

# Linux for beginner

A Linux little helper  
to start with  
**SYMPHONIE**

**Sirocco, LEGOS, CNRS : 2021**

# Content

1. Introduction to LINUX Environment.....	3
1.1 The « root ».....	3
2. Opening a terminal.....	5
3. Some basic command principles in LINUX.....	5
3.1 General definition of a command:.....	5
4. The tree structure :.....	6
4.1 Where are we?.....	6
4.2 How to know what is in the directories :.....	6
4.3 How to move in the directories: relative path and absolute path.....	7
5. Practical use of the console shortcuts.....	11
5.1 The Completion.....	11
5.2 History, command recall and copy/paste with the mouse :.....	11
5.3 The jokers :.....	12
5.4 The Permissions :.....	13
5.6 Handling of files and directories:.....	14
5.7 Viewing and editing files :.....	15
5.8 Text Editor and Co.....	16
5.9 Finding a file: the find command.....	16
5.10 Search for a string : the « grep » command .....	16
5.11 The redirection « > » (and “>>”) and the "pipe" «   »:.....	17
5.12 File archiving and compression:.....	17
5.13 Need help learning more about a command and its options?.....	18
6. Beyond the current Linux command.....	19
6.1 suspend and resume an action :.....	19
6.2 Remote connection.....	19
6.3 See the running processes and how to stop a process:.....	19
7. Checklist of basic commands.....	20
8. Text editor in the terminal: VI.....	25
8.1 The modes of Vi.....	25
8.2 Basic commands.....	25
8.3 Editing commands.....	25
8.4 Find and Replace.....	26
Vim Commands Cheat Sheet.....	27
How to Exit.....	27
Editing a File.....	27
Inserting Text.....	27
Inserting a file.....	27
Deleting Text.....	27
Changing (or Replacing) Text.....	28
Substituting.....	28
Undo/Redo/Repeat.....	29
Screen movement commands.....	29
Marks.....	29
Searching.....	30
Selecting Text (Visual Mode).....	30
How to Suspend.....	31

# Linux to start with SYMPHONIE

The purpose of this document is to give some basic ideas about the Linux work environment no matter what distribution you are considering (Ubuntu, Debian, OpenSuse etc.). Without pretending to be exhaustive (far from it) this document should allow you to go from novice to advanced enough to understand the commands you will need when using a computational code like SYMPHONIE. For the exercises and examples, it is assumed that you already have the TP\_LINUX files on your account.

We will consider as solved the problems of installation of distribution and account. That is to say that you have a working Linux machine and you have a **login** and a **password**. **N.B. often under Linux it does not appear of character " \* " when one types the password, it is an additional safety which prevents to know how many letter your password is composed for an indiscreet person.**

Before getting into the practical aspects that concern us, it is interesting to know a little about the architecture of the file system and directories under Linux.

## 1. Introduction to LINUX Environment

Under Linux, file management is not like under Windows. The separation of the various disks, USB keys, CD drive, etc... is done according to a single tree structure. Where under Windows the files contained in different "disks", would be differentiated with respect to a disk identifier (C:\ or D:\ etc.), under Linux any file is visible from a "zero level" which is called the root.

### 1.1 The « root »

There is no folder higher than " / ", that means that there is no folder that contains the folder " / ". When you are at the root, you can't go backwards because... you are already at the very beginning. (Windows equivalence: even if it is not totally true, you can see / as equivalent to [C:\](#))

Unlike a Windows system, in the GNU/LINUX system the files that manage it are distributed in a tree where each of the files has a well defined place:

Path	Description
/	name of the root of the system tree
/bin	containing the executable necessary for the operation of the system
/dev	containing the special files corresponding to the device drivers
/usr/bin	containing the executable of the installed programs
/usr/sbin	containing commands available only to the super user (root)
/sbin	containing commands available only to the super user but which are essential to the functioning, especially during the boot, of the system
/boot	containing the kernel(s) and other files necessary to boot the system
/etc	containing the configuration files of most of the system programs, servers and sometimes the default configuration files of the users' programs.
/var	containing files used by different system programs, for example it contains logs, sockets etc.
/usr/lib	containing dynamic or static libraries available on the system. This is the default search directory for the linker.
/usr/lib/X11	containing files specific to the graphical server (Xorg)
/usr/include	containing headers files for C programming (standard C library, API of

third party libraries, headers usable for the Linux kernel like V4L (video for linux)).

/usr/local which can contain a kind of new tree with a bin, etc, lib for applications added after the installation of the system.

/mnt which can contain mount points of other storage devices (usb key, other hard disk, NFS, etc) today the tendency is rather to use /media (or /run/media).

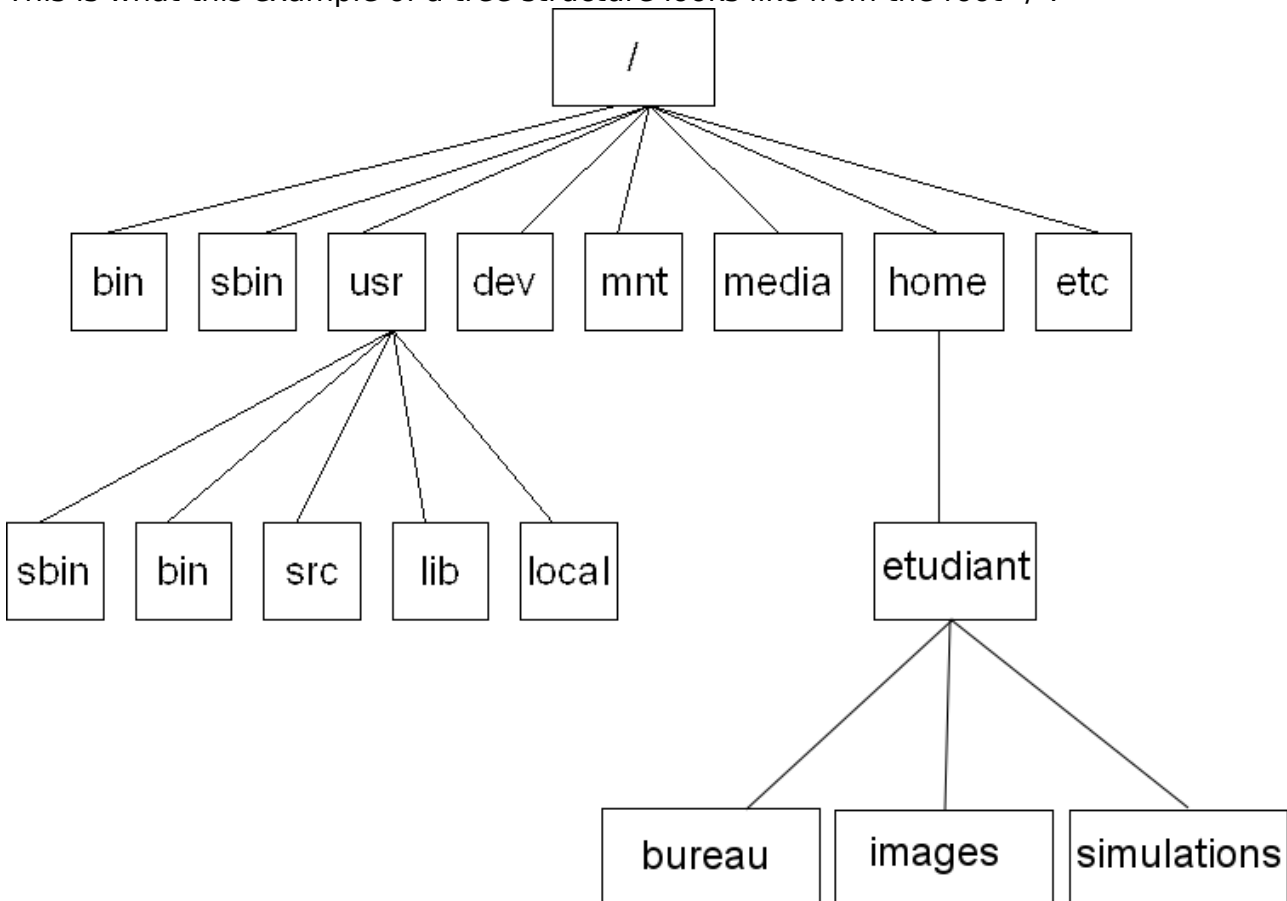
/home containing one directory per user in which all the user's files are stored.

For a user (for example "**etudiant**" in the rest of the document the login will always be **etudiant**), the most important directory will be **/home/etudiant**.

(Windows equivalence: you can give /home/student as My Documents or C:\Users\etudiant )

This is where all his files will be. Thus the user's files will be distinctly separated from the system files.

This is what this example of a tree structure looks like from the root "/":



The first steps will therefore be to take control of the working environment via the appropriate tools. With the recent Linux distributions, it is possible to have a graphical work environment (like Windows). This is often sufficient for "office" use of the system, such as writing a report, making pictures/images or simply going on the Internet. However, for a use like modeling, the terminal (or console) and the command line are much more efficient. This is even indispensable when it is necessary to connect remotely to a computing machine while the connection speed is low (which can make the use of any graphical interface very laborious and some time really painful).

## 2. Opening a terminal

The terminal is a program that opens a console in a simple graphical interface. This console allows you to launch bash commands that will be executed by the machine.

To open a terminal, there are several solutions:

- Go through the graphical interface: Menu → System → Terminal
- or directly from the keyboard:
- <Alt><F2> and type **konsole** then < enter >

That's it you have opened a terminal. By default, the terminal opens in the **/home/etudiant** directory which the user names "home".

(Windows equivalent: The terminal opens by default in My Documents)

The line that appears in the terminal looks like this  
**etudiant@machine:~>**

This line gives you, the user name (**etudiant**) the name of the machine on which you are connected (here **machine**), the symbol " ~ " represents the " **home** " of the user (we will often come back to it for explanations) and the character " > " invites you to enter a command. This last character can vary according to the installations. Then you have a white square, it is the command cursor or the prompt.

All you have to do is type a command and press "**enter**" to execute it. This is the next step.

## 3. Some basic command principles in LINUX

### 3.1 General definition of a command:

A command is either a program in its own right (that you would have coded for example), or integrated into the shell. To make a long story short, the shell is the computer language that will understand the commands (and therefore execute them) that you will use in the console.

The commands always use the same formatting, namely :  
**command [OPTIONS] [PARAMETERS]**

The options allow you to modify the behavior of the command, and the parameters can be the target, for example a file or a directory. What is in square brackets ([ ]) can be, or not be, specified. Depending on the desired behavior of the command they are not necessary. Often options include the "-" sign (minus sign), so the command interprets what is behind this sign as an option. We will see this again later.

Windows equivalent: the file explorer executes LINUX commands in the background. When you rename a file in the file explorer, it executes the "move" or mv command: mv myFile myRenamedFile . So you can do a lot of things without a graphical interface.

Most of the commands are counter-intuitive (at first sight) because they are often contractions of English words. For example to know the content of a directory the command is "**ls**", it is a contraction of the word "list". So "**ls**" will list the contents of

the current directory.

More explicitly, the command : **date** , will give you the date of your computer. It's easy!

*Exercise: try the **date** command... and there you have it, a command under Linux! So, isn't it great?*

## 4. The tree structure :

### how to find your way around, know the contents and move around:

To start seeing some sample commands, we'll kill three birds with one stone: knowing where you are, seeing the contents, and how to move through the tree.

File and folder paths in Linux use the slash "/" to separate folder names. The tree structure from the **root** for a file: "fichier.txt" which would be contained in the directory "lib" itself contained in the directory "usr" is therefore :

**/usr/lib/fichier.txt**

As a reminder, at the beginning we are on the user's directory, that is: **/home/etudiant**

This is what we call the "**home**" of the user.

#### 4.1 Where are we?

To find out the name of the directory in which you are, which is called the **current directory**, the command is: pwd (print working directory)

If we are in the home of the user "etudiant", pwd returns :

**/home/etudiant**

*Exercise : Display the name of your current directory : **pwd***

#### 4.2 How to know what is in the directories :

To know the content of the current directory, we use the command: **ls**

**ls** means "list". **ls** will return the list of directories and files contained in the current directory.

*Exercise: use the **ls** command to see the contents of the current directory.*

Some options of **ls**, for the options of **ls** you have to use the "-" sign :

- l displays a detailed list (with creation times, rights, size, etc)
- a shows ALL files, even "hidden files"
- h (h for human readable) converts the size (in bytes by default) in KB, MB, GB to make it more readable.
- R (capital letter, R for recursive) recursively gives what is in the sub-directories, be careful from the root "/" ls -R will show you all the content of the system... not a good idea...
- r (lowercase) gives the same result but "reversed" (r like "reverse")

*Exercise : Display the content of this directory with the l, r, h and t options separately or together: **ls -l**, then **ls -r** , then **ls -lht** etc...*

For an example of **[PARAMETERS]** on the ls command, we can use a directory name. If we do not specify, the command will be carried out on the current directory.

For example in the results of the exercises from the "**home**", there was the directory **TP\_LINUX**. It is then possible to see the contents of this directory with the command: **ls TP\_LINUX**.

But if you give the name of a different directory it also works as long as you respect the **path** to this directory in relation to the location of the **current directory**. The notion of path is very important in terms of understanding the tree structure in the console. We will explore this notion with the ls command and also the command that allows you to move in the tree.

So with the ls command we can find out what a directory contains, and possibly its sub-directories. There are a lot of options, some of them are about the size of the files, their creation dates or we can make the list by imposing that the result contains or excludes (-l, capital i) some strings.

*Example: **ls -l test.txt** will list all files and directories in the current directory except the file **test.txt**.*

### 4.3 How to move in the directories: relative path and absolute path

To move in the file system, we will use the command: **cd**

**cd** means "change directory" .

We can specify either a relative path or **PATH** (path in English) **relative** (in relation to the **current directory**) or **absolute** (in relation to the root " / ").

So an absolute path always begins with /.

For example :

**cd Rep1** moves us to the directory "Rep1" from the **current directory**, it is a **relative path**.

**cd /REP1** moves us directly to the directory "REP1" from the **root**, this is an **absolute path**.

The **[PARAMETERS]** used here are Rep1 or /REP1, they are information for the "cd" command. If we don't specify anything, the command : cd will send us directly to "**home**" (**/home/etudiant**). Other special "parameters" exist, here are 4 very important and/or useful ones:

- the "." "
- the ".. "
- the "- "
- the "~ "

The "." (the "dot") represents the current directory, so: "**cd** ."doesn't change anything, you stay in the same directory. But this designation is important for other purposes than the command "**cd**".

The ".."(the "dot" "dot") is the "parent" directory (that is, the one that contains the current directory). For the "home" of the user "**etudiant**", that is **/home/etudiant**, the parent directory is **/home**. So to go back one step from the current directory the command would be: "**cd ..**" simply.

For two steps backwards it will be: **cd ../../** and so on.

This parameter is fundamental to understand how to move efficiently in the tree structure. You must not neglect it!

The "-" (the "minus sign") is the directory where you were before the current directory. If for example you were in the directory : /home/rep1, and you had then changed directory with, for example, the command : "**cd /home/rep2**" (that is to say to pass from /home/rep1 to /home/rep2 with an **ABSOLUTE path**). Then to go back to /home/rep1 you just have to use the command : **cd -** .

And if you ever wanted to go to /home/rep2 again, you just have to use the same command : **cd -** . Because this time the old directory would become /home/rep2 .

Finally, the "~" (pronounced tilded), this parameter corresponds to the "home" of the user. In our case the "home" is /home/etudiant so the commands :

"**cd ~**" or "**cd /home/etudiant**" will do the same thing as "**cd**".

The ~ doesn't seem to be useful when presented like this but if your "home" is: **/path/to/my/new/home/login\_name** and you want to go to the **rep1/toto/** folder of your "home" from the directory: **/truc/machin/bidule/tata** (or anywhere else in the tree), you will only have to use the command

**cd ~/rep1/toto** , which is quite easy, isn't it ?

And it's shorter than **cd /path/to/my/new/home/login\_name/rep1/toto** ...especially if you have to type it...

So you can move from several directories in the tree. To move **from** /home/rep1 **to** /home/rep1/rep2/rep3 you can do directly :

**cd rep2/rep3** (in **relative path**) or **cd /home/rep1/rep2/rep3** (in **absolute path**).

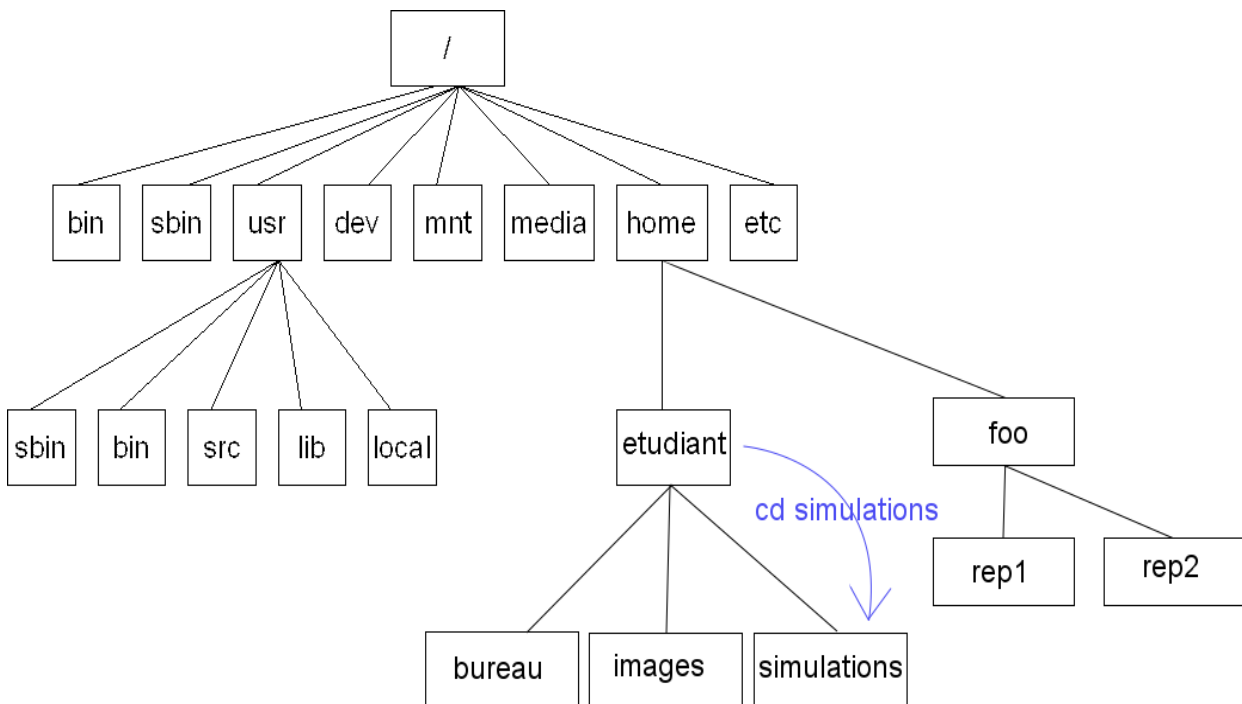
In the same way, if we are in the /home/rep1/rep2/rep3 directory and we want to go to the /home/toto1 directory, we can use **cd ../../../../toto1** (in **relative path**), etc.



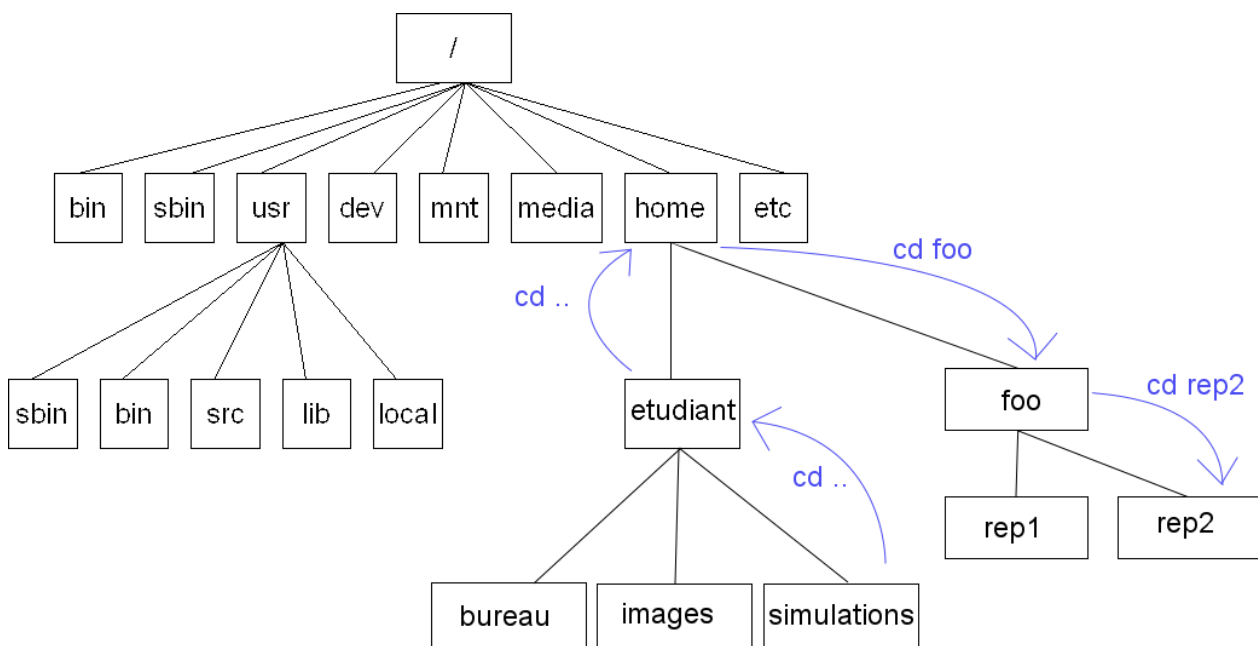
Visually, what does it look like?

If we are in the "home": /home/etudiant .

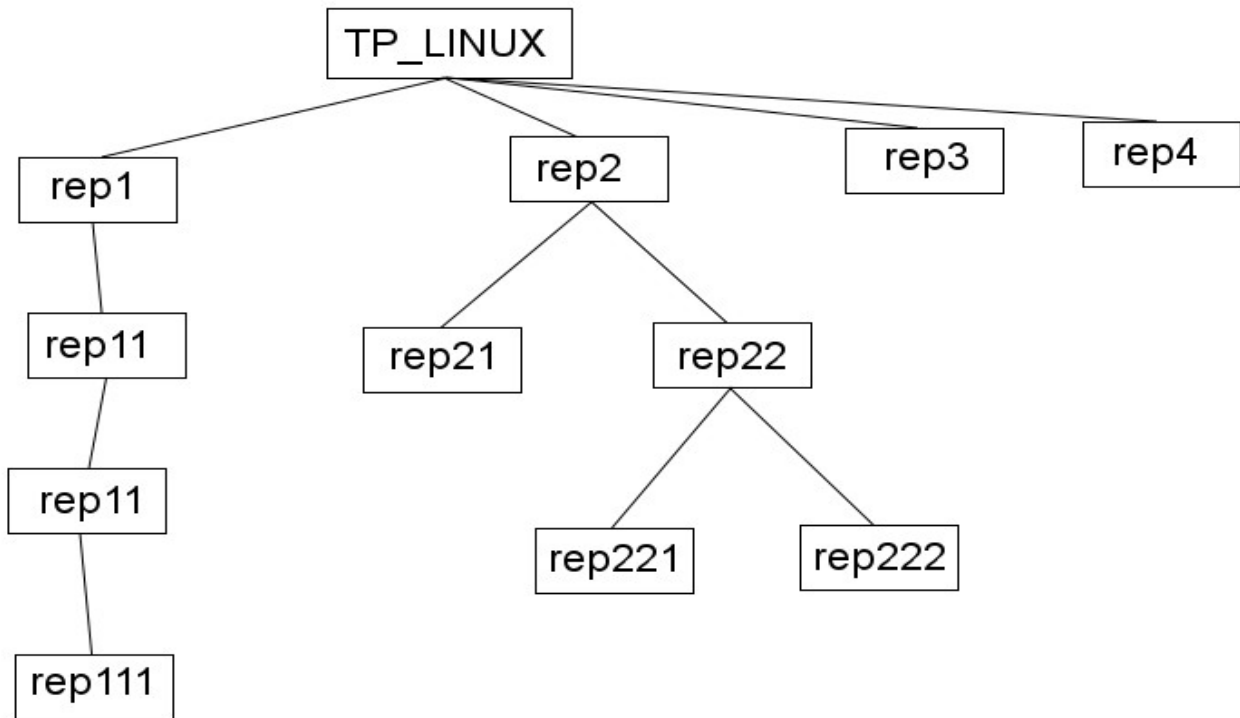
The command: **cd simulations** makes us go to /home/etudiant/simulations :



from this directory (/home/etudiant/simulations) the command: **cd ../../foo/rep2** sends us to the directory /home/foo/rep2 :



Exercise : Use the command : **cd ~/TP\_LINUX** . From this directory and with the help of the tree diagram of this one (given below), test the result of the **cd** command in relative path to go to the directory **rep21**. Then try to go directly from this directory to the directory **rep111**. Use the **pwd** command to see if you are where you think you are and use the **ls** command to see what the directories contain. Then practice moving around the TP\_LINUX directory tree as you wish. Finally use the command "**cd**" alone to return to the "home".



### To summarize this first part:

- Under linux, everything is organized in the form of files and directories. There is no disk like in windows like C: or D: etc...
- All folders are accessible from a directory called the root: /
- The starting folder of the user (for example **etudiant** login), where she/he will store all her/his files is the home: **/home/etudiant**
- the command to find out the name of the folder where you are is : **pwd** . This folder is called the **current directory**.
- The position of the directories in the tree is what we call the path or the **PATH** . A path is either **relative** (to the **current directory**) or **absolute** (from the root)
- The **ls** command is used to list the contents of directories
- **cd** is used to move from directory to directory.

## 5. Practical use of the console shortcuts

### 5.1 The Completion

The shell (the command interpreter) offers an interesting feature: the completion which automatically completes what you type, by using the **"TAB"** key (**tabulation key**)

-If the machine "beeps", it means that it is impossible (nothing matches)

-If it does not beep, but nothing happens, it is hesitating. Press **TAB** again so that it lists the set of possibilities, and complete a little more...

-If it finds without ambiguity, it completes, whether it is the name of a directory, a file or an executable...

*Exercise: go to the TP\_linux directory using the command: **cd ~/TP\_linux***

*Then type: **cd r** and press **"TAB"** once. The machine completes "r" and gives **rep**. Press **"TAB"** again, the machine then finds several possibilities, here **rep1/**, **rep2/**, **rep3/** and **rep4/**. Type for example **2** and then **"TAB"** twice again, the machine then gives two new sub-directories **rep21/** and **rep22/** as possibilities.*

### 5.2 History, command recall and copy/paste with the mouse :

#### -History


In computer science you have to know how to be lazy, as we often carry out the same type of command over and over again, it is very practical to know how to find a command that we typed five minutes ago (or even five seconds ago) and if possible without having to do it again ourselves. A computer rarely forgets what it has done, unlike us poor organic brains...

The **history** command will list all the commands you have already typed in the console.

Exercise : try this command in the console.

#### - Command recall

Knowing what you have already typed as a command is already good, it helps to know what you have done (if you have not made a mistake for example) or how you have done it. But what is even better is to be able to do it again very quickly. By using the

**"up arrow" key**  it allows you to go back in the history of your commands and once you find the one you want to execute it immediately by pressing "enter".

Of course the **"down arrow" key**  allows you to go back to a more recent command.

*Exercise: test this feature to find the last "cd" command you executed.*

Sometimes it still doesn't go fast enough. For example, if you are looking for an old command line typed several hours ago. It is possible to find it in several ways.

If you know how it starts: for example **"ls"**, you type **ls** in the console **without pressing "enter"**, then you use the **"Page Up" key** as you would use the "Up Arrow" key, i.e. "Page Up" will take you up the history of commands that start with **"ls"** and **"Page Down" key** will go the other way.

*Exercise: use the "page Up" key to search for the last command beginning with "ls".*

More advanced keyword search, **Ctrl+R** (Capital R): find a command typed with a few letters

In case the "up arrow" and the **history** command are not enough to find an old command you have typed, there is a very useful shortcut: **Ctrl +R**. So press the Ctrl and R keys at the same time and the computer will go into "search for a typed command" mode ("R" for Search).

Here you can type any string that matches an old command. For example, press Ctrl + R and then type "all". Linux finds the command `ls --all` which contained the word "all". You just have to type "Enter" to run the command again!

(reverse-i-search)`all': `ls --all`

If this is not the command you were looking for, press Ctrl + R again to move up the list of commands containing "all".

It may look silly on a command like this, but some of them are really long and it's a real pleasure not to have to rewrite them all!

*Exercise: test this feature with the string of your choice (it must have been in a previous command).*

#### **- copy/paste with mouse**

It is often possible to save a lot of time by copying and pasting with the mouse. To do this under Linux, simply select the text you want to copy by highlighting it with the left mouse button. This action will put the selected content in memory. Then you just have to click with the mouse wheel (on laptop pad using right and left click at the same will do the same) to paste this selection at the place where you put your cursor. In the case of the console you can only write at the command line level. So no matter where you put your mouse in the console window the "paste" will always be done at the prompt.

*Exercise: test the copy/paste with the mouse on the "ls" command from the previous exercise.*

With the `ls` command, you can see lists of objects (files and directories), regardless of their nature. By using special characters, which are called jokers, it is possible to make particular lists according to our needs.

### **5.3 The jokers :**

A Joker is a special character that allows you to replace strings:

- \* Replaces an entire string, any string (even no string !)
- ? Replaces a single character, any character but **one!**

For example, the command `ls test*` will list all files starting with "test":  
`test1 test2 test10 testa2 etc....`

while the command `ls test?` will list only the files of 5 characters beginning with test:  
`test1 test2 test3`

*Exercise: We will test the **ls** commands and the use of jokers.  
Using the `ls` command list the files :*

Using the joker \* :

- list the files starting only with "Prog".
- then those beginning with "test"



Or to remove the write and execute rights to the group and others.

**chmod go-wx file**

Now that you have seen how to change directories and list the various objects contained in them, we will see how to manipulate these objects.

### 5.5 Create and delete a directory:

The command: **mkdir reptest** will create a directory "**reptest**" in the current directory (mkdir for make directory). To create a directory at an address other than the current directory, you have to put the path, either in relative or absolute.

For example, **mkdir /home/etudiant/TP\_LINUX/reptest** will create a directory **resptest** in the directory **/home/etudiant/TP\_LINUX** (if it exists). This is an absolute path creation. In relative path it can be simply **mkdir ../reptest** that will create a directory **resptest** in the previous directory in the tree.

To delete a directory, the command **rmdir reptest** (remove directory) will delete the directory **reptest**, **if it is empty**. We will see later another command to delete a directory and all its sub-directories with the **rm** command.

**WARNING: unlike Windows which has a Trash can, by default when you delete a directory or a file with the rmdir or rm commands it is permanent!**

### 5.6 Handling of files and directories:

#### - Copy :

**cp** (copy) allows you to copy files (or directories).

For example,

- copy **FILE1** to **FILE2** the command will be :  
**cp FILE1 FILE2**
- copy **FILE1** AND **FILE 2** into the directory **DIRECTORY** the command will be:  
**cp FILE1 FILE2 DIRECTORY/**
- **cp -r** (or **cp -R**) allows you to copy a directory recursively: **cp -r rep1 rep5** the entire **rep1** directory will be copied into the **rep5** directory. If the directory already existed it will contain the contents of the **rep1** directory in addition to what it already contains. **Be careful if two files have the same name already in rep1 and rep5. The one in rep5 will be overwritten by the "new" one from rep1.**

#### - Rename or move

**mv** (move) allows to rename a file or a directory.

- **mv FILE1 FILE2**  
FILE1 will then be renamed FILE2. This also works for directories. If FILE1 was a directory then FILE2 will also be a directory with everything that was in FILE1...etc.....
- **mv FILE1 [FILE2]... DIRECTORY/**  
File1 (and other file FILE2 etc.) is moved to the directory : DIRECTORY .

#### - Delete : **WARNING THIS IS PERMANENT!!!**

**rm** (remove) allows you to delete files (or directories)

- **rm [OPTIONS] FILE1 [FILE2]...**

FILEn can be directories but you have to use the « **-r** » option like for cp. **WARNING**

## THIS IS PERMANENT!!!

*Exercise : From the ~/TP\_LINUX directory*

*With the command **cp** :*

*copy in rep1 the file testa1. Then check your action with the **ls** command if the file testa1 exists in the rep1 directory.*

*With the **rm** command*

*remove the file testa1 from the directory rep1*

*Check your action with the **ls** command if the file testa1 has been removed from the rep1 directory*

*copy all the files beginning with " test " in the directory rep4 with the command **ls** and the joker \**

*Check your action with the **ls** command by listing the contents of the rep4 directory.*

*With the **cd** and **mv** command*

*From the **TP\_LINUX** directory, go to the rep2/rep22/rep221/ directory*

*Then from **this directory** move the file "rep\_test" to the directory rep3 which is in the initial **TP\_LINUX** directory.*

*Check your action with the **ls** command by listing the contents of the rep3 directory and also check that the rep\_test file no longer exists in the **TP\_linux/rep2/rep22/rep221/** directory.*

### 5.7 Viewing and editing files :

There are several commands that allow you to edit files or simply view them. Here are several commands that are very useful.

cat file	displays the content of the file on the screen in ASCII
more file	displays a file progressively on the screen: Enter = go down one line, Space = go down one page, q = quit
less file	like more, but you can use the Back Page key. Not available on all systems.
head file	displays by default the first 10 lines of a file
head -n{N} file	displays the first N lines of a file
tail file	displays by default the last 10 lines of a file
tail -n{N} file	displays the last N lines of a file

*Exercise: From the **TP\_LINUX** directory, with the **cat** command, display the contents of the **test.txt** file.*

*Compare with the use of the commands **more**, **less**, **head** and **tail**. For better readability, use the **clear** command between each test*

The rest of the document is not necessary for primary use of the console in the case of modeling with SYMPHONY. Of course you will have seen only a minor part of the commands, but it is still very sufficient. In the following we present several other commands that you will probably need if you go further in the modeling. This is a new language with its syntax and vocabulary but don't forget that the commands always work on the same principle. So as the author would say: DON'T PANIC!

## 5.8 Text Editor and Co

In a terminal, the two most popular editors are vim (or vi) and emacs. Both are equally valid. They require a little investment to get used to them, but from the point of view of remote file editing and without a graphical editor they are of enormous interest.

vi file            edit a file with vim text editor . All linux system has vim installed

emacs file        edit a file with emacs text editor . All linux system hasn't emacs installed

A vi help is available at the end of this document.

More accessible and advisable for beginners, there are several editors that are more similar to WORD, like kate, kedit, gedit, gvim, etc. They are not always installed by default but it is very easy to get them.

## 5.9 Finding a file: the find command

It is sometimes laborious to search for a file in the tree of the disk on which you are working. For that there is the **find** command.

**find [path] [criteria]**

The most common options for **find** are :

- **-name file\_name**                    : The name of the file you are looking for
- **-user user\_name**                    : The owner of the file is name
- **-group group\_name**                 : Same thing with group
- **-size [+ -]file\_size[cbkMG]**     : Selects if the size matches (use + or - for larger than or smaller than, and c b k M G to indicate the unit of measurement of size...)
- **-atime number** , **-mtime number** , **-ctime number** : The file has been **accessed, modified** or **created** less than « number » days ago.

To use several criteria: (**expr -or expr**) (**expr -and expr**)

Example the command :

**find . -name file.txt -and -size +10k :**

will search from the **current directory** « . » and its **sub-directories**, all files that have the name "file.txt" and a size greater than 10ko.

*Exercise: use the command: **find . -name fichier.txt** then, try : **find . -name fichier.txt -and -size +10k**, what is the difference between the two results? Confirm that the **fichier.txt** file in the rep2/rep21 directory is less than 10kb while the one in rep3 is more than 10kb by using the **ls** command with the necessary option(s).*

## 5.10 Search for a string : the « grep » command

**grep string file** searches the file for the string "string"

Thus used the command does not reveal its full potential. By doing a search for the string on a list of files (using the joker "\*" for example), it allows to find all the corresponding strings in all the files of the list.

*For example: do the command **grep Jedi test.txt**,*



then the command **grep Jedi test\***, this last one writes to the screen all the lines of the files of the list **test\*** containing the character string "Jedi".  
The result gives the name of the corresponding file as well as the string you have highlighted.

### 5.11 The redirection « > » (and ">>") and the "pipe" « | »:

The commands we've seen so far return text, either file lists or information. These lines of text can be further processed.

The simplest way is the redirection using the « > » character.

For example, the command **ls test\*** returns a list of files or directories in the current directory beginning with "test". If you do the command: **ls test\* > toto1** you create a text file **toto1** which contains the result of the command. Be careful, if the file **toto1** already exists, its content will be overwritten by the result of the command.

*Exercise : do this command : **ls test\* > toto1** then using the command **more toto1** check the content of **toto1**.*

You can also redirect the result of a command to **add it at the end** of an already existing file using « >> ». By doing the command: **ls test\* >> toto1**, **toto1** will contain twice the result of the command **ls test\*** (the previous one from the exercise and the new one from the second command).

Another type of redirection is the so-called "pipe" or the "|" character ("|" is obtained on a french PC keyboard by pressing the keys: <Alt Gr> <6>, on english PC keyboard it is <Shift> « left to enter »). This command is used to link several commands together. Schematically, it acts like this: **command1 | command2** i.e. **command2** will be executed on the **result of command1**. There are multiple uses of this feature. For a simple example, **ls test\* | grep 1**, this command will search for the string "1" in the list of files that start with "test".

Another example as an exercise, use the command **ls -R ~/\***. The result is huge and it scrolls on the screen without you having time to see what is going on. If you use the command: **ls -R ~/\* | more** you have time to see the information on the screen "**page by page**". To exit this mode, simply press "q".

This addition of the "**| more**" command can be very useful when running a program. It allows you to follow the progress of the program in a debugging session.

### 5.12 File archiving and compression:

Without going into detail, you can always come back to it later according to your needs, the **tar** and **gzip** commands allow you to archive files and trees (with tar) and to compress them (with gzip).

This is useful to save memory space, to archive directories or to send a large number of files in a single archive.

Examples for **tar** :

<b>tar -xvf archive.tar</b>	extracts the files from the archive archive.tar, displaying the names of the files in it
<b>tar -xvzf archive.tar.gz</b>	extract the files from the archive using gzip and then tar
<b>tar -jxvf archive.tar.bz2</b>	extracts the files from the archive using bzip2

tar -cvf archive.tar file1 [file2...]	then tar creates a file "archive.tar" containing file1, file2...
tar -cvzf archive.tar.gz my_folder	creates a gzip file containing all the content of the folder 'my_folder'
tar -tvf archive.tar	Allows to see the content of an archive without extracting it

Example for **gzip** :

gzip file.txt	creates the compressed file file.txt.gz from file.txt
gunzip file.txt.gz	extracts the file file.txt from file.txt.gz

### 5.13 Need help learning more about a command and its options?

Most commands have a manual page that gives a description of their uses in varying degrees of detail, sometimes useful, sometimes obscure. Nevertheless it is usually useful, but you have to know how to sort out the information. You will often only need a small part of the information that is provided.

This command is: **man** (like manual).

For example, to know the options of the tar command, just type :

**man tar**

Always try to use the internet when you can... google will find examples for the different commands. You should never neglect this possibility. We use it all the time even when we are used to Linux.

## 6. Beyond the current Linux command

Several actions are possible under the linux environment to allow you to better use the machine.

### 6.1 suspend and resume an action :

For example, you are in a vi editor. You want to check a filename under the directory and return to the editor. To do this you can use <Ctrl><z> to suspend the activity of the vi editor. Then to return use the **fg** (for foreground, **bg** will put your vi session in background but it is not very usefull...) command is used to make the suspended program (in the example vi) active again.

### 6.2 Remote connection

You will have to connect to computers to do modeling in the future. For that you will have to connect to a remote machine. For that we use the ssh command like for example,

**ssh your\_login@host\_ip\_adress**

to have a graphical server, you have to add **-X** (X in capital letter) after the ssh, like this

**ssh -X your\_login@host\_ip\_adress**

Obviously, to be able to do this you will need to get a **login** and **password** on the machine in question.

Very often, at the beginning, the administrators of these machines will give you a temporary **password** that you will have to change quickly for more security. To change the password (on a remote machine or not) you have to log in first, then use the command: **passwd**

**For the purposes of this tutorial, please do not do anything of the sort. But the information will probably be useful to you in the future.**

It is also possible to copy files from a remote machine to a local machine and vice versa (depending on the rights you have as a user of course).

There are several commands, one of the most common is **scp**. It works like the **cp** command. Either : **scp source destination**

For example,

**scp your\_login@host\_ip\_adress:/home/your\_login/file1.txt file2.txt**

this command will copy from the remote machine " host\_ip\_adress " the file " file1.txt " which is in the directory " /home/your\_login/ " and copy it in the **current directory** under the name **file2.txt**

### 6.3 See the running processes and how to stop a process:

The **top** command gives the running processes with user names, load in % of CPU and memory. But also a process number designated by the **PID**. This PID is the identification of the process. To quit the top command just press « q ».

To stop a process we will use the kill process\_number command. Be careful to stop the process you want...

## 7. Checklist of basic commands.

### System commands :

ls list files and folders

- l : detailed list
- a : show all hidden files and folders
- F : indicate the type of item
- t : sort by last modification date

Count the number of files in a directory : ls (display) and count the number of lines (wc -l) : `ls -la | wc -l`

`ls -al | wc -l`

cd "path" : change folder  
relative : `cd ../../user`  
absolute : `cd /usr/games`

du : size occupied by the files  
-h : the size for humans  
-a : show the size of folders AND files  
-s : get just the grand total

df -h : know the used space

cat : show all the file  
less : display the file page by page

head : display the beginning of the file  
tail : display the end of the file

touch : create a new empty file  
mkdir : create a new folder

cp : copy a file  
ex : `cp *.jpg myfolder/`

copy a file from one linux to another linux with ssh:  
`scp -r /home/bidule.rpm 8.8.8.8:/home`

mv : move a file  
ex : `mv *.jpg myfolder/`

Rename a file :  
`mv filebidon superfile`

rm : delete a file  
-i : ask for confirmation  
-f : force the deletion, whatever happens  
-v : tell me what you are doing, you little sneak  
-r : delete a folder and its content

ln : create links between files (shortcuts)  
-s : symbolique link

## Users and rights :

sudo "command": become root for a moment  
sudo su : become root and stay root

passwd : change the password

usermod : modify a user

-l : rename the user (the name of his home directory will not be changed though)  
-g : change group

chmod : change access rights

+ means "add the right".

- means "remove the right".

= means "assign the right".

u = user (owner)

g = group

o = other

ex: chmod -g+w report.txt

-R to assign recursively

ex: chmod -R -g+w my\_folder/

## Search for files :

history : Displays the command history  
-c : delete command history

locate : a quick search (not installed by default)

ex: locate notes.txt

=> /home/mateo21/notes.txt

!/ the flaw of locate I wanted to tell you about: the command does not search your whole hard disk but a database of your files. Once a day, your system will update the database. Force indexing in the database : updatedb

find : search the files currently present on the whole hard disk

find "where" "what" "what to do with" (where : it's the name of the folder in which the command is going to search // what : it's the file to search // what to do with : it's possible to do actions automatically on each of the found files)

(only the "what" parameter is mandatory)

Search by name

find -name "logo.png"

ex: find /var/log/ -name "syslog"

Search by size

find -size +10M

Search by date of last access

find -name "\*.odt" -atime -7

-type d : to search only directories.

-type f : to search only files.

"what to do with":

- print : display
- delete : delete
- exec : execute a command

### Extract, sort and filter data :

grep : filter data (Its role is to search for a word in a file and to display the lines in which this word was found.)

- i : not to take into account the case (upper / lower case)
- n : know the line numbers
- v : reverse the search : ignore a word
- r : search in all files and subfolders (ex: grep -r "Site du Zéro" code/)

Use grep with regular expressions:

grep -E [Aa]lias .bashrc ... returns all lines that contain "alias" or "Alias".

grep -E [0-4] .bashrc ... returns all lines that contain a number between 0 and 4.

grep -E [a-zA-Z] .bashrc ... returns all lines that contain an alphabetic character between a and z or between A and Z.

sort : sort the lines

- o : write the result in a file (ex: sort -o names\_tries.txt names.txt)
- r : sort in reverse order
- R : sort randomly
- n : sort numbers

wc : count the number of lines/words/bytes

- l : count the number of lines
- w : count the number of words
- c : count the number of bytes
- m : count the number of characters

uniq : delete duplicates

- c : count the number of occurrences
- d : display only duplicate lines

cut : cut a part of the file (ex: cut -c 2-5 names.txt => keep only the characters 2 to 5 of each line of the file)

ex: cut -c 3- names.txt => from 3rd to last

Cut according to a delimiter

- d : indicates the delimiter in the file
  - f : indicates the number of the field(s) to cut
- ex : cut -d , -f 1,3 notes.csv

### Redirection flows:

- > : redirect to a new file
- >> : redirect to the end of a file
- 2> : redirect errors in a new file
- 2>> : redirect errors to the end of a file
- 2>&1 : "send errors to the same place as the rest"
- < : read from a file
- << : read from keyboard progressively

| : chain commands : "Chain commands" ? This means connecting the output of one command to the input of another command.

### **Monitor system activity:**

who : The list of connected processes

ps : list of static processes

-u USER : list processes started by a user

-ef : list all processes of all connected users

-ejH : display processes in tree

top : list dynamic processes

Ctrl + C : stop a process launched in console

kill : kill a process (you have to get the PID of the process(es) you want to kill. For this, 2 solutions : ps / top)

-9 : stop the process

reboot : restart the computer (may not work depending on your administration rights)

### **Archiving and compressing :**

tar -cvf : create a tar archive

-c : means "create" a tar archive.

-v : means to display the details of the operations.

-f : means to assemble the archive into a file.

tar -tf : display the content of the archive without extracting it

tar -rvf : add a file

tar -jxvf (for bzip2)/ -xzf : extract files from the archive (extracted in the directory you are in)

gzip & bzip2 : compress an archive

\* gzip : it is the most known and the most used.

\* bzip2 : it is a little less frequently used. It compresses better but more slowly than gzip

=> compresses then modifies the name:

.tar.gz : if the archive has been compressed with gzip.

.tar.bz2 : if the archive has been compressed with bzip2.

gunzip & bunzip2 : decompress an archive

\* tar -zcvf : archive and compress with gzip

\* tar -jcvf : archive and compress with bzip2

zcat : equivalent of cat, able to read a compressed file (gzipped).

zmore : equivalent of more, able to read a compressed file (gzipped).

Zless : equivalent to less, able to read a compressed (gzipped) file.

unzip -l : unzip a .zip file

unrar e : unzip a .rar (install the unrar package)

### **Remote connection :**

install openssh-server (on debian/ubuntu):  
sudo apt-get install openssh-server

se onnecting :  
ssh login@ip

### **Transferring files:**

wget + URL

scp : copy files over the network

ftp server connection:

ftp ftp.debian.org (/or sftp)

put : send a file to the server.

get : download a file from the server

rsync : compares and analyzes the differences between 2 files and copies only the changes

-a : keeps all the information about the files, like the rights (chmod), the modification date, etc.

-r: also saves all subfolders in the folder to be saved.

-v: verbose mode, displays detailed information about the current copy.



## 8. Text editor in the terminal: VI

Vi is one of the most popular text editors under Linux (with Emacs and pico) despite its very limited ergonomics. Indeed, Vi is an editor entirely in text mode, which means that each action is done with text commands. This editor, although not very practical at first sight, is very powerful and can be very useful in case the graphical interface is not working.

The syntax to launch Vi is the following:

```
vi file_name
```

Once the file is open, you can move around using the arrow keys, as well as the h, j, k and l keys (in case the keyboard does not have arrow keys).

### 8.1 The modes of Vi

Vi has 3 modes of operation:

- **Normal mode:** the one you are in when you open the file. It allows you to type commands
- **Insert mode:** This mode allows you to insert the characters you type into the document. To switch to insert mode, simply press the Insert key on your keyboard, or alternatively the i key.
- **Replace mode:** This mode allows you to replace the existing text with the text you enter. You just have to press insert (or i) again to switch from insertion mode to replacement mode, and press the Esc key to return to normal mode.

### 8.2 Basic commands

Commande	Description
:q	Quit (without save)
:q!	Force vi to quit without saves (discard all the changes that have been made to the document))
:wq	Save and quit
:w <i>myfile</i>	Save in new name file « myfile »

The "!" is used to force the editor to do the command. It is applicable to :w

### 8.3 Editing commands

Commande	Description
x	Deletes the character currently under the cursor
dd	Deletes the line currently under the cursor
<i>n</i> dd	Deletes <i>n</i> lines from the one currently under the cursor
<i>n</i> x	Deletes <i>n</i> characters from the one currently under the cursor
<i>n</i> yy	Copies <i>n</i> lines from the line currently under the cursor
p	Pastes the contents of the clipboard
dw	Deletes the word under the cursor
d0 «(dzero)»	Deletes the cursor at the beginning of the line

d\$	Deletes the cursor at the end of the line
u	Uncancel the changes (successively)
G (« <i>shift+g</i> »)	Sends to the end of the file
nG	Sends to line n of the file

## 8.4 Find and Replace

To search for a word in a document, you just have to type / followed by the string you want to search for, then validate by pressing the enter key. It is then possible to go from occurrence to occurrence thanks to the n key.

To go back to the previous occurrence you have to use the key combination "shift+n".

To replace a string by another one on a line, there is a very powerful command under Vi using regular expressions. Here is its syntax:

```
:s/string_to_replace/replacing_string/
```

For a replacement on a given number of lines, for example we do the replacement on line 10 to 15:

```
:10,15s/string_to_replace/replacing_string/g
```

It is possible to generalize it to the whole document thanks to the following syntax:

```
:%s/string_to_replace/replacing_string/g
```

This document is not intended to be exhaustive, there are a wide variety of commands in Vi or Vim. Many pages on the internet are available to help you.

# Vim Commands Cheat Sheet

(from : <https://www.fprintf.net/vimCheatSheet.html>)

## How to Exit

<code>:q[uit]</code>	Quit Vim. This fails when changes have been made.
<code>:q[uit]!</code>	Quit without writing.
<code>:cq[uit]</code>	Quit always, without writing.
<code>:wq</code>	Write the current file and exit.
<code>:wq!</code>	Write the current file and exit always.
<code>:wq {file}</code>	Write to {file}. Exit if not editing the last
<code>:wq! {file}</code>	Write to {file} and exit always.
<code>:[range]wq[!]</code>	[file] Same as above, but only write the lines in [range].
<code>ZZ</code>	Write current file, if modified, and exit.
<code>ZQ</code>	Quit current file and exit (same as <code>":q!"</code> ).

## Editing a File

<code>:e[dit]</code>	Edit the current file. This is useful to re-edit the current file, when it has been changed outside of Vim.
<code>:e[dit]!</code>	Edit the current file always. Discard any changes to the current buffer. This is useful if you want to start all over again.
<code>:e[dit] {file}</code>	Edit {file}.
<code>:e[dit]! {file}</code>	Edit {file} always. Discard any changes to the current buffer.
<code>gf</code>	Edit the file whose name is under or after the cursor. Mnemonic: "goto file".

## Inserting Text

<code>a</code>	Append text after the cursor [count] times.
<code>A</code>	Append text at the end of the line [count] times.
<code>i</code>	Insert text before the cursor [count] times.
<code>I</code>	Insert text before the first non-blank in the line [count] times.
<code>gl</code>	Insert text in column 1 [count] times.
<code>o</code>	Begin a new line below the cursor and insert text, repeat [count] times.
<code>O</code>	Begin a new line above the cursor and insert text, repeat [count] times.

## Inserting a file

<code>:r[ead] [name]</code>	Insert the file [name] below the cursor.
<code>:r[ead] !{cmd}</code>	Execute {cmd} and insert its standard output below the cursor.

## Deleting Text

<code>&lt;Del&gt; or x</code>	Delete [count] characters under and after the cursor
<code>X</code>	Delete [count] characters before the cursor
<code>d{motion}</code>	Delete text that {motion} moves over

dd	Delete [count] lines
D	Delete the characters under the cursor until the end of the line
{Visual}x or {Visual}d	Delete the highlighted text (for {Visual} see <a href="#">Selecting Text</a> ).
{Visual}CTRL-H or {Visual}	When in Select mode: Delete the highlighted text
{Visual}X or {Visual}D	Delete the highlighted lines
: [range]d[delete]	Delete [range] lines (default: current line)
: [range]d[delete] {count}	Delete {count} lines, starting with [range]

## Changing (or Replacing) Text

r{char}	replace the character under the cursor with {char}.
R	Enter Insert mode, replacing characters rather than inserting
~	Switch case of the character under the cursor and move the cursor to the right. If a [count] is given, do that many characters.
~{motion}	switch case of {motion} text.
{Visual}~	Switch case of highlighted text

## Substituting

: [range]s[ubstitute]/{pattern}/	For each line in [range] replace a match of {string}/[c][e][g][p][r][i][I] [count] {pattern} with {string}.
: [range]s[ubstitute] [c][e][g][r][i][I] [count] : [range]&[c][e][g][r][i][I] [count]	Repeat last :substitute with same search pattern and substitute string, but without the same flags. You may add extra flags

The arguments that you can use for the substitute commands:

[c] Confirm each substitution. Vim positions the cursor on the matching string. You can type:

'y' to substitute this match

'n' to skip this match

to skip this match

'a' to substitute this and all remaining matches {not in Vi}

'q' to quit substituting {not in Vi}

CTRL-E to scroll the screen up {not in Vi}

CTRL-Y to scroll the screen down {not in Vi}.

[e] When the search pattern fails, do not issue an error message and, in particular, continue in maps as if no error occurred.

[g] Replace all occurrences in the line. Without this argument, replacement occurs only for the first occurrence in each line.

[i] Ignore case for the pattern.

[I] Don't ignore case for the pattern.

[p] Print the line containing the last substitute.

## Copying and Moving Text

"{a-zA-Z0-9.%#:-}"	Use register {a-zA-Z0-9.%#:-} for next delete, yank or put (use uppercase character to append with delete and yank) ({.%#:-} only work with put).
:reg[isters]	Display the contents of all numbered and named registers.
:reg[isters]	Display the contents of the numbered and named registers

{arg}	that are mentioned in {arg}.
:di[sp]lay [arg]	Same as :registers.
["x]y{motion}	Yank {motion} text [into register x].
["x]yy	Yank [count] lines [into register x]
["x]Y	yank [count] lines [into register x] (synonym for yy).
{Visual}["x]y	Yank the highlighted text [into register x] (for {Visual} see <a href="#">Selecting Text</a> ).
{Visual}["x]Y	Yank the highlighted lines [into register x]
:[range]y[ank] [x]	Yank [range] lines [into register x].
:[range]y[ank] [x] {count}	Yank {count} lines, starting with last line number in [range] (default: current line), [into register x].
["x]p	Put the text [from register x] after the cursor [count] times.
["x]P	Put the text [from register x] before the cursor [count] times.
["x]gp	Just like "p", but leave the cursor just after the new text.
["x]gP	Just like "P", but leave the cursor just after the new text.
:[line]pu[t] [x]	Put the text [from register x] after [line] (default current line).
:[line]pu[t]! [x]	Put the text [from register x] before [line] (default current line).

## Undo/Redo/Repeat

u	Undo [count] changes.
:u[ndo]	Undo one change.
CTRL-R	Redo [count] changes which were undone.
:red[o]	Redo one change which was undone.
U	Undo all latest changes on one line. {Vi: while not moved off of it}
.	Repeat last change, with count replaced with [count].

## Screen movement commands

z.	Center the screen on the cursor
zt	Scroll the screen so the cursor is at the top
zb	Scroll the screen so the cursor is at the bottom

## Marks

m{a-zA-Z}	Set mark {a-zA-Z} at cursor position (does not move the cursor, this is not a motion command).
m' or m`	Set the previous context mark. This can be jumped to with the "" or "`" command (does not move the cursor, this is not a motion command).
: [range]ma[rk] {a-zA-Z}	Set mark {a-zA-Z} at last line number in [range], column 0. Default is cursor line.
:[range]k{a- zA-Z}	Same as :mark, but the space before the mark name can be omitted.

'{a-z}	To the first non-blank character on the line with mark {a-z} (linewise).
'{A-Z0-9}	To the first non-blank character on the line with mark {A-Z0-9} in the correct file
`{a-z}	To the mark {a-z}
`{A-Z0-9}	To the mark {A-Z0-9} in the correct file
:marks	List all the current marks (not a motion command).
:marks {arg}	List the marks that are mentioned in {arg} (not a motion command). For example:

## Searching

/ {pattern} [/]	Search forward for the [count]'th occurrence of {pattern}
/ {pattern} / {offset}	Search forward for the [count]'th occurrence of {pattern} and go {offset} lines up or down.
/ <cr&gt;< td=""> <td>Search forward for the [count]'th latest used pattern</td> </cr&gt;<>	Search forward for the [count]'th latest used pattern
// {offset} <CR>	Search forward for the [count]'th latest used pattern with new. If {offset} is empty no offset is used.
? {pattern} [?] <CR>	Search backward for the [count]'th previous occurrence of {pattern}
? {pattern}? {offset} <CR>	Search backward for the [count]'th previous occurrence of {pattern} and go {offset} lines up or down
? <CR>	Search backward for the [count]'th latest used pattern
?? {offset} <CR>	Search backward for the [count]'th latest used pattern with new {offset}. If {offset} is empty no offset is used.
n	Repeat the latest "/" or "?" [count] times.
N	Repeat the latest "/" or "?" [count] times in opposite direction.

## Selecting Text (Visual Mode)

To select text, enter visual mode with one of the commands below, and use [motion commands](#) to highlight the text you are interested in. Then, use some command on the text.

The operators that can be used are:

- ~ switch case
- d delete
- c change
- y yank
- > shift right
- < shift left
- ! filter through external command
- = filter through 'equalprg' option command
- gq format lines to 'textwidth' length

v	start Visual mode per character.
V	start Visual mode linewise.
<Esc>	exit Visual mode without making any changes

## How to Suspend

CTRL-Z	Suspend Vim, like ":stop". Works in Normal and in Visual mode. In Insert and Command-line mode, the CTRL-Z is inserted as a normal character.
:sus[pend] [!] or :st[op][!]	Suspend Vim. If the '!' is not given and 'autowrite' is set, every buffer with changes and a file name is written out. If the '!' is given or 'autowrite' is not set, changed buffers are not written, don't forget to bring Vim back to the foreground later!