, and users alike for its efficiency and versatility in handling text data.

grep [OPTIONS] pattern [files]

Useful options of the **grep** command are:

- **-i**: ignores, case for matching
- **-c**: this prints only a count of the lines that match a pattern.

```
francesco@rafaela : (02_cli_part2) -> grep -i ciao my_file.txt
Ciao, this is a test file.
francesco@rafaela : (02_cli_part2) ->
```

Basics of I/O standard streams

In Linux standard inputs and outputs (I/O) are distributed across 3 streams:

- Standard input - **stdin** numbered as 0
- Standard output - **stdout** numbered 1
- Standard error - **stderr** numbered 2

During standard interactions **between the user and the terminal**, standard input comes from the user's keyboard. Standard output and standard error are displayed on the user's terminal as text. **Stdin, stdout, and stdout** are referred to as the standard streams.

```
francesco@rafaela : (02_cli_part2) -> ls %
ls: %: No such file or directory
francesco@rafaela : (02_cli_part2) ->
```

Streams redirection

Linux includes commands to **redirect** each stream:

- $>$ - redirect standard output
- $<$ - redirect standard input
- $2>$ - redirect standard error

```
francesco@rafaela : (02_cli_part2) -> echo Ciao, Francesco!
Ciao, Francesco!
francesco@rafaela : (02_cli_part2) -> echo Ciao, Francesco! > greetings.txt
francesco@rafaela : (02_cli_part2) -> ls
greetings.txt my_file.txt   u_velocity
francesco@rafaela : (02_cli_part2) -> cat greetings.txt
Ciao, Francesco!
francesco@rafaela : (02_cli_part2) ->
```

Streams redirection - append

Linux includes commands to **redirect** each stream without **OVERWRITING**:

- >> - redirect standard output
- << - redirect standard input
- 2 >> - redirect standard error

```
francesco@rafaela : (02_cli_part2) -> ls
greetings.txt my_file.txt   u_velocity
francesco@rafaela : (02_cli_part2) -> cat greetings.txt
Ciao, Francesco!
francesco@rafaela : (02_cli_part2) -> echo Come stai? >> greetings.txt
francesco@rafaela : (02_cli_part2) -> cat greetings.txt
Ciao, Francesco!
Come stai?
francesco@rafaela : (02_cli_part2) ->
```

Pipes in Linux

Pipes are used in Linux to redirect a stream from one program to another. When a program's standard output is sent to another **through** a pipe, the first program's output will be used as input to the second, rather than being printed on terminal. The syntax for a pipe is the following:

command1 | command2

```
francesco@rafaela : (02_cli_part2) -> ls
greetings.txt my_file.txt   u_velocity
francesco@rafaela : (02_cli_part2) -> ls | less
```

What will be the output of this line???

The shebang

Essentially anything you type in the command line can be put in a **bash script**. A bash script is nothing but a recipe that tells the shell what command to execute and in what order.

Why? efficiency and productivity.

A few rules for beginners:

- Bash scripts ALWAYS begin with a shebang **#!/bin/bash**. Shebang is simply an absolute path to the bash interpreter.
- Bash scripts, by convention, have a **.sh** extension.

My first bash script

Using the commands we have learn so far, let's create our first bash script:

```
francesco@rafaela : (02_cli_part2) -> echo '#!/bin/bash' > my_script.sh
francesco@rafaela : (02_cli_part2) -> echo 'echo Ciao, Francesco!' >> my_script.sh
francesco@rafaela : (02_cli_part2) -> echo 'echo Come stai?' >> my_script.sh
francesco@rafaela : (02_cli_part2) -> cat my_script.sh
#!/bin/bash
echo Ciao, Francesco!
echo Come stai?
francesco@rafaela : (02_cli_part2) ->
```

To run a bash script simply type in the terminal:

./script_name

Oops ...


```
francesco@rafaela : (02_cli_part2) -> ./my_script.sh
-bash: ./my_script.sh: Permission denied
francesco@rafaela : (02_cli_part2) ->
```

Problem: the bash script we have just created IS NOT executable

The CHMOD command

In Linux the **chmod** command stands for **change mode** and is used to change the access mode of a file. Permissions have three categories: **read**, **write**, and **execute** simultaneously represented by "r", "w" and "x". The syntax to change the mode of a file is:

chmod [options] [mode] [file_name]

```
francesco@rafaela : (02_cli_part2) -> ls -l my_script.sh
-rw-r--r--  1 rafaela  staff  50 14 Jan 23:49 my_script.sh
francesco@rafaela : (02_cli_part2) -> chmod +x my_script.sh
francesco@rafaela : (02_cli_part2) -> ls -l my_script.sh
-rwxr-xr-x  1 rafaela  staff  50 14 Jan 23:49 my_script.sh
francesco@rafaela : (02_cli_part2) ->
```

Execute our first bash script


```
francesco@rafaela : (02_cli_part2) -> ls
greetings.txt my_file.txt   my_script.sh  u_velocity
francesco@rafaela : (02_cli_part2) -> ./my_script.sh
Ciao, Francesco!
Come stai?
francesco@rafaela : (02_cli_part2) ->
```