051176- Computational Techniques for Thermochemical Propulsion
Master of Science in Aeronautical Engineering

# Breaking-in the Linux shell

Federico Piscaglia*

## Foreword

This tutorial is about the basics of the Unix/Linux shell. To access a shell from your desktop environment, either:

- Click on the icon named *terminal* in some application menu
- Right-click on the Desktop and select *open in terminal*

Important things to know about the Linux shell:

- The Shell (like the OS as a whole) is **case sensitive**. Be sure to use the correct capitalization of words: `myname` ≠ `MYNAME` ≠ `MyName`. This applies to everything: commands, file names, variables, etc.
- **When naming files, use only letters (lo- and hi- case), numbers, dash, dot and underscore. AVOID any other character (e.g., accents) and spaces**.
- After you have started typing a command, press TAB to automatically complete it (if possible). If multiple alternatives do exist, a selection of choices will be proposed.

```
user@host ~$ ged<press TAB>
```

- Some commands accept *options* and *arguments*. [Command-line] options are introduced by '`-`' (single dash) or '`--`' (two dashes). Options must always precede arguments. Short options (one dash, one letter) can be combined:

  `ls -h -s -a` is equivalent to `ls -hsa`

- When specifying files or folders as arguments of a command, you can use *wildcards*: `?` means "every character"; `*` means "every combination of characters, of any length" (empty strings included).
- You can do copy-paste of strings (not files!) by using the mouse context menu (right-click). But a fastest way is the following:
  1. Anything that is selected with the mouse is automatically copied in the clipboard
  2. To paste it, just click with the center button (or the scroll wheel)

  This applies also for every text string within the Linux environment.

*ph. +39 02 2399 8620, e-mail: federico.piscsaglia@polimi.it

# 1 Show the content of a folder

## 1.1 Basic

Type "`ls`", this is the short for "list"

## 1.2 Long format

Type "`ls -l`" or "`ll`" (the latter may not work in some configurations). Short for 'long list'.

The output contains a lot of information on the file, as follows:

1. first character: whether is a plain file ("-"), a directory ("d"), a symbolic link ("l") or something else.
2. File permissions 'rwx-...' (see Sec. 3)
3. Number of links (ignore it)
4. Name of owner
5. Name of group
6. Size (in non-human format)
7. Date and time of last modification
8. Name of file (or directory)

To have file sizes in human-readable format (e.g. kB, MB, etc.) use `ls -lh`

### 1.3 Hidden files

Files (and folders), whose name begins with a dot, are treated as "hidden". They are not shown by "ls". To see hidden files, type:

```
ls -a
```

Usually hidden files and folders are used to store local configuration.

## 2 Work with folders

### 2.1 Navigate

- To change folder: type `cd <name of folder>` (use TAB completion!)
- To move up one level: type `cd ..` ("`..`" means "the upper directory")
- To do multiple "cd" at once:

```
cd dir1/subdir2/subsubdir4
```

- To go back to the previous directory (no matter on which level)

```
cd -
```

- To go directly to the home directory (usually `/home/<your username>`)

```
cd
```

- To check in which directory I am: `pwd`

### 2.2 Create

```
mkdir <name of new dir>
```

To create nested folders at once:

```
mkdir -p newdir/subdir1 ndir2/sdir3/ssdir4
```

### 2.3 Delete

```
rmdir dirToRemove
```

Note: you can remove a directory only if it is empty. To remove a directory and all contained files, see Sec. 4.2.

## 3 User and permissions

You can login into a Linux system only if you have an username. You can check what your username is by typing:

```
whoami
```

At the same time, each user belongs to one or more groups. You can check which group(s) you belong to by typing

```
groups
```

All operation of Reading/Writing/eXecution of a file are subject to permissions. Permissions are (in general) different between the *User* who owns the file, the *Group* and all *Other people*. Permissions can be checked by

```
ls -l <file>
```

$-\underbrace{rwx}_{\text{user}}\underbrace{rwx}_{\text{group}}\underbrace{rwx}_{\text{others}}$ `myFile.txt`

### 3.1 Changing ownership /group /permissions

Provided that you have sufficient permissions, you can change a file's owner, group or permissions.

To change a file's owner:

```
chown [-R] <newOwner>.<newGroup> <file(s)>
```

To change a file's group:

```
chgrp [-R] <newGroup> <file(s)>
```

To change a file's permissions:

```
chmod [-R] <code> <file(s)>
```

Permission are represented by a 3-digit code such that :

1. the first position means "user", the second one "group" and the third one "others"
2. The digit is "4" for reading, "2" for writing and "1" for executing. They must be summed.

Example:

- 777 means "everybody can read, write and execute (4+2+1)"
- 750 means "Members of group can read and execute (4+1), but only user can write (4+2+1). Others have no permission (0)"

- 400 means "only user can read (4), nobody can write or execute"

NOTE: the command line option [-R] means "apply to that folder and all files and subfolders it contains"

NOTE II: To navigate inside a directory, you must have *read and execute* permissions on that folder.

NOTE III: To change permissions of a file, you must be its owner (or you must be root).

### 3.2 The Superuser

The super-user (or root user) has always permissions to read, write and execute any file.

To become root type:

`su` ([open-]SuSE)

or

`sudo su` (Ubuntu and like)

To execute a command as you were root:

```
sudo <command>
```

## 4 Work with files

### 4.1 Create an empty file:

```
touch <file>
```

### 4.2 Remove a file

```
rm [-rf] <file(s)>
```

Option [-r]: delete a folder and all its content (files and subfolders). Use with caution.

Option [-f]: do not ask before deleting write-protected files. Do not issue error messages in case a file is not found.

### 4.3 Move a file

```
mv <originPath/file(s)> <path2/>
```

Renaming a file is the same as moving it:

```
mv <oldFile> <newFile>
```

To move a file from a different folder to the current one:

```
mv ../../fileToMove .
```

"." means "the folder I'm in now".

### 4.4 Copy a file

```
cp [-r] <path/file(s)> <destination>
```

The flag [-r] has the same meaning as above.

### 4.5 Read a text file

- Display the entire file on screen: `cat`
- Display the file and allow scrolling: `less`
- Display the first n lines of a file: `head [-n <nLines>]`
  Default is 10 lines.
- Display the last n lines of a file: `tail [-n <nLines>]`
  Default is 10 lines.
- Display the end of a file and reread each time is updated: `tailf`

### 4.6 Find a string in one or more files

```
grep [-nri] <string> <file(s)>
```

`-n` display also matching lines number

`-i` ignore case

`-r` descend into subdirs

EXAMPLE: FIND IN WHICH FILE IS THE STRING "PASSWORD" CONTAINED:

```
grep -ri password *
```

### 4.7 Find a file

Example: find all files with extension "pdf"

```
find . -name '*.pdf'
```

## 5 System variables

System variables are bound to a particular shell (terminal tab), and they persist as long as the shell remains open. To list all system variables:

```
env
```

To view the content of a variable:

```
echo $<variable>
```

### 5.1 The PATH variable

The PATH variable contains all folders where the executables are located. If the folder is listed in the PATH, its content can be executed from whatever location.

Try:

```
echo $PATH
```

NOTE: the current folder (".") is *never* included in the path. Thus, to execute a program in your same folder, you have to type:

```
./myProgram
```

To find where an executable is located:

```
~$ which gedit
/usr/bin/gedit
```

## 6 Manage processes

### 6.1 Launch a process

Process are launched by typing the name of their executable on the command line. When you launch a program from the command line, the shell remains busy until the program terminates. To execute further operations, you must open a new shell or terminate the program. Try:

```
gedit
```

If you want the shell to be freed after the beginning of the execution, you must launch your program in *background*: try:

```
gedit &
```

### 6.2 List all processes

You can see all process running on the system by the command `top`. To exit `top` just press "q".

### 6.3 Terminate a process

A process which is running in *foreground* can be terminated by pressing CTRL-C. A process running in background is terminated by

```
kill [-9] <PID> or
pkill [-9] <name of the process>
```

### 6.4 Switch foreground/background

- a process running in the foreground can be put in background by pressing CRTL+Z followed by the command `bg`.

- a process running in the background can be pulled to the fore by the command `jobs` followed by

```
fg <id of job>
```

### 6.5 Long running jobs

A process, even in background, is terminated as the shell which originated it closes. To avoid this, launch the program with `nohup`

```
nohup <program> &
```

## 7 Input & Output

### 7.1 Output redirection

A program's output is usually displayed on screen. In Linux, there are two "qualities" of output: `stdout` and `stderr`; as the name suggest, to the latter belong all error messages.

The output, which normally would be displayed on the screen, can be *redirected* to another destination: a file, a pipe or a virtual device. The operator for redirection is ">".

EXAMPLE:

```
ls > fileList
```

will write all folder content into a file named "fileList".

Error messages are not redirected by default.

```
ls *.exe > fileList
ls: cannot access *.exe: No such file or directory
```

To redirect error messages to the same file as the output the syntax is:

```
ls > fileList 2>&1
```

To save errors in a different file:

```
ls > fileList 2>log.err
```

The redirect operator ">" overwrites the file used for redirection. To *append* to a file use the operator ">>".

### 7.2 Pipelines

The output of a program can be the input of another program without need of being saved on disk. This is done with the operator "|" ("pipe").

EXAMPLE:

```
tail -n 100 log | grep Time
```

"Extract the last 100 lines of file `log` and search in them the string 'Time"'

## 8 File size and compression

### 8.1 Check file and folder size

To check how large a file is

```
ls -hs <file>
```

To check how large a folder is (considering all its content)

```
du -hs
```

To check the disk occupancy

```
df -h
```

### 8.2 Compressing files

To compress a single file

```
gzip <file>
```

A new file with the same name and extension `.gz` will be created. The old (uncompressed) file is deleted.

To create an archive

```
tar czf archiveName.tgz <file(s)>
```

All archived files will not be deleted from disk. Alternatively, one can use

```
tar cjf archiveName.tbz <file(s)>
```

(best compression, slower).

#### 8.2.1 Uncompressing files

To uncompress a single file:

```
gunzip <file>.gz
```

An uncompressed file is created and the old compressed one is deleted.

To extract an archive:

```
tar xzf archiveName.tgz
tar xjf archiveName.tbz
```

#### 8.2.2 Viewing compressed files

Some commands listed in Sec. 4.5 have their "compressed" counterpart: `zcat, zless, zgrep`. Vim can open compressed files without specifying any particular option.

## 9 Putting into practice

When launching simulations from command line, the correct practice is:

- Launch the process in background
- Avoid termination if shell is closed
- Save output to a file
- Save error messages as well

```
nohup myLongProcess > log 2>&1 &
```

If the log is going to be very large, you can write it compressed:

```
nohup myLongProcess | gzip > log.gz 2>&1 &
```

## 10 Hands on

```
cd
ls
ls -l
ls -a
mkdir linuxTutorial
cd linuxTutorial
echo Hello world!
echo A first text line in my file > myFile.txt
cat myFile.txt
echo A second text line in my file >> myFile.txt
cat myFile.txt
cp myFile.txt copyOfMyFile.txt
rm myFile.txt
mv copyOfMyFile.txt myFile.txt
mkdir aSecondDir
cd aSecondDir
cd ..
rmdir aSecondDir
mkdir -p aSecondDir/aThirdDir/aFourthDir
cp myFile.txt aSecondDir/aThirdDir/aFourthDir
ls aSecondDir/aThirdDir/aFourthDir
cd aSecondDir/aThirdDir/aFourthDir
pwd
cd ../../../
less myFile.txt
head myFile.txt
tail myFile.txt
tailf myFile.txt
grep second myFile.txt
grep -r first aSecondDir
find aSecondDir -name '*.txt'
rm -r aSecondDir
```

```
cd
rm -r linuxTutorial
```

```
xlogo &
jobs
ps -A | grep xlogo
fg
CTRL-Z
xlogo
CTRL-Z
bg
top
kill <PID>
kill -9 <PID>
which xlogo
```

## Acknowledgments

This document has been written with Dr. Andrea Montorfano, who is greatly acknowledged.