Chapter 1 – First steps on the Unix command line

As users of digital services we are all familiar with the notion of **application program**, a piece of software that performs a specific task, such as writing documents, playing games, or browsing the web. But applications are not designed to run alone on bare hardware: computers are too complex and too diverse for every application to support every machine. Instead, computers are equipped with an **operating system**, a set of utility programs that collectively form an execution platform for applications. Example of operating systems include Microsoft Windows, MacOS, Android, etc.

1 Setting up your work environment

In this course, we will be doing all our practical work on Linux. There are several flavours of Linux operating systems available out there, like Fedora, Ubuntu, Debian, etc. If you already have a favourite version of Linux, then use it and skip directly to section 3 on page 2. Otherwise, please read on.

The easiest way to get started is to use INSA's Virtual Desktop Infrastructure, as described in section 2 below. However, there are several alternatives that would work just as well (in no particular order):

- Install a Linux distribution on your laptop, possibly as a *dual boot* option alongside Windows.
- Create a bootable USB stick with a *live* version of e.g. Ubuntu or Fedora.
- Install a Linux distribution inside a virtual machine (through VirtualBox or VMWare).
- Use the Windows Subsystem for Linux (WSL).
- Work through SSH on a remote Linux host. For instance, INSA provides such machines with names of the form `2d-linux-NNN.insa-lyon.fr` (for NNN in 001 to 150).
- Not use Linux but another system from the Unix family, such as macOS. Most of this course (not all, but most) will still apply.

You are encouraged to explore (and/or combine several of) these solutions. Please don't hesitate to ask us for help, even though we cannot promise to solve all your setup problems. In all cases, please do give us feedback on your work environment of choice !

2 Using INSA's Virtual Desktop Infrastructure (aka "Bureau Virtuel")

Visit `https://bv.insa-lyon.fr/` and login with your INSA credentials[1]. You get a selection page looking like figure 1 below. Choose "2D-LINUX". You should then see an in-browser Linux desktop (cf figure 2).

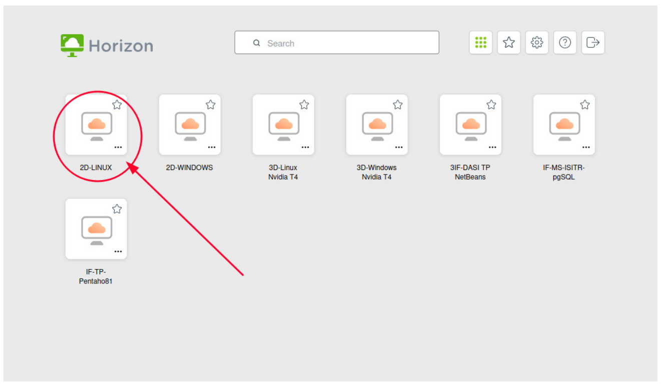


Figure 1: Selecting the Linux VM

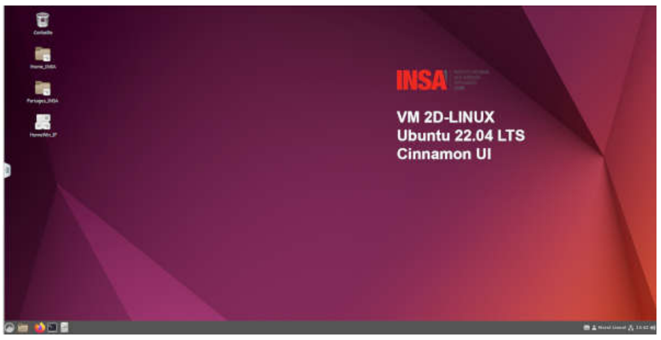


Figure 2: INSA BV's Linux desktop

[1]Note: The login form expects credentials of the form `username@insa-lyon.fr` (i.e. not just your `username` and not `email.address@insa-lyon.fr` either)

2.1 Switching the interface to English

By default, INSA's Bureau Virtuel is configured for French. Here is the procedure to switch the whole interface to English. First open the main menu in the bottom-left corner, and type "langues" in the search box and then click on "Prise en charge des langues" (see figure 3 on the next page). A small window appears, cf figure 4.
In the first tab "Langues", move English to the top of the list. In the "Format régionaux" tab, choose English from the dropdown menu. Click on "Fermer".

Now log out then reconnect to the 2D-LINUX virtual machine. A new window appears (see figure 5) asking you to "Update standard folders to current language". Click on "Update Names".

Your system is now configured to show all messages (menus, help pages, etc) in English.

2.2 Configuring the keyboard

By default, INSA's Bureau Virtuel is configured for French keyboards with AZERTY layout, which is probably not what you want.
In the main menu, click "Preferences" then "Keyboard" and go to the third tab "Layouts". Click "+" and add your preferred layout, then remove "French" with "–" and close the window.

3 Through the Looking Glass: First Steps in Unix Land

Modern Linux systems such as Ubuntu come with a **Graphical User Interface** (GUI) similar to Windows or MacOS. But for the purpose of this course, we will mostly interact with the system through its **Command-Line Interface** (CLI). Historically, command-line interfaces emerged in the 1960s as a more user-friendly alternative to punched cards. Sitting in front of a *computer terminal* (a screen+keyboard device connected to a computer) the user would type their inputs interactively and read the results as text in real-time.

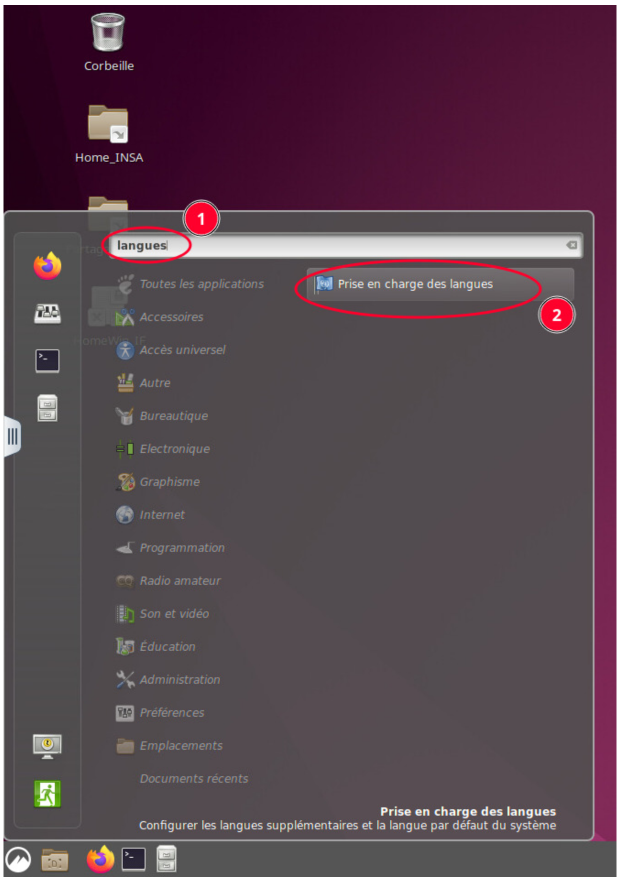
Nowadays, most users rely on graphical user interfaces for their everyday use. However, many programs and operating system utilities have no GUI, and are intended to be used through the command-line. For this reason, modern systems generally include a *terminal emulator* program, often referred to as just a **terminal**. This combination brings the benefits of both worlds: the expressive power of the command-line and the comfort of a graphical interface.

3.1 Terminal, Shell, Commands

Exercise 1 Open a terminal window. On the Bureau Virtuel, the Terminal application can be launched from the main menu (in "System Tools" cf figure 6) or by clicking on its icon in the task bar (cf figure 7).

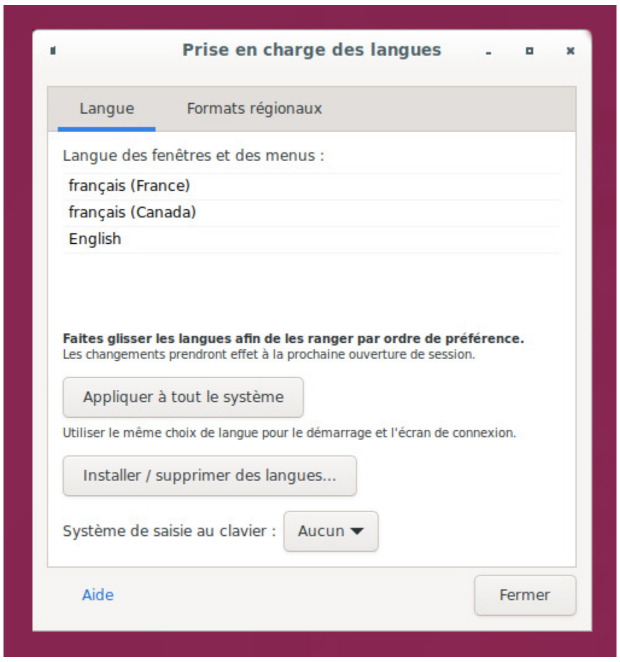
Exercise 2 Any text typed in the terminal window is fed to a **shell**[2], i.e. a command-line interpreter, in our case the `bash` shell. The shell presents itself as a **command prompt** (cf figure 8 on page 4) waiting for you to type a line of text. Start by typing your login name then press "enter": the shell will complain that it is not a valid command (cf figure 9 on page 4). The shell only understands a finite number of command names. We will explore some of them today.[3]

[2]yes, like an eggshell. cf `https://en.wikipedia.org/wiki/Shell_(computing)`
[3]Thanks to Igor Khmelnitsky for the inspiration. `http://www.lsv.fr/~khmelnitsky/teaching/2019-2020/arch_sys/tp01_eng.pdf`



Prise en charge des langues
Configurer les langues supplémentaires et la langue par défaut du système

Figure 3: Finding the "change your language" option.



Langue des fenêtres et des menus :
français (France)
français (Canada)
English

Faites glisser les langues afin de les ranger par ordre de préférence.
Les changements prendront effet à la prochaine ouverture de session.

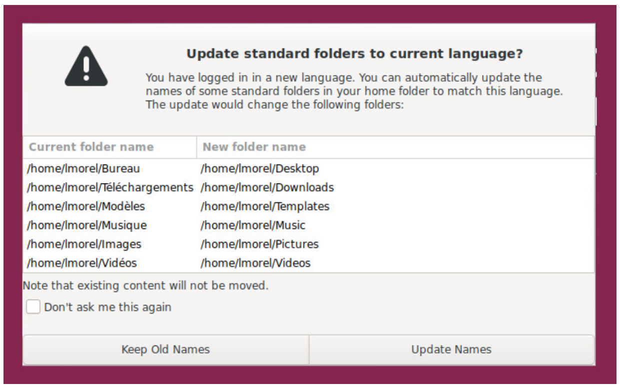
Appliquer à tout le système

Utiliser le même choix de langue pour le démarrage et l'écran de connexion.

Installer / supprimer des langues...

Système de saisie au clavier : Aucun

Figure 4: The window allowing to change the system's language.



Update standard folders to current language?

You have logged in in a new language. You can automatically update the names of some standard folders in your home folder to match this language. The update would change the following folders:

Current folder name	New folder name
/home/lmorel/Bureau	/home/lmorel/Desktop
/home/lmorel/Téléchargements	/home/lmorel/Downloads
/home/lmorel/Modèles	/home/lmorel/Templates
/home/lmorel/Musique	/home/lmorel/Music
/home/lmorel/Images	/home/lmorel/Pictures
/home/lmorel/Vidéos	/home/lmorel/Videos

Note that existing content will not be moved.

☐ Don't ask me this again

Figure 5: Updating standard folder names.

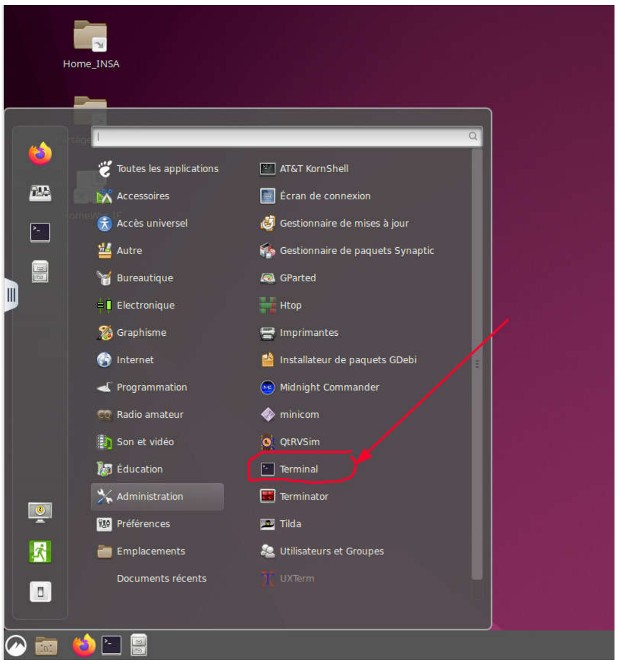


Figure 6: Terminal shortcut in the main menu.

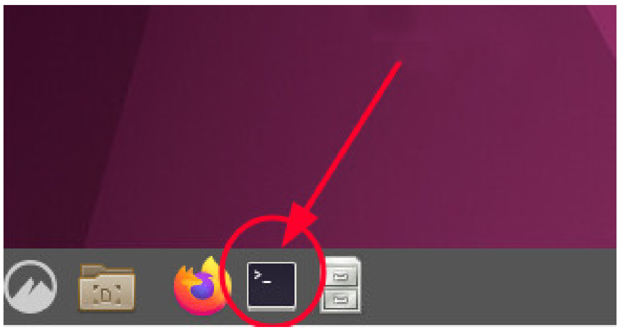
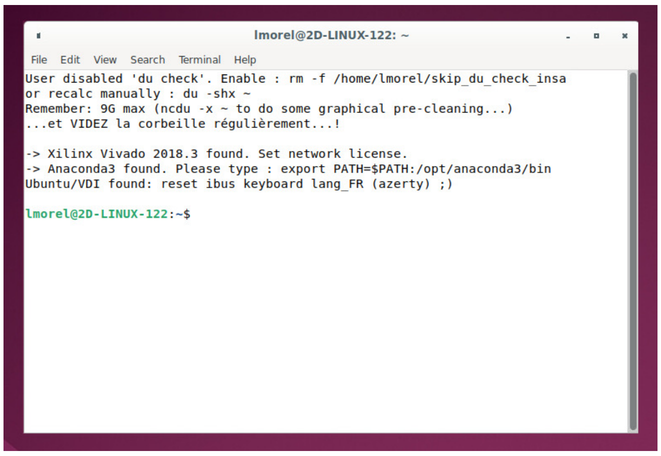


Figure 7: Terminal shortcut in the taskbar.

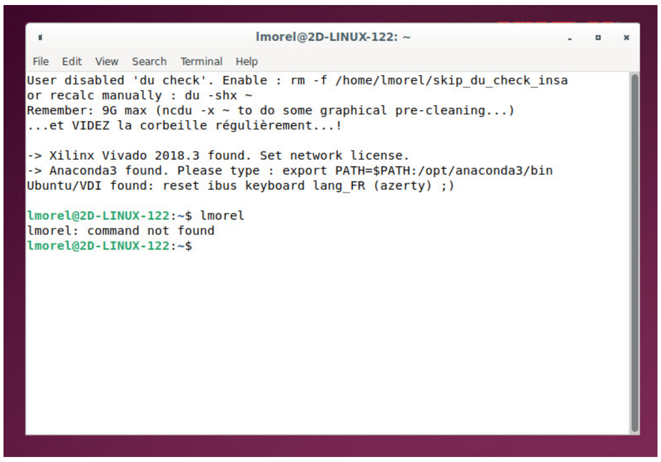


```
User disabled 'du check'. Enable : rm -f /home/lmorel/skip_du_check_insa
or recalc manually : du -shx ~
Remember: 9G max (ncdu -x ~ to do some graphical pre-cleaning...)
...et VIDEZ la corbeille régulièrement...!

-> Xilinx Vivado 2018.3 found. Set network license.
-> Anaconda3 found. Please type : export PATH=$PATH:/opt/anaconda3/bin
Ubuntu/VDI found: reset ibus keyboard lang_FR (azerty) ;)

lmorel@2D-LINUX-122:~$
```

Figure 8: A new terminal window, ready to accept commands.



```
User disabled 'du check'. Enable : rm -f /home/lmorel/skip_du_check_insa
or recalc manually : du -shx ~
Remember: 9G max (ncdu -x ~ to do some graphical pre-cleaning...)
...et VIDEZ la corbeille régulièrement...!

-> Xilinx Vivado 2018.3 found. Set network license.
-> Anaconda3 found. Please type : export PATH=$PATH:/opt/anaconda3/bin
Ubuntu/VDI found: reset ibus keyboard lang_FR (azerty) ;)

lmorel@2D-LINUX-122:~$ lmorel
lmorel: command not found
lmorel@2D-LINUX-122:~$
```

Figure 9: An invalid command line.

Exercise 3 Type command `date` and observe the results. A command line is a list of words separated by spaces: the first word is the command name and the rest are the **arguments**. By convention, arguments starting with a dash are called **options**. Type command `date -d yesterday` and compare the results.
It is the responsibility of the shell to split the line into word arguments before running the actual command. When needed, we can also group multiple words into a single argument, by enclosing them in quotes, e.g. `date -d "next month"`.

Exercise 4 You have probably already come across other types of prompts. For instance in the Python interpreter, the prompt looks like “>>> ”. Type `python3` and play with python for a bit. Notice how our input is not interpreted by `bash` anymore, until we type `quit()` to terminate python and go back to the shell. What happens is, a terminal can accommodate several programs at the same time but only one of them can be **in the foreground** at any point. All programs can print text on the terminal, but only the foreground program will receive keyboard input.

Exercise 5 While in the middle of typing a command line, you can press the “TAB” key and the shell will try to **auto-complete your command**. Type “`dat`” and then press “TAB”. Press Ctrl-C to cancel. If there are has a multiple matches for completion, the shell will do nothing, but pressing “TAB” again will print out all of the completion matches. Type just “`da`” and then press “TAB” twice. Notice how there are several matches. Type an additional “t” and press “TAB” again.

Exercise 6 You may have noticed that in a terminal, pressing Ctrl-C does not trigger the usual copy-and-paste mechanism[4], but instead it means “please stop the current foreground command”. However, you can copy-and-paste with Shift-Ctrl-C and Shift-Ctrl-V.

3.2 Looking for help

Of course there are many ways to find help:

- invoke a command with option `--help` at the end usually shows a help screen, e.g. `date --help`,
- use a web search engine, e.g. visit `https://duckduckgo.com` and type “unix date command”.
- ask a fellow student or teacher,
- but when everything else fails, you might want to just RTFM.

Linux comes with a builtin documentation system that is accessible through a dedicated shell command named `man` (short for “manual”) You can type `man` followed by the name of a command and you will see some documentation for that command. All **man pages** are organized in a similar way:

[4]this is for historical reasons: unix terminals and shells predate the idea of copy-and-paste by quite some time

1. NAME : name and purpose
2. SYNOPSIS : short summary of the command-line syntax
3. DESCRIPTION : long description, typically listing all supported options
4. sometimes other sections, like EXAMPLES, or KNOWN BUGS
5. AUTHOR : the people who developed the program
6. SEE ALSO : references to other docs: man pages, websites, etc

Exercise 7 Type `man date` and try the following: scroll with up/down arrows, advance by one screen with “space”, and go back to the top with “g”. To search for some text, press “/” then your search query followed by “enter”, then use keys “n” and “N” (i.e. Shift-N) to navigate between search results. Press “h” to get some help and press “q” to exit and return to the shell.

3.3 Essential commands: ls, cd

Exercise 8 Browse the man page for `ls` and answer the following questions:

1. In no more than 10 words, what is the purpose of this command ?
2. Which option will print the name of all the files, even the “hidden” ones ? (by default, filenames starting with a dot are not shown, for instance `.bashrc`)
3. Which option will print the listing with a longer, more detailed format ?
4. Which option will print the size of the files in a human readable way ?
5. Which option will recursively print all the subdirectories of a given directory ?

Note the file named `.bashrc` is a configuration file for the `bash` shell (more on that later)

Exercise 9 Read the man page for `cd` and find our what this command is for. Then:

- Go to the `/tmp` directory at the root of the system and list its contents;
- Return to your home directory;
- What does `cd -` do ? What does `cd` with no arguments do ?

4 Working With Files

4.1 File system organisation

One of the key principles of the Unix philosophy is that “everything is a file”. To keep things organized, all files in the system are structured as a giant tree, where every node has an associated name, like illustrated in figure 10. Internal tree nodes are called **directories** (aka folders) and they contain other directories as well as ordinary files. The topmost node is called the **root directory** and is denoted by “/”. Every node can be denoted by its **path** from the root (e.g. `/home/lmorel/.bashrc`). Some important directories:

- `/bin` is where most of the shell commands are stored, e.g. `/bin/date` or `/bin/ls`
- `/home` contains all the users' personal directories
- `/home/yourlogin` is your personal **home directory**. In the shell, you can type “~” as a shortcut for this path, e.g. `ls ~`
- `/tmp` is for temporary files. It is emptied every time the system reboots

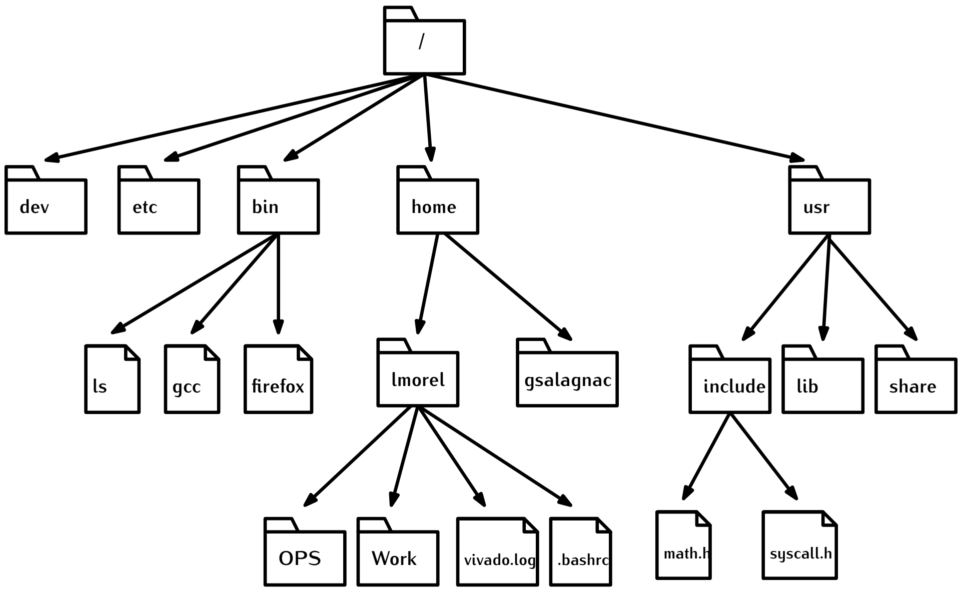


Figure 10: Typical directory hierarchy of a Linux filesystem.

A command-line shell provides basically the same features as a graphical file manager. In the following exercises you will learn how to create new files and directories (sections 4.2 and 4.3), how to copy and move things around (section 4.4), how to look into files (section 4.5), and also how to delete them permanently (section 4.6).

Warning A CLI shell is a powerful tool and it will blindly obey your commands. However, what you type may or may not be what you wanted. Compared to a typical GUI system, there are fewer safety nets and some actions cannot be “undone”. This can feel intimidating at first, but having such a powerful tool under your belt will often prove very useful.

4.2 Creating directories: mkdir

Exercise 10 Type `man mkdir` and read the manual page. Within your home directory, create a directory named “`ops`”.
Now we want to create a directory with path “`~/a/b/c/d`”. Look in the `mkdir` man page for the option to create this entire directory hierarchy in just one command.

Exercise 11 Try to create a directory under the `/usr` folder. What error message do you get in response? Explanation: an ordinary user is allowed to read/write everything under their homedir, but the rest of the filesystem is only accessible read-only, to protect the system against (accidental or intentional) damage. You would need administrator (aka super-user, aka root[5]) privileges to modify these files (cf section 4.7 on the following page).

4.3 Creating empty files: touch

Exercise 12 Read the first lines of the `touch` man page. Create an empty file named `readme.txt` inside the `ops` folder. Using `cd` and `ls`, make sure your file has been created.

4.4 Copying and moving things around: cp and mv

Exercise 13 Browse the man page for the `cp` command.
Use it to create a copy of your `readme.txt` file named `readme_new.txt`.
Try to copy the `readme_new.txt` to a file that has the same name. What happens?
The `cp` command can also be used to copy folders. Look at the man page again and read the description of the `-r` option.

Exercise 14 Copy the whole the `ops` folder to a new location e.g. `ops_new`. Enter that folder, and check that all the original contents has been duplicated here.

The `mv` command lets you move files and directories around. Now instead of duplicating things, we can change their location anywhere in the file system. Anywhere? well, only in places we have write-access to, of course. But we'll cover this later.

Exercise 15 Create a new directory named `work` inside your home folder. Note that you can do that from anywhere in the file system, by specifying the full path, e.g `mkdir /home/yourlogin/work` or even better, by using the "~" shortcut.
Now move your original `ops` folder to place it under your `work` directory.

4.5 Looking at files

Many files in the system are so-called **plain text** documents i.e. they just contain ASCII characters with no formatting. Plain text files are important for the rest of the course, and for programming in general, because all tools in the programming universe (compilers, editors, etc) work with those.

Exercise 16 Go into the `~/work/ops` directory, and type `code readme.txt`. This opens Visual Studio Code, a simple IDE that you will use to write your programs.
Modify the file by adding some lines, save it.

Exercise 17 We can edit files with a graphical editor. But we can also display their content from the command line directly. This is often interesting for quickly checking some info (configurations of system variables, quick tour of a program, etc).
Back in the shell, use commands `cat` and `less` to observe the content of the `readme.txt` file.
What is the difference between these two commands ? Browse their respective man pages to find out.

[5]Warning: Don't get confused, there are two different concepts involving that same word: "root directory" https://en.wikipedia.org/wiki/Root_directory vs "root user" https://en.wikipedia.org/wiki/Superuser

4.6 Deleting stuff: rm and rmdir

Exercise 18 Browse the man pages for commands `rm` and `rmdir`.
Navigate to your `~/work/ops/` folder and type `rm readme.txt` to remove the file. Go back to the parent folder with `cd ..`. From there, we now want to delete the whole `ops` folder.
Try command `rm ops` and watch it fail:

```
$ rm ops
rm: cannot remove 'ops': Is a directory
```

Now try and type `rmdir ops`, which fails too:

```
$ rmdir ops
rmdir: failed to remove 'ops': Directory not empty
```

Indeed, the `ops` folder still contains the `readme_new.txt` file.

There are two ways to actually delete the entire `ops` directory:

- First remove everything inside, i.e. go into the folder, and remove the `readme_new.txt` file. Then, remove the (now empty) directory itself with `rmdir`.
- Force the removal of the whole `ops` hierarchy by typing `rm -rf ops`. Go read the `rm` man page again to learn more about these options.

Warning! Saying `rm -rf` asks for deletion of a whole directory tree, without further confirmation. Use this command with extreme caution or you will lose precious data.

Exercise 19 To make these commands less error-prone, open your `~/.bashrc` file in VSCode and add the following lines at the bottom:

```
alias rm="rm -i"
alias cp="cp -i"
alias mv="mv -i"
```

What does this `-i` option do ?

4.7 File ownership and access rights

In the shell, go back to your home directory and type `ls -l` You will see something comparable to figure 11.

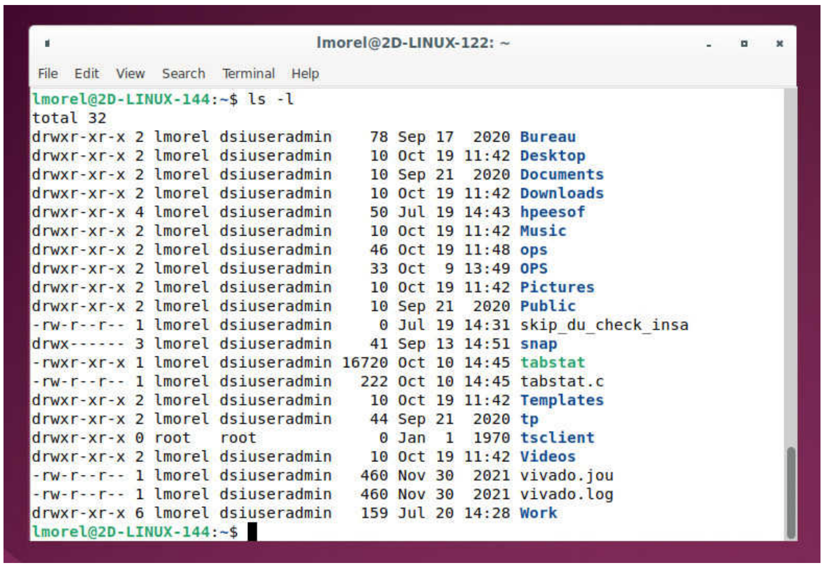
Let's go through all columns, from right to left. For each file, we have: its name, its timestamp of last modification (aka **mtime**),[6] its size (in bytes)[7] , then some ownership info (login and group names), then its *link count* (let's ignore that for now), and finally a series of `rwx` letters.

The leftmost column describes the **access rights** to the file, in a sequence of characters similar to e.g. "`-rw-r--r-`" or "`drwxr-xr-x`". You should read this sequence in four groups of 1, 3, 3, and 3 boolean flags, respectively:

- The very first character is "`-`" for an ordinary file or "`d`" for a directory.[8]
- The first `rwx` trigram indicates the access rights (read, write, execute) for the file's owner
- The second `rwx` trigram indicates the access rights for the members of the file's owning group (let's ignore groups for now)
- The last `rwx` trigram indicates the access rights for everyone else.

The `x` flag means *execute*: for an ordinary file, it means that the file contains an executable program, which we can run from the the shell by typing its name. For directories, it means that one is allowed to *navigate* (e.g. with `cd`) to that directory.

[6]Try to `touch` a file and observe that it changes its mtime
[7]Warning: a directory's size info has nothing to do with the total size of files in that directory.
[8]Or it can be "`l`" for a *symbolic link* to another path, but we'll ignore that for now



```
lmorel@2D-LINUX-144:~$ ls -l
total 32
drwxr-xr-x 2 lmorel dsiuseradmin    78 Sep 17  2020 Bureau
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Desktop
drwxr-xr-x 2 lmorel dsiuseradmin    10 Sep 21  2020 Documents
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Downloads
drwxr-xr-x 4 lmorel dsiuseradmin    50 Jul 19 14:43 hpeesof
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Music
drwxr-xr-x 2 lmorel dsiuseradmin    46 Oct 19 11:48 ops
drwxr-xr-x 2 lmorel dsiuseradmin    33 Oct  9 13:49 OPS
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Pictures
drwxr-xr-x 2 lmorel dsiuseradmin    10 Sep 21  2020 Public
-rw-r--r-- 1 lmorel dsiuseradmin     0 Jul 19 14:31 skip_du_check_insa
drwx------ 3 lmorel dsiuseradmin    41 Sep 13 14:51 snap
-rwxr-xr-x 1 lmorel dsiuseradmin 16720 Oct 10 14:45 tabstat
-rw-r--r-- 1 lmorel dsiuseradmin   222 Oct 10 14:45 tabstat.c
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Templates
drwxr-xr-x 2 lmorel dsiuseradmin    44 Sep 21  2020 tp
drwxr-xr-x 0 root   root             0 Jan  1  1970 tsclient
drwxr-xr-x 2 lmorel dsiuseradmin    10 Oct 19 11:42 Videos
-rw-r--r-- 1 lmorel dsiuseradmin   460 Nov 30  2021 vivado.jou
-rw-r--r-- 1 lmorel dsiuseradmin   460 Nov 30  2021 vivado.log
drwxr-xr-x 6 lmorel dsiuseradmin   159 Jul 20 14:28 Work
lmorel@2D-LINUX-144:~$ 
```

Figure 11: Output of the `ls -l` command.

Exercise 20 Use `ls -l` to explain why you can create new files and directories under `/tmp` but not under `/usr`.

Exercise 21 If you want to know more about access rights, read the man page of command `chmod` and try it out on a few examples. Otherwise you can probably forget about it.

Exercise 22 Even without the `-l` option, command `ls` can tell us some details about the file with the `-F` option. Try it out, then find it in the `ls` man page. Open your `~/.bashrc` once more and add the aliases below. Then play with them in your home directory.

```
alias   ls="ls -hF"
alias   ll="ls -l"
alias   la="ls -la"
alias  lat="ls -lat"
alias lart="ls -lart"
alias larS="ls -larS"
```