# Pediluve — Tutorial D1

version **#dirty**


**Assistants C/Unix 2024** <assistants@tickets.assistants.epita.fr>

## Copyright



## Contents



*https://intra.forge.epita.fr

# 1 Foreword

The goal of this tutorial is to get started with basic principles and commands of *Linux*.

As such, you will see that each section is quite long, but what we ask you to do is either elementary or quick to perform. We do not ask you to mechanically learn *all* these new things in detail, mainly because you will assimilate them automatically as you start using them, but also because it is very important that you know where to find the information you need. On the other hand, there are basic notions that must be understood without doubt. Consider this tutorial to be your *survival kit*!

Be curious and ask yourself: What kind of problem did I just solve? How can I go further? How can I combine all these simple notions to achieve something greater? Think about that and you should make the most out of this session.

When you first launch i3, you will have to choose your modifier key. By default, it is the `Windows` key.

A modifier key, also called `Mod` or `Meta`, is a special key on your keyboard, which allows you to send commands to a program when it is held down with other keys. This key is also known as the `Super` key.

## 1.1 Useful commands to get started

Here are some useful commands to navigate and use your current window manager (i3):

- Meta + Enter: open a terminal
- Meta + {up bottom left right}: switch to respective windows
- Meta + shift + {up bottom left right}: move window to respective direction
- Meta + d: enter dmenu (upper-left corner of the screen). It allows you to run a program (for example, `firefox`)
- Meta + shift + q: close the current window
- Meta + {1 2 3 ...}: switch to workspace {1 2 3 ...}
- Meta + shift + e: prompt the logout box

The best way to understand them is to **try them**.

# 2 Terminal and shell

The notions in this section are **very** important. The shell is the main interface you will use to do things on the school's computers, it will be your main tool to explore files, launch programs, create folders, submit exercises and projects, and more. Moreover, you will be regularly evaluated on it, so make sure you understand its basic functionalities, practice diligently and do not hesitate to ask the assistants for help if you have any questions.

## 2.1 What is the shell?

In order to use a computer, you make use of an operating system. The operating system is the piece of software on your computer that knows how your hardware (hard drives, screen, GPU, keyboard, etc.) works and effectively does all the operations that involve the hardware. For example every program that reads or writes files from your hard drive or display things on your screen must do so by asking the operating system to send the right instructions to the drive or the screen.

The shell is just a textual user interface for the operating system. Using it allows you to directly and simply manipulate files, launch other programs, get information about your system, its hardware and its files, and many other things. Other programs can give you similar functionalities, and sometimes graphicaly (like the `explorer` on windows), but none is as versatile nor as direct as the shell.

The english word *shell* was chosen because it kind of acts as a carapace around your operating system, giving you access to its functionalities.

## 2.2 Practice time

Time to work! To get started, open a terminal (actually a terminal *emulator*). There are many of them, such as `alacritty`, `xterm` or `urxvt`, each having a different set of features. On the school's computers, and depending on what mod key you chose for i3's key combinations, the shortcut `windows + enter` or `alt + enter` should open the default terminal.

The terminal is a simple window capable of displaying text and reading keyboard input. By default, when a terminal opens, it automatically runs an other program: the *shell*. The shell will write something on the terminal and wait for your input.

When you open your terminal you should see something like this:

```
42sh$
```

This is the *prompt* of your shell, ours is `42sh$`, yours will probably be a bit different and longer but should still end with a `$`. The shell first displays this prompt and then waits for you to type a command.

**Tips**

You will discover that there are many ways to customize your shell, *and* your terminal.

Do not fall for the simplicity of copy/pasting examples found on the internet, try to understand *how* and *why* what you want to copy works, and you will soon be completely autonomous (which will be vital for exams where you will need to use the shell, since the internet will not be accessible in those situations). You should be able to explain why and how each command you are using works. This rule is valid for your shell, your editor, etc. and is generally considered a good practice.

Commands can be as simple as a single word: the name of a program to run.

```
42sh$ date
Wed Jun  1 03:27:29 PM CEST 2022
42sh$
```

In this example, we are using the `date(1)` program to display the current date and time. You can see that the `date` program wrote some things on the terminal, and that it then terminated because the shell displays its prompt again, allowing you to type another command.

Most programs can take arguments, such as flags (options) and data; these must be written after the program name and are separated by spaces. Flags are easy to recognize because they start with one or two dashes (-). For example, the `--universal` flag of `date` can be used to display the current date in the UTC (Universal Time Coordinated) timezone, instead of system's timezone:

```
42sh$ date --universal
Wed Jun  1 01:27:13 PM UTC 2022
42sh$
```

Some flags *themselves* can take an argument, in this case you must type in the flag followed by its argument, still separated by a space. If the argument contains a space it must be enclosed in quotes. In the following example, we are using `date` to display a date and time that is 3 weeks, 5 day, 12 hours and 15 minutes from now.

```
42sh$ date --date '3 weeks 5 days 12 hours 15 minutes'
Tue Jun 28 03:41:48 AM CEST 2022
42sh$
```

Notice that in this case, the `--date` flag of `date` *must* be given an argument. If you try to give only the flag, the program will print an error message. Most programs will tell you if you gave them an invalid set of arguments.

**Tips**

If you realize you typed the command wrong, maybe you forgot one of the quotes (`'`) for example, and you want to type the same command again while changing only a part of it, you do not need to type it all again from the start. Your shell maintains a history of all the commands you typed and you can recall the last one by using the `up` key on your keyboard, then `left` and `right` to move the cursor around and change parts of it before hitting `enter` again. There are a lot of keyboard shortcuts available on your shell that can make your life easier and a lot faster once you get used to them, more on that as a bonus at the end of this section.

```
42sh$ date --date
date: option '--date' requires an argument
Try 'date --help' for more information.
42sh$
```

Finally, commands can take data to work on, once again separated by spaces. The `factor` command, for example, will display the integer factorization of each number you gave it:

```
42sh$ factor 1 121 12321 1234321 1234567654321 12345678987654321
1:
121: 11 11
12321: 3 3 37 37
1234321: 11 11 101 101
1234567654321: 239 239 4649 4649
12345678987654321: 3 3 3 3 37 37 333667 333667
42sh$
```

Sometimes, you will want to give some data that starts with a dash, which can be an issue, because the program will interpret it as a flag and either print an error message because it does not recognize it as a known flag (if you are lucky) or will recognize a flag it knows and do something that you did not plan for at all:

```
42sh$ factor -42
factor: invalid option -- '4'
Try 'factor --help' for more information.
42sh$
```

To solve this ambiguity, there is a special flag you can give to most programs telling them that everything past this will be data and not flags, even if it starts with a dash. This special flag is '--', in our example we would use it like so:

```
42sh$ factor -- -42
factor: `-42' is not a valid positive integer
42sh$
```

Here the program `factor` understands that after the `--`, everything is a number it should work on, and not a flag. It still prints an error message but only because `-42` is a negative integer and it can only work on positive integers.

These commands are simple and useful, but they can also be much more complex, as you will see in the rest of this course. Most operations you will do on the school's computers, like displaying the content a directory, copying, renaming, moving and removing files, will be done by typing a command in a shell. The lack of graphical interface can be a little hard at first, but keep practicing and you will soon be as efficient with it as with any graphical interface for everyday operations, if not more. A well-mastered shell and a keyboard is no match for the best graphical interface there is and a mouse.

**Tips**

Linux has two kinds of *clipboard* for copying and pasting. The first is the clipboard you are familiar with, usually used with `Ctrl + C` and `Ctrl + V` on most software. The second one is called primary selection, and works more implicitly: you only have to select the text you want to copy, and you can paste it using the middle button of your mouse or `Shift + Insert`. On `alacritty` for example, you would use these shortcuts:

| | |
|---|---|
| `Ctrl + Shift + C` | Copy selection to the clipboard. |
| `Ctrl + Shift + V` | Paste content of clipboard. |
| `Shift + Insert` | Paste content of primary selection. |
| `Ctrl + Insert` | Copy content of primary selection. |

Inside most terminals, `Ctrl + C` and `Ctrl + V` have different meanings than copy and paste, prefer to use the primary selection when in doubt, or search for the documentation and configuration of your terminal to learn its key bindings.

For now, we **heavily encourage** you to type the examples and commands by yourself to practice, rather than copying and pasting them in any way.

### 2.3 Bonus: quick history of existing shells

The first shell was the Thompson shell, written in the 1970s, and was minimalist, it led to the better known Bourne shell (from the name of its creator: Stephen Bourne) also known as `sh`, the common ancestor of all modern shells. Here's a little overview of some of the most well-known shells:

- `bash`: the Bourne-Again shell, named as wordplay on the Bourne shell it replaced in 1989. It is part of the GNU Project[1] and probably is the most wildly used shell in the world. It is the default shell on most Linux distributions, and it is the **default shell installed on the PIE.**
- `zsh`: the Z shell, it inherits heavily from `bash`, and provides enhanced automatic completion of commands, options and arguments, shared history, typographical errors corrections, and more.
- `fish`: considered an "exotic" shell because of its many important differences with other popular shells like `bash` and `zsh`. it is a modern shell that is still pretty popular because of its default behavior some prefer to those of more classic shells.
- `csh`: mostly historic, the *C* shell was mostly used on BSD systems and is known for bringing a syntax closer to the *C* language to shell scripting (`tcsh` is its enhanced successor and is currently the default FreeBSD shell).
- `ksh`: mostly historic too, the Korn shell (from its creator: David Korn), includes many features from `csh` and introduces important features such as job control, or `emacs` and `vi` readline editing styles. `csh` and `ksh` were great inspirations for `bash`, which included many of their features when it came out.

## 3 Window managers and i3

By now, you should have noticed you are not using a conventional desktop environment but something called `i3`.

`i3` is what we call a *window manager* (WM). A window manager is a very lightweight piece of software that will organize your windows for you. `i3` aims to maximize screen space and lets you focus on more important things than rearranging your windows by yourself. `i3` is probably the best-known window manager in its category but there are other alternatives (Awesome, xmonad, dwm, just to name a few) you might want to try them if you install Linux on your own computer. However, you will only be allowed to use `i3` during the year, so you should try to get comfortable with it as soon as possible.

All window managers, including `i3`, are also highly customizable, so you can change how it looks, how it behaves, and keyboard shortcuts to your liking via your WM's configuration file. You should go take a look at the documentation in your free time to see how you could do that.

[1] https://www.gnu.org/

## 3.1 i3lock

### 3.1.1 Why?

**Be careful!**

At school, there are three kinds of people: the nice ones, the evil ones and the prankster ones. While the first kind is the most common, the last two can easily ruin your day if you are not careful. When you leave your seat without locking your session (protecting it with a password or something similar), anybody can come and do whatever they want, for example:

- Steal, delete, or modify your files.
- Cheat off your work in which case you would both receive a flag for cheating.
- Retrieve the passwords you stored unprotected.
- Do something forbidden, break the law, usurp your currently logged-in accounts.
- Worst of all: change your keyboard layout to bépo.

To protect yourself while not having to close and open a session over and over again all the time, simple tools exist, called *screen lockers*. The one we ask you to use is `i3lock`. It is very simple to use: when you want to unlock your session, type in your password and press Enter.

## 3.2 Exercises

You can now try to `i3lock` first with your shell, and then using the `dmenu`. Try to understand what differences it makes in the use of your computer and choose which way you prefer.

**Going further...**

If you have finished reading this tutorial early, an easy and actually useful way to try customizing `i3` is to add a keyboard shortcut to run `i3lock` (`Meta+Ctrl+l` is a good candidate if you lack ideas).

# 4 Editors

There is a historical rivalry in the programming community regarding which is the best text editor between:

- Emacs: https://www.gnu.org/software/emacs
- Vim: https://www.vim.org

These are the most well-known text editors of the free software community. Both of them are actively maintained and provide a large amount of features, without even talking of the extension possibilities.

**Be careful!**

In the two following parts, you must try out each command we show you. Moreover, there is an exercise at the end of this section to test them. Do not hesitate to go further and practice a little.

After this, you will be asked to use one of these editors (or both) throughout this year.

### 4.1 Vim

Vim is a text editor built with flexibility, usability, and minimalism in mind. It is not easy to get started, but once the first step is made the possibilities expand rapidly and after some time your efficiency will be hardly comparable.

Vim is a modal editor, which means it contains multiple **modes** and each mode has a single purpose and its own set of specific rules.

The two primary modes are **Normal mode**, which is, as its name suggests, the default one, and **Insert mode**, where you can directly insert text.

Thus, instead of using keys like `Ctrl` or `Alt` to perform specific actions like saving or copying and pasting, you will leave the **Insert mode** for another mode dedicated to command execution: the **Normal mode**.

To take a quick tour and learn how to use basic features of Vim, start the `vimtutor` program. To use Vim, type `vim` in your shell.

Here is a selection of some of the fundamental commands of **Normal mode**:

| | |
|---|---|
| `i` | Switch to **Insert mode**. |
| `h j k l` | Move the cursor, you can also use the arrow keys. |
| `yy` or `Y` | Copy the current line. |
| `dd` | Cut the current line. |
| `cc` | Change the current line. |
| `p` or `P` | Paste. |
| `gg` | Go to first line. |
| `G` | Go to last line. |

**Tips**

- You can see if you are in **Insert mode** by checking the bottom-left corner of your terminal.
- When in **Insert mode**, you can go back to **Normal mode** using the `Esc` key.
- You **should not** stay in **Insert Mode** all the time. Doing so destroys the purpose of Vim.[1]
- Vim commands form a language. Learn it and you will be **very** efficient!

[1] The rationale behind modes is simple: **writing text** is different from **editing text**.

By typing the ':' key, you enter the command-line mode, used to enter Ex commands. Here are some Ex commands you may find useful:

| | |
|---|---|
| `:q` | Exit vim. |
| `:w` | Save the current buffer. |

You can find an extensive list of vim commands on the documents section on the Assistant's Intranet.

**Tips**

If you need help for any of these commands, you can use `:help command` to search for this particular command. You can also search for other entries in the help system, such as `:help find-manpage`.[2]

[2] Vim's help pages are very well-made and complete. You should read them, even if you already know Vim pretty well.

**Going further...**

Any Linux system compliant with the *Linux Standard Base* (LSB) specification must provide the `vi` text editor. Vim is the continuation of this utility, and thus you will not be lost if you ever get to use a legacy system that only has the `vi` text editor installed. To be compliant with the *LSB*, some systems ship the original `vi` package, while others fall back on Vim in compatible mode.

Try the `vi` utility on your system. Is it the original `vi`, or Vim in compatibility mode?

## 4.2 Emacs

**Emacs** is the editor from the GNU project. Unlike vim, you are almost always inserting text and have to use keyboard shortcuts to perform special actions. It also has a notion of modes, although it is very different from vim ones.[3] It is a very powerful editor with unmatched customization capabilities.

To use Emacs, type `emacs -nw` in your shell. The `-nw` option tells Emacs to start inside your shell. Ignoring it will start it in a new window.

As we said earlier, you can perform all actions using keyboard shortcuts. The following notation is used to describe shortcuts:

- '`C-`' represents the `Ctrl` key pressed simultaneously with the key that follows.
- '`M-`' represents the `Meta` key pressed simultaneously with the key that follows. The `Meta` key is the `Alt` key by default, but other keys can be associated with it.
- '`SPC`' represents the spacebar.

Here is a selection of basic Emacs shortcuts:

| | |
|---|---|
| `C-x C-c` | Exit Emacs. |
| `C-x C-s` | Save the current buffer. |
| `C-a` | Move the cursor to the beginning of the line. |
| `C-e` | Move the cursor to the end of the line. |
| `C-k` | Cut from the cursor to the end of the line. |
| `C-SPC` | Place a mark and start a selection. |
| `M-w` | Copy the current selection. |
| `C-w` | Cut the current selection. |
| `C-y` | Paste. |
| `M-<` | Go to first line. |
| `M->` | Go to last line. |

[3] Emacs makes a distinction between **major modes** and **minor modes**. A **major mode** is basically the mode for the language you are editing (like `c-mode` for example). **Minor modes** are independent of **major modes** and add additional functionalities to the editor.

Finally, just like vim's command-mode, you can use commands in Emacs with `M-x`. It will then ask you which command to run.[4] For example, you can run a command you might find very useful during your semester: `delete-trailing-whitespace`.

You can find an extensive list of Emacs commands on the documents section on the Assistant's Intranet. **Some of these shortcuts also work in your shell, as well as in most GNU tools!**

## 4.3 Exercise

To get help for the following exercise, look into the help systems of the editors.

**Tips**

if you have questions about how to do something in the editor do not hesitate to read documents about it, in the official documentation inside the editors or in the cheat sheet given by the ACUs

For each text editor, execute the following actions using the respective command system:

1. Start the editor.
2. Vertically split the window.
3. In the left window, create or open a new file in your home directory, write five lines of text in it.
4. In the right window, split horizontally.
5. Go back to the left window, go to the first line and copy/paste it three times to the end of the buffer.
6. Copy the whole content of the buffer to the bottom-right window.
7. Close all windows except the bottom right one.
8. Save the current buffer in a new file of the current directory.
9. Quit the editor.

## 4.4 Do it yourself

You are now free to practice on the vim tutorial (`vimtutor`) or the Emacs tutorial (`C-h t` on Emacs).

**Be careful!**

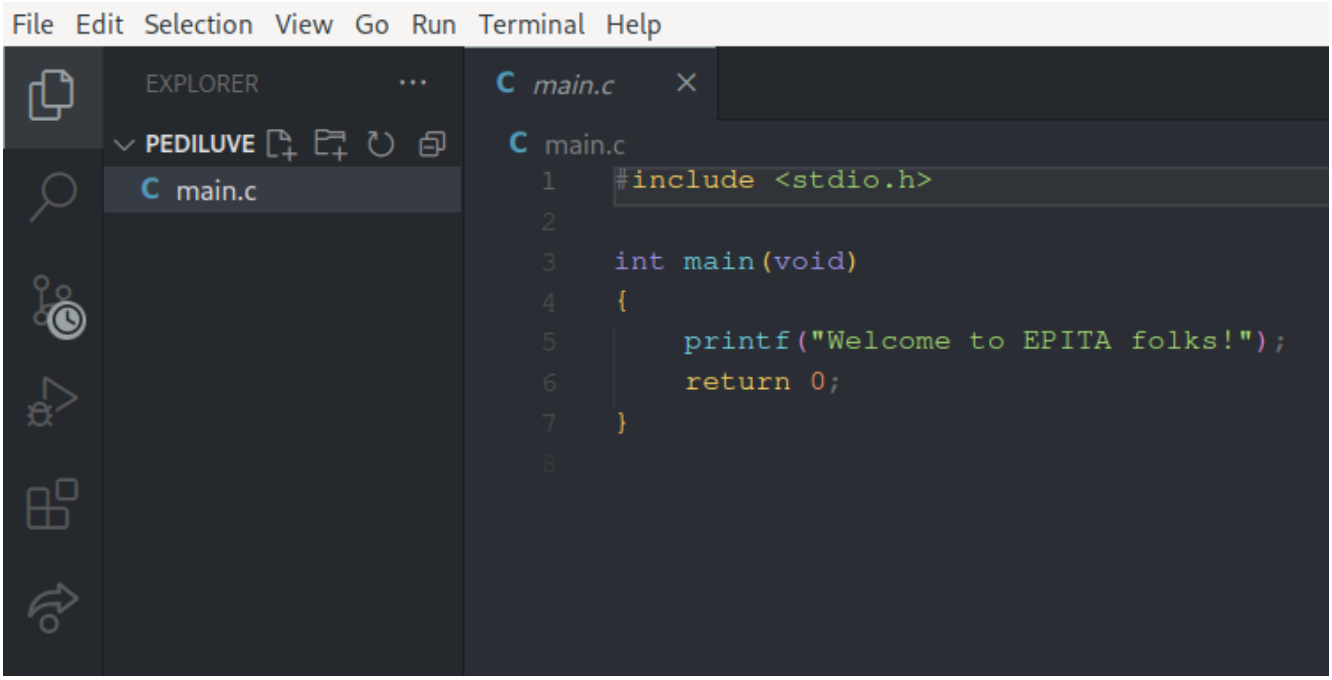
As you will only be allowed to use vim or Emacs during this year, try to be accustomed to using at least one of them.

[4] We are of course talking about Emacs commands, not shell commands.

### 4.5 Bonus: Code Editor and IDE

Even if you are only going to work with vim and emacs for this semester. it is still good to know that there is other kinds of editors and then understand why you shouldn't use them for now.

The two other big groups of editors are Code Editors and IDE (Integrated Development Environment), there purpose are mostly the same : helping you to write and edit code (this help can come in very various way: syntax check, static code analysis, debugging intergrate in the editor, etc.). They come with a bigger GUI, for example, here's what Vscode (one of the most popular code editor at the moment) looks like:




**Be careful!**

Now that you know that this kind of tool exists you can wonder why we **forbid** them for now:

- All exam sessions do not have Vscode installed on them, therefore you may only use either Vim or Emacs durng those exams.
- During your work as an engineer, you will not always have your perfect environment. So to be efficient in every situation you need to be able to use code tools first.
- The goal of this year is not only to make you learn how to code but also to make you learn the toolchain of a compilation. So if you already use tools that have everything already included, you will probably achieve what is asked from you but will not understand anything.
- We want you to discover shortcuts to make you more efficient. So even if GUI tools have shortcuts it does not force you to learn them.

# 5 Files and directories

In this section we will manipulate files and directories. The combination of the two, the way they are stored, and what you can do with them is called a file system. The file system behaves like a tree structure, the root directory being '/'.

You may have heard that "in UNIX, everything is a file", but in our systems, that is not a hard rule and more of a useful abstraction. For file systems, it means that you can manipulate all their objects as if they were files. Therefore, a directory can be manipulated just like a file, and you may encounter some documentation or other technical resources during your time at EPITA and after that use the word "file" to talk about any kind of file system entity, directory included. It is a bit confusing at first but it will make more sense once we cover some system programming topics later this year.

Every user in the system has a home directory. That is where you will put your projects and your personal configuration files. When you start a shell, your current working directory is automatically set to your home directory.

## 5.1 Exercise: Lumos

1. List the content of the `/etc` directory.
2. List the hidden files in your home directory. What makes a file hidden?

### 5.1.1 Exercise: parkour

1. Go to the `/var/log` directory, and list its content.
2. Go back to your home directory.
3. Go to the root directory: `/`, then go back to the directory you were in before, without typing its path.

At any point, remember you can check where you are by displaying the full path of your current working directory with `pwd`.

## 5.2 File and directory management

In this section, we will see how to create files and directories. Some commands will only work with files, others with directories, and a few can work on both.

Try to have at least a quick glance at the man pages of each command to have an idea of what they can do.

### 5.2.1 Exercise: creating

1. In your home directory, create a directory named `petit` and an other directory named `aculover`.
2. In the `petit` directory, create the following folder hierarchy using only one command: `poisson/nage/dans/la/piscine/`.
3. Create two regular empty files named `nemo` and `.tseT` in your home directory.
4. Create a temporary directory.

### 5.2.2 Exercise: copying

1. Copy the file named `.tseT` to the `aculover` directory.
2. Copy the `.tseT` file to the temporary directory you just created and rename it to `Test.` (all in one command). Is this file still hidden? What command would you use to confirm that?
3. Copy the whole `aculover` directory in the `petit` directory.

### 5.2.3 Exercise: moving

1. Move the `.tseT` file from your home directory to the `aculover` directory in your home directory. Find and use the option that prompts you for confirmation before overwriting.

### 5.2.4 Exercise: removing

**Be careful here: when you remove a file there is no way to get it back!**

1. Go to the `petit` directory and recursively remove all of its content (but not the `petit` folder itself).
2. Go back to your home directory and delete the `petit` directory without using the `rm` command.

## 5.3 Bonus: Filesystem basics

### 5.3.1 File System

File Systems (abbreviated FS) are the way data is stored and manipulated in a storage device. They define a way to organize data into similarly shaped pieces (called "files") and to group these files into "directories". You can see most file systems as a tree where nodes are directories and leaves are files.

Many different implementations exist with their pros and cons in terms of space optimization, usability and extra features such as journaling, encryption, replication, network access, etc. Here are some well known filesystems, some of which you will encounter naturally during your time at EPITA: FAT (FAT12, FAT16, FAT32), exFAT, NTFS, HFS and HFS+, HPFS, UFS, ext2, ext3, ext4, XFS, ZFS, Btrfs, AFS.

Confusingly, the expression "Unix filesystem" can also refers to a different but close concept. What we usually call the Unix filesystem is the way the file *hierarchy* (not *storage*) is organized on Unix systems, regardless of what underlying storage filesystem is used.

The school's computers run on a Unix-like system that loosely follows the Unix filesystem convention. For example, if you list the content of your your root folder (/), you should see something like this:

```
42sh$ ls -l /
total 60K
drwxr-xr-x   2 root root 4.0K Jun  3 15:23 bin/
drwxr-xr-x   7 root root 4.0K Jan  1  1970 boot/
drwxr-xr-x  22 root root 3.8K Jun  3 15:23 dev/
drwxr-xr-x   2 root root 4.0K Nov 15  2018 esp/
drwxr-xr-x  29 root root 4.0K Jun  3 15:23 etc/
drwxr-xr-x   3 root root 4.0K Aug  6  2021 home/
drwxr-xr-x   4 root root 4.0K Aug  6  2021 nix/
dr-xr-xr-x 240 root root    0 Jun  3 15:23 proc/
drwx------   6 root root 4.0K Jan  5 16:03 root/
drwxr-xr-x  24 root root  620 Jun  3 15:23 run/
drwxr-xr-x   2 root root 4.0K Aug  6  2021 srv/
dr-xr-xr-x  13 root root    0 Jun  3 15:23 sys/
drwxrwxrwt  28 root root  20K Jun  3 16:34 tmp/
drwxr-xr-x   3 root root 4.0K Aug  6  2021 usr/
drwxr-xr-x   9 root root 4.0K Apr  8 17:51 var/
```

Let us take a look at some of these directories and others that appear in common Unix systems:

- `/bin` usually contains the essential binaries like `pwd`, `echo`, `mkdir` and many others[1]
- `/sbin` is an extra folder that exists on some systems, it contains the system binaries (for administrators)
- `/usr` (for **U**nix **S**ystem **R**essources) contains some directories like the ones in `/` but that are not necessary to minimal usage of the system[2]. For example:
  - `/usr/bin`
  - `/usr/sbin`
- `/etc` (stands for **et c**etera, or more recently **e**ditable **t**ext **c**onfigurations) contains some system configuration files.
- `/tmp` contains temporary files, including those created by `mktemp`. On most systems, this directory is emptied when the system shuts down.
- `/var` contains **var**iable files. The main goal of this is to keep `/usr` read-only and do all customization in `/var`.
- `/nix` the folder that makes the school's system (NixOS) a bit different than the rest of Linux systems. Almost every system files and configuration actually reside in this folder. You will not see this folder on other Linux systems unless you also use NixOS or the program `nix`.

**Tips**

On most Unix and Linux systems (but sadly not on the school's computers), if you want to know

[1] it is a bit more complicated on the school's computers though and you should only see a file called `sh` (a shell) inside that folder. On most other Unix and Linux systems you should see a lot more files here.

[2] Here again, on the school's computers you should actually only see one folder (`bin`) containing one file (`env`), but a lot more things on any other Linux system.

more about the content of your system, you can type:

```
42sh$ man 7 hier
```

# 6 PIE

The **PIE**, meaning "Parc Informatique de l'EPITA", is the working environment you will come across on the school computers and you will use it for all your projects. It will allow you to:

- Do all your assigned work.
- Use all the languages you will be taught.
- Have access to a Linux operating system without having to install your own.

The PIE is under supervision of the **Forge** and usually includes every software and package you will need in your curriculum.

## 6.1 AFS

AFS is a distributed file system. Anything you need to know about it can be found in the CRI documentation. It is **essential** that you understand this notion and how to edit or create your configuration files.

**Be careful!**

Do not hesitate to go further and read each file in your `~/afs/` directory. **Remember that anything stored outside this directory will be lost at reboot.**

Forge —

The **Forge**, is an EPITA laboratory which takes care of:

- School Computer Equipment
- Working environment (**PIE**)
- Forge student accounts

### 6.1.1 Useful pages

The CRI Intranet can be used to perform the following:

- **Profile**: modify your EPITA account information (including password and SSH keys). You can access it by clicking on your login at the top right of the page.
- **Advanced search**: have a view on current EPITA students and workers. You can use `Groups` to help your search.
- **Documentation**: look up contact information and basic documentation.
- **Maps**: check all school maps on `Epimap`.

- **Moodle**: useful for your classes and exams.
- **Zeus**: look up your schedule.
- **SM charter**[1]: read the rules in place to access and use the campus resources.

### 6.1.2 Report an issue

Problems (non-booting PCs, broken mice, non-working screens, ...) will arise during the year and therefore, the number of computers available will drop. This can be smoothly overcome if these issues are reported quickly, allowing the Forge to solve them. So here is how you can make these known.

Concerning computer equipment issues, an issue on your Forge account or the PIE, you can contact the Forge by sending a ticket at **<tickets@cri.epita.fr>**.

**Going further...**

For computer equipment issues, be sure to use the tags **[PANNE][NAMEOFSM]**, where `NAMEOFSM` is replaced by the name of the SM the issue occured in.

# 7 The Assistants' portal

The portal is located at https://assistants.epita.fr, where you can find many useful links such as the Forge intranet, the Assistants' trove and other internal EPITA related websites.

## 7.1 The Forge intranet

The Forge intranet has many features. You will have to use them on a daily basis, so we advise you to get familiar with the them as soon as possible:

- *Documents*: the Assistants' coding style, the netiquette, various manuals and slides from conferences.
- *Projects*: list of the current and past projects, exercises, groups, and traces.

The Forge intranet is available at https://intra.forge.epita.fr.

## 7.2 The Assistants' trove

The trove stores various resources as well as tips for the programming languages and tools you will learn during your ING1.

The Assistants' trove is available at https://trove.assistants.epita.fr.

[1] SM stands for *Salles Machines*, French for *computer rooms*.

# 8 Communication 101

During the year, most computer science projects and workshops will be handled by the ACU (Assistants C/UNIX, often simply called “assistants”). Being able to communicate with the assistants even when you are not in their presence will be very important. Reasons for communication include keeping yourself up to date with the current project’s or workshop’s details, asking questions about a subject, or asking personal questions about your submissions or your intranet account.

These communications fall into two categories: information and questions that may concern all the students (like asking for clarification about the subject of an exercise) and personal enquiries (like asking why one of your submissions got rejected). Consequently, we use two different tools for these two types of communication: discourse forums (historically called “the news”) and email tickets.

## 8.1 Discourse forums

A lot of official information will be communicated by the assistants on the discourse forums, regularly consulting them is **mandatory**. It is also the main and standard way to communicate with the other students at EPITA. The forums can be accessed at the following address using your Forge credentials: https://news.epita.fr.

In these forums, students can post questions that will be publicly visible (for people at EPITA) and other students or assistants can answer them. There are a many forums, also called “categories”, and they are all dedicated to a specific subject.

You can find the most important forums on the left pane of the discourse website, and a complete list can be found by clicking the “Toutes les catégories” at the bottom of that left pane. For example, if you have a question about a current project, you can ask it on the related forum:

| **Forum** | **Description** |
|---|---|
| Test | Test forum |
| Assistants ING - Annonces | General information from the assistants |
| Assistants ING - Piscine | Questions and news about the piscine |
| Assistants ING - Harmonisation | Same for the harmonisation |
| Assistants ING - Projets | Same for the various projects |
| Forge - Annonces | General information from the Forge |

When you post a message on these forums, everyone in your promotion and the assistants will be able to see it and, more importantly, will be able to see the answers others give to it, including official answers from the assistants that may benefit all students. This is why we will encourage you to use this tool every time you have a question that other students may also have, so that the answer can be made available to everyone.

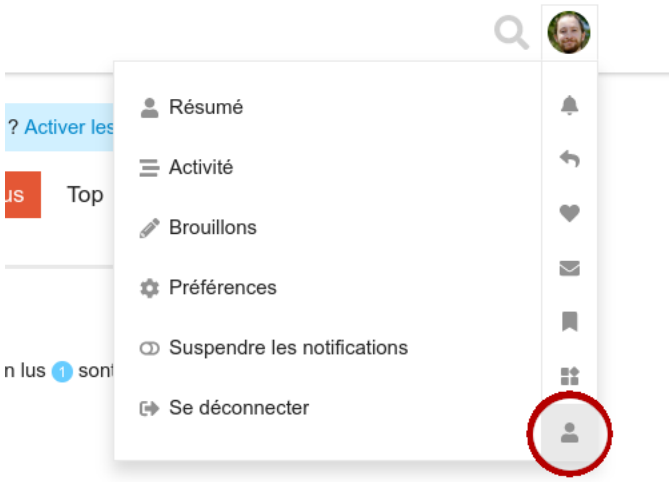
Also note that, for all official communication such as those happening on the discourse forums, we will require you adhere to a set of rules that will ensure a professionnal tone and a standardized presentation of your message. This set of rules is called the “Netiquette” and is available on the Assistants’ intranet at the following URL: https://intra.forge.epita.fr/epita-ing-assistants-acu/documents. If you want us to answer your questions, your messages **MUST** comply with the *Netiquette*.

**Be careful!**

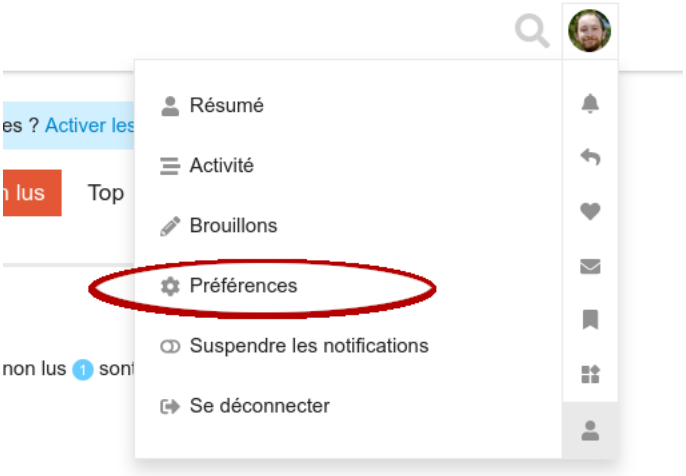
During the entire year, each message you exchange with the assistants (tickets, discourse post, etc.) will have to follow the Netiquette. If it does not, your question or request will be ignored and you will not get any answer. It can be a bit hard to get your messages right at first though, so please do not hesitate to ask the assistants to help you understand the Netiquette and how to make your messages comply with it. Also feel free to use the "Test" discourse forum to practice sending discourse posts, you will not get into any trouble for sending non-compliant posts there (and you will probably see assistants and teachers practice or test their configuration there too).

The default configuration for your discourse account should suit your needs for this year, but you can customize it if you want by following these steps:

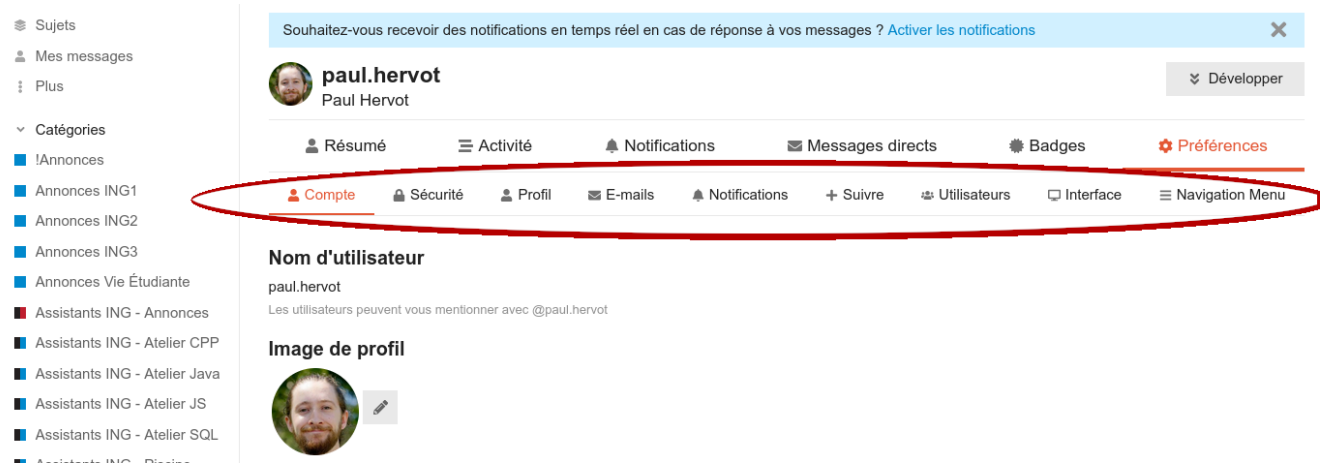
- click on your profile picture on the top-right;
- click on the character silhouette on the bottom of the drop-down menu that appeared:




- on the left, click on "Préférences"




You can then go through the different tabs to find the available customizable settings:




## 8.2 Email tickets

Emails are the main tool for solving private issues between the students and the Assistants. Any kind of problem can be reported and tracked using the ticket system. You must know how to write and send such emails. They can be written either in French or English.

Like on discourse, different email addresses are dedicated to different subjects, you will find a list of them on the "Contact" page of assistants' intranet: https://intra.forge.epita.fr/epita-ing-assistants-acu/contacts.

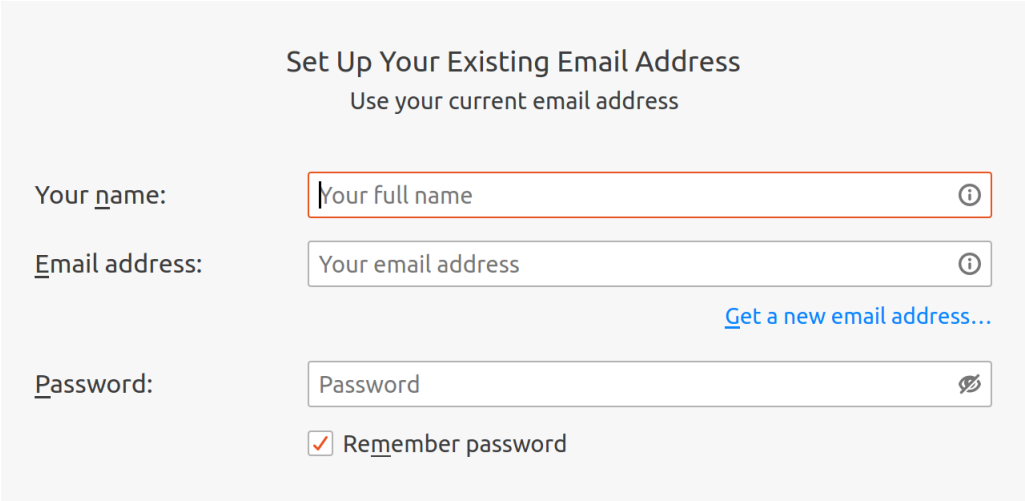
Like on discourse, every email you send to those addresses must follow the netiquette. Please carefully read the netiquette document, you will notice that the rules for tickets are a bit different to the rules for discourse forums (although very similar in spirit).

## 8.3 Thunderbird

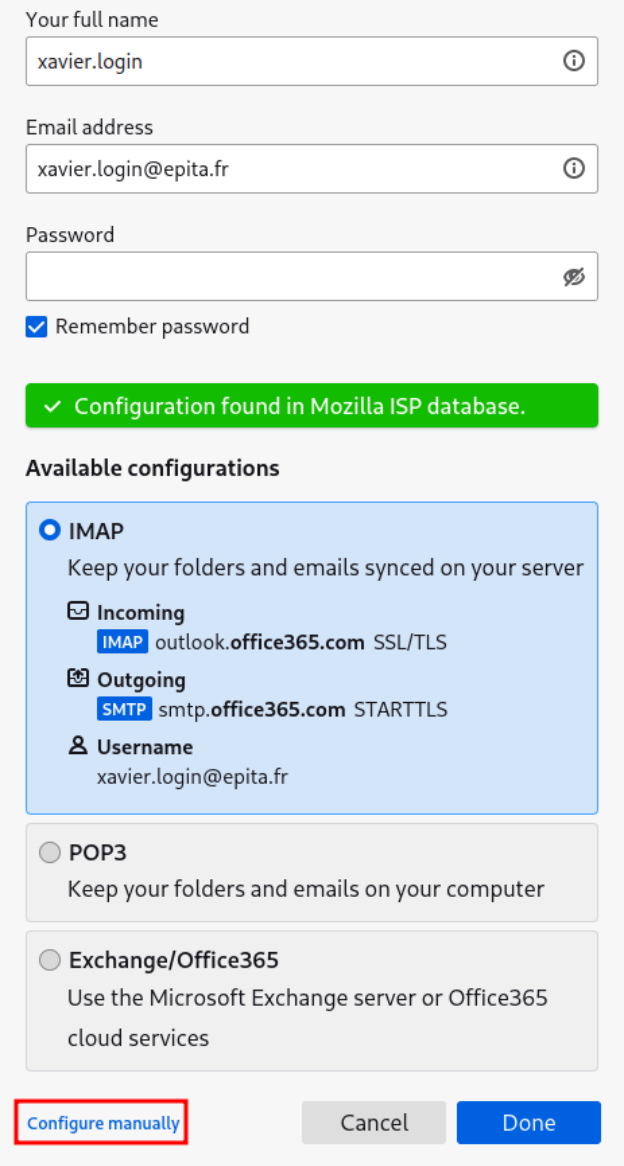
Although you can use Outlook from Microsoft 365 (presented later in this tutorial) to send emails and thus tickets, it can be very hard to comply to the netiquette with its interface. To help with this, we invite you to use Thunderbird, an email client available on the school's computers. In this section, we show you how to configure Thunderbird to link it with your EPITA email address.

First launch Thunderbird (you can use the dmenu for that!). If it is your first time, you will be asked to setup your email account. Fill in the required information:

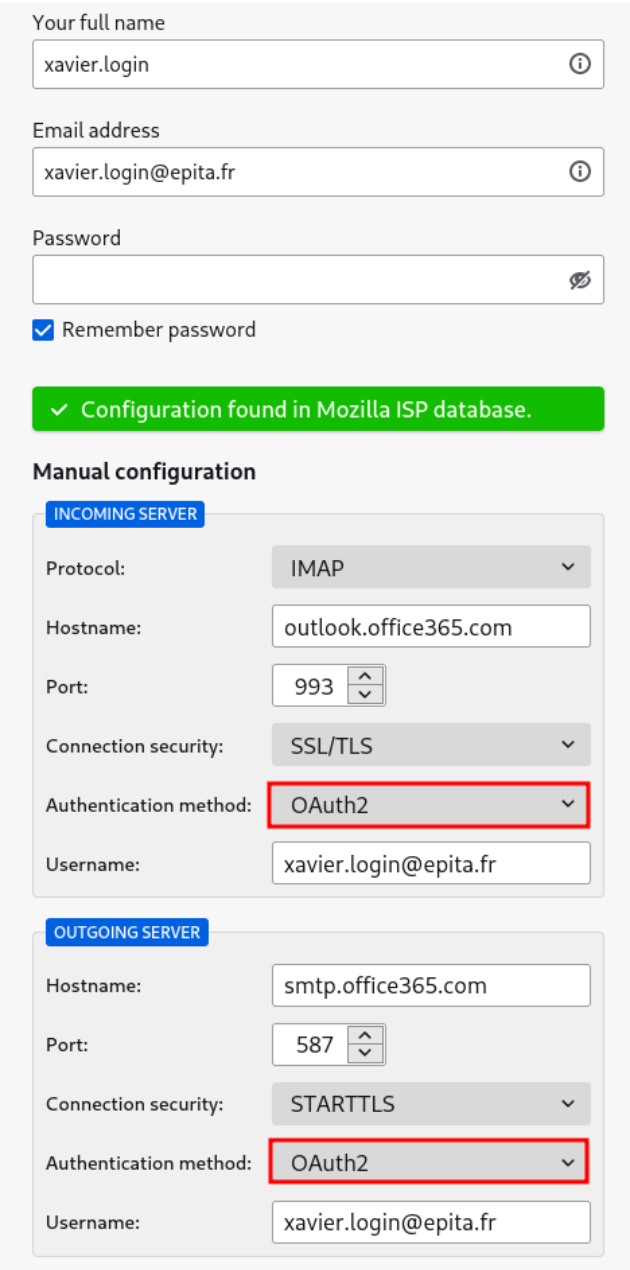
- **Your Name** should be your full name.
- **Email address** should be your Epita email address.
- **Password** should be your Epita email account's password.




Press on `Continue` and wait until the button becomes `Done`. Now press `Configure manually` at the bottom left.



Then select `OAuth2` as Authentication method for Incoming Server **AND** Outgoing Server.




Press the `Done` button. A new window should open, login with your Bocal account. Login with MFA, and then click “Authorize” if asked.

Congratulations, you have successfully setup your email client!

If you did not setup correctly you can still follow the Forge tutorial.

Before sending your first ticket, you will have to deactivate the option that composes your emails in *HTML*. Plain text emails are the only accepted format when writing to the assistants.

To do so, click on your email inbox in the left navigation pane, then click on the `Edit` menu, then `Account settings`. Under your email account settings, click on `Composition & Addressing` and uncheck the `Compose messages in HTML format` box.

**Be careful!**

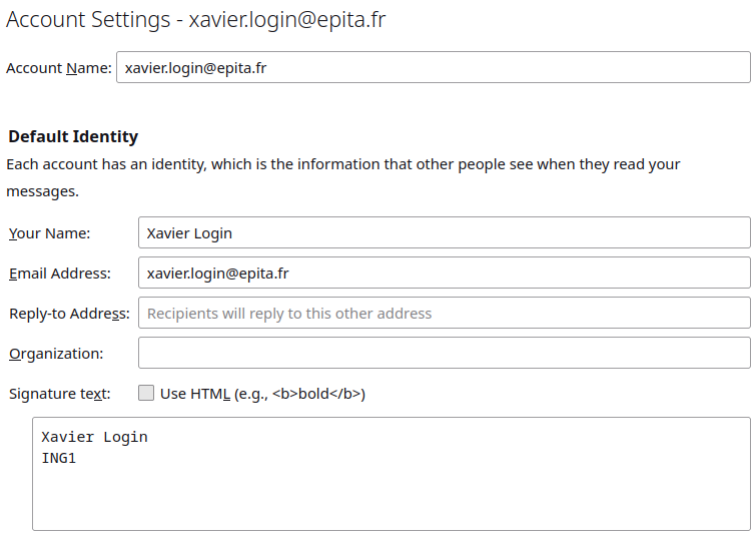
Thunderbird will ask you if you want to download your emails, as you do not have much space on your AFS, try not to download too many emails. You can go to `Account Settings`, then go to `Synchronization & Storage`, in the section `Disk Space` you can manage how many emails you want to keep on your AFS.

### 8.3.1 Signature

To create and add your signature so that it is automatically included in every email, click on your email inbox, then on `Account settings`. In the `Signature text` box, you can write your own signature. Remember that it **MUST** conform to the Netiquette available on the Assistants' Intranet.

**Tips**

If you look at the Netiquette you will see that your signature has to begin with two "-" and a space. You do not need to worry about it here however, as Thunderbird will add it for you automatically every time you start writing a new email.




From now on, every email you will send from Thunderbird will be signed with the content of your signature.

**Tips**

If you want to access your school's email through another email client, here are some configurations it may ask you:

- Imap: outlook.office365.com
- Port: 993 SSL
- Smtp: smtp.office365.com
- Port 587 STARTTLS
- Login: firstname.name@epita.fr

- Password : “password” for your email address, given by the BOCAL at the beginning of your schooling at EPITA.

## 8.4 Exercises

Let us make sure you are able to use these communication tools properly. Make sure you successfully complete these two exercises, **your grade in this lesson will fail if you do not**. We don’t expect you to get everything right on the first try though, and it is perfectly fine if it takes a few days before you get a hang of these tools and the netiquette, but do try to do them now.

### 8.4.1 Discourse

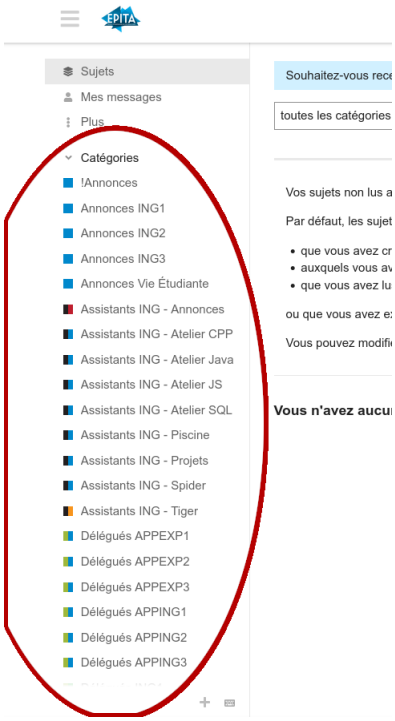
First, write a post in the `Assistants ING - Piscine` discourse forum. The post must conform to the Netiquette available on the Assistants’ Intranet and its subject must start with:

```
[PEDI][SUBJECT] xavier.login: my first day
```

The two tags at the beginning allow everyone to immediately now what this post is about. You must replace `xavier.login` by your own login (we will often use this fake login as a placeholder: when you see it in future subjects, know that you need to replace it with your own).

In the content of your post, you will tell us about your day and what you are expecting from the next one. Do not forget, you **MUST** follow the Netiquette when posting on the discourse forums!

To create that post, find the correct forum (category) on discourse’s left pane (if you can’t find it, scroll down that pane to find the “Toutes les catégories” button, which will send you to a more complete list):




Then click the “Créer un sujet” button:


A new pane should appear at the bottom where you can write your post's title and body. When you are done writing, confirm by clicking the blue "Créer un sujet" button at the bottom.

**Tips**

You will probably have to make a few tries before getting it right. You will notice that discourse refuses to create a topic that has the same subject as a previous one. Since the subject of your topic must only **start** with what we told you above, feel free to add precisions like `v2` or `second try` at the end of it so that you can post again.

### 8.4.2 Tickets

For this exercise you must send a ticket to `test@tickets.assistants.epita.fr`, with the following email subject:

- Subject: [PEDI][SUBJECT] xavier.login: my first day
- Body: `J'aime les ACUs, c'est de la folie !`
- A signature

**Be careful!**

Do not forget:

- to place greetings and salutations around the body;
- to replace `xavier.login` with your login;
- and once again, to **follow the netiquette**.

You can find the most common tickets addresses in the `contact` section of the Forge's intranet.

**Tips**

In your career, you will receive many emails about the same projects or events. If you want to quickly find emails with a particular subject or sent by or to a specific person you can use the `Quick Filter` options from thunderbird's top bar.

If you prefer to sort your emails in the long run, you can also check the menu at the right corner of the top bar, and then select `Tools`. From there you can click on `Message Filters` then create

and manage folders where your emails will be automatically moved to, or various tags and status you can mark your emails with.

Using proper tags in the subject of articles and emails (like `[PEDI][D1]`) makes it very easy to setup message filters that sort articles and emails in neat categories, which is why those tags are mandatory for all communication involving the assistants. Every project this year will have a tag you must use when writing emails or news articles related to it, it will always be indicated on the assistants' intranet; see if you can find the one for this lesson!

## 8.5 The assistants' duty periods ("permanences")

During this year you will have many workshops and projects to work on. To help you complete them, the assistants will give various conferences about the technologies and development techniques relevant to each of them, but they will also be able to help you individually during some dedicated time periods. These times are called duty periods (or "permanences" in french). They will be announced ahead of time (on discourse!) and dedicated to help you on a specific project or workshop.

During these duty periods, you can ask an assistant for help on any problem you are personally struggling with regarding the project. You can for example ask for help fixing a bug you are stuck with, request guidance on how to organize your code, or ask for advice on how to start a particular task if you are out of ideas.

Some of these duty periods will be "on site", meaning the assistants will be physically present in the computer rooms, available to help you. But other periods will be completely "remote": the assistants will be available through a Discord server via private text and voice channels. This is particularly important for students outside of the Paris campus, for whom the assistants will not be on site as often.

### 8.5.1 The Discord server

To make sure you can receive this kind of help, let us set up your access to this Discord server and the bot it uses.

**Tips**

If you do not already have a Discord account, you can create one by visiting its official website: https://discord.com/ and clicking "open Discord in your browser" (or download and install the app, then following the instructions it will give you).

If you already have a Discord account, but do not want to use it for communicating with the assistants, feel free to create another account. To do so, open Discord (the desktop application or the website work the same), click your profile picture on the bottom left of the interface, then "switch accounts", then "manage accounts", then "add an account".

The assistants and the school will not reach out to you themselves through this Discord server, so you do not have to worry about missing notifications (the notifications you **should** worry about are your school's email address and the discourse forums), you only need to make sure you can connect to this server and use the bot to ask the assistants for help when you need it.

Here is an invitation link for the Discord server dedicated to the duty periods : https://discord.gg/3kgvJBDRwG. If this link does not work anymore at the time you are reading this, another one should

have been given on discourse. Like always, be sure to read carefully everything that is posted on the assistants categories of discourse.

Once you are connected to the server, you will need to authenticate your discord account to the Forge. Simply follow the few instructions in the `#accueil` channel of the server to do so.

Once you are authenticated, you should be granted access to a few other channels. Most importantly, you should see a bot called "Sandworm". To request the help of an assistant during a duty period, you should not ask your question on a public channel, but instead send a command to this bot. A command is a simple private message that follows the format given on the Forge's documentation. For example, if you need help on an exercise in the XYZ lesson, you can send the following private message to Sandworm: "`!request [XYZ] I don't understand why my discourse post was rejected`" (you can also write the question in french). If there indeed is a duty period linked to this tag at that time, the bot will notify the assistants that you need help. After a (hopefully) short time, you will be invited to a private channel where you can discuss you problem with an assistant (by voice or text).

# 9 SSH

`SSH` (short for *Secure SHell*) is a network protocol that provides a secure connection. It uses a client-server architecture, thus the connection is established by an SSH client that wants to connect to an SSH server. Once the connection is established, the machine can be controlled from a distance, giving access to informations and services.

To initiate a connection process, SSH is based on the principle of public key cryptography and asymmetric encryption. It uses two mathematically related keys: a public key and a private key.

**Tips**

You can read more about it by taking a look at `ssh(1)`.

## 9.1 Use

In the present day, SSH is mostly used for:

- Remote access: it ensures an encrypted remote connection.
- File transfers: it provides a safe way to manipulate files over a network, using the Secure Copy Protocol (SCP) or SSH File Transfer Protocol (SFTP).
- Tunneling: it provides secure data transfers from one network to another.

During this semester, you will also use your SSH keys to connect to the server containing the `Git` repositories. We will learn more about them in the following tutorial.

### 9.2 Key generation

You may follow this tutorial to generate your SSH key.

**Be careful!**

If anyone asks for your private key, **do not** give it to them.

**Be careful!**

If you are not careful, SSH keys **can be stolen** (if you leave your computer unlocked for example). In this case, the thief could try to impersonate you using your key to steal, modify, or delete your work. The key's passphrase is your last protection against that kind of people.

## 10 Documentation and help

### 10.1 man

`man` is probably the command you will have to use the most this semester. As its name suggests, it gives you access to pages of your system's reference manuals, which concerns most programs you will interact with.

Using this command must therefore be a first reflex when your are in doubt. The Assistants will not always be there to help you. To make your life easier, always read the manual page associated with what troubles you[1].

**Tips**

All the man pages are available during exams as they do not require an internet access.

The man pages cover a wide range of subjects: shell commands, *C* functions or even general notions. Since these subjects are so varied and there are so many things documented, pages are sorted in sections of common topic generally designated by a number. The word `printf` for example refers to a `C` function, but also to a shell command, both have their own man page which reside in different sections. If you want to see the man page of the C function `printf`, you will have to tell `man` "I want the page for `printf` in the C function section", the syntax to do so is: `man [section number] topic`.

The list of generally available sections is:

- 1: Executable programs or shell commands
- 2: System calls (functions provided by the kernel)
- 3: Library calls (functions within program libraries)
- 4: Special files (usually found in `/dev`)
- 5: File formats and conventions e.g. `/etc/passwd`
- 6: Games

[1] Usually synthesized as *Read The Fucking Manual*, or *RTFM*, this kind of remark is used in the IT community if you ask a question whose answer is in the documentation/manual. The idea is to always take a look at the documentation before asking people for their time.

- 7: Miscellaneous
- 8: System administration commands (usually only for root)
- 9: Kernel routines [Non standard]

If you do not specify a section, `man` will search through all the sections in a default order[2]. For example `man chmod` is equivalent to `man 1 chmod`, which is a different page from `man 2 chmod`.

**Tips**

In written conversations and documentation with other humans, when we talk about different things that have the same name, we like to avoid any ambiguity by mentioning in parentheses the section of the man corresponding to the specific thing we are talking about. This is a convention we will follow in these tutorials, so if you see us writing about `chmod(1)` for example, you should understand that we are talking about "the thing from the section 1 of the man called `chmod`" or more concisely, the *program* `chmod`, whereas if we write `chmod(2)`, you should understand that we are talking about the *system call* named `chmod` (we will see what a system call is later this year, just know that it is very different from a program).

Lastly, as many terminal-based tools you will use this year, there are few keyboard shortcuts you should know to use `man`:

- `q` will close the page and return to your shell
- `/` will allow you to search some text inside the page
  - `n` **during a search will jump to the next occurrence of your searched** string
  - `N` **during a search will jump to the previous occurrence of your** searched string
- the `up` and `down` arrow keys will scroll the page line by line, the `j` and `k` keys will do the same (we call these vi-style keybindings, as they immitate bindings from the vi editor, using these is easier on the wrists)
- `page-up` and `page-down` will scroll the page by large portions the size your screen (`space` does the same thing as `page-down`)

From now on, for each new command you encounter (and you will encounter quite a few of them in this document), you **must** have the reflex to look it up in the man pages.

**Going further...**

Know that there is another type of documentation available offline on your computer:

- **the info documents:**
  These are documents sometimes a lot more detailed than man pages about various topics and commands of your system. It is less important for now so concentrate on the other commands, but you can try learning about it if you want to go further (be careful, the keyboard shortcuts are not the same as man's).

[2] To find this default order, look at the man page of `man` itself. Close to the end of this page you will see the path of file described as the "man-db configuration file", the order is specified in the `SECTION` directive of that file. do not try to type the file's path yourself to open it, use the copy and paste techniques we showed you!.

- **the help options:**
  For most shell programs, if you just want to quickly know what the program does and what the common options are, you can type:

```
42sh$ <program> -h or --help
```

## 10.2 info, whatis, apropos

In your endless search for answers and documentation, there are three other commands you can use to help you navigate the man pages:

- `info topic` reads documentation in Info format with a CLI.
- `whatis topic` displays all the sections that has a page called `topic`, along with a short description of what `topic` means in that section.
- `apropos keyword` searches in the name and description of all man pages for the keyword you are looking for and tells you which pages mention your keyword. It is very useful when you are searching for a command you know exists but have forgotten the name.

```
42sh$ whatis info
info (1)              - read Info documents
info (1ssl)           - OpenSSL application commands
info (5)              - readable online documentation
42sh$ whatis whatis
whatis (1)            - display one-line manual page descriptions
42sh$ apropos apropos
apropos (1)           - search the manual page names and descriptions
42sh$ apropos echo
echo (1)              - display a line of text
echo (3x)             - curses input options
echo_sp (3x)          - curses screen-pointer extension
echo_wchar (3x)       - add a complex character and rendition to a curses window, then advance␣
↪the cursor
echochar (3x)         - add a character (with attributes) to a curses window, then advance the␣
↪cursor
gifecho (1)           - generate a GIF from ASCII text
l2ping (1)            - Send L2CAP echo request and receive answer
lessecho (1)          - expand metacharacters
noecho (3x)           - curses input options
noecho_sp (3x)        - curses screen-pointer extension
pam_echo (8)          - PAM module for printing text messages
pecho_wchar (3x)      - create and display curses pads
pechochar (3x)        - create and display curses pads
ping (8)              - send ICMP ECHO_REQUEST to network hosts
tracefs_hist_echo_cmd (3) - Update and describe an event histogram
tracefs_synth_echo_cmd (3) - Retrieve data of synthetic events.
wecho_wchar (3x)      - add a complex character and rendition to a curses window, then advance␣
↪the cursor
wechochar (3x)        - add a character (with attributes) to a curses window, then advance the␣
↪cursor
```

**Tips**

Try typing `info info` in your terminal and browse the documentation.

### 10.3 Exercise: `man` and `whatis`

1. To become more familiar with a command, nothing works better than reading its man page, even if it seems really abrupt, it becomes easier to find what you need with time and practice. Let us successively use `man` and `whatis` to discover the following commands:

**Tips**

Do not only read man pages and descriptions.

You have time, so try to run `man` and `whatis` on each of these commands by yourself.

- `whatis`
- `apropos`
- `man`
- `cat`
- `echo`
- `time`
- `type`
- `stat`
- `setxkbmap`

2. What are the differences between `whatis` and `man`?

### 10.4 Exercise: `apropos`

If you do not know the exact name of a command you are looking for, you can use the `apropos` command. First, read its man page to see how to specify a section to look into, then try it by searching the following keywords in the **shell commands** section:

- `copy`
- `rename`
- `terminal`
- `editor`
- `shell`

# 11 Numeral system

## 11.1 Concept

Numeral systems are ways to represent numbers with symbols. Other than having a unique representation per number or defining sets of numbers (integers, rationals...), a numeral system has a finite amount of symbols, which define a base.

For a base $n$ we have (with $a$, $b$, $c$, $d$ and $e$ belonging to the symbols of the base):

| ... | $n^4$ | $n^3$ | $n^2$ | $n^1$ | $n^0$ |
|---|---|---|---|---|---|
| ... | $e$ | $d$ | $c$ | $b$ | $a$ |

In the end, the value of a number $x$ represented in base $n$ can be found:

```
x = a * n^0 + b * n^1 + c * n^2 + ...
```

We mostly use base 10, with digits (Hindu-Arabic numerals). Keep in mind that each base-$n$ systems is just a representation. In fact, each and every number expressed in a base can be expressed in another base as well.

## 11.2 Binary

Only two symbols: 1 and 0. The values 1 and 0 are thus the maximal and minimal values you can get with a single digit in this numeral system. Represents perfectly the on and off states in which a transistor can be.

Let's take the example of 42. To get the binary form of 42, we try to decompose it with powers of 2.

```
42 = 32 + 8 + 2
42 = 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0
```

So 42 is 101010 in binary.

However with big numbers this method does not work well. You can use euclidean division to find the same results and with no knowledge of powers of 2:

```
42 = 2 * 21 + 0
21 = 2 * 10 + 1
10 = 2 * 5  + 0
5  = 2 * 2  + 1
2  = 2 * 1  + 0
1  = 2 * 0  + 1
```

When reading the remainders from the bottom to the top, you get 101010 which is the correct form of 42 in binary.

Now you know how to go from decimal to binary. Let's try doing the opposite. Try to understand why 1001 in binary is equal to 9 in decimal.

As you probably guessed,

```
1 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 1 * 8 + 1 * 1 = 9
```

You can try computing the examples below by yourself.

```
101010                --> 42
1100 + 100 = 10000    --> 12 + 4 = 16
11111111              --> 255
100000000             --> 256
```

## 11.3 Octal

Eight symbols: 0 to 7. The values 7 and 0 are thus the maximal and minimal values you can get with a single digit in this numeral system.

Just like you did with the binary, you can use euclidean division to find the octal form of a number.

```
42 = 8 * 5 + 2
5  = 8 * 0 + 5
```

When reading the remainders from the bottom to the top, you get 52 which is the correct form of 42 in octal.

Now you know how to go from decimal to octal. Let's try doing the opposite. Try to understand why 112 in octal is equal to 74 in decimal.

As you probably guessed,

```
1 * 8^2 + 1 * 8^1 + 2 * 8^0 = 64 + 8 + 2 = 74
```

You can try computing the examples below by yourself.

```
14 + 4 = 20           --> 12 + 4 = 16
777                   --> 511
1000                  --> 512
```

## 11.4 Hexadecimal

Sixteen symbols: 0 to 9 then a to f. The values f and 0 are thus the maximal and minimal values you can get with a single symbol in this numeral system.

Just like you did with the binary and the octal, you can use euclidean division to find the hexadecimal form of a number.

```
42 = 16 * 2 + 10
2  = 16 * 0 + 2
```

When reading the remainders from the bottom to the top, you get 2 and 10. The 10 becomes an 'a'. So the correct hexadecimal form of 42 in hexadecimal is 2a.

Now you know how to go from decimal to hexadecimal. Let's try doing the opposite. Try to understand why CAFE in hexadecimal is equal to 51966 in decimal.

As you probably guessed,

```
C           A           F           E
12 * 16^3 + 10 * 16^2 + 15 * 16^1 + 14 * 16^0 = 49152 + 2560 + 240 + 14 = 51966
```

You can try computing the examples below by yourself.

```
c + 4 = 10            --> 12 + 4 = 16
1ff                   --> 511
200                   --> 512
```

# 12 Access control

## 12.1 Concept

Access control is a notion attached to every file. For each file[1], there is a list defining the individuals and their authorized actions.

In this section, we will look at the basic access control mechanism.

## 12.2 Ownership

For each file, it is possible to define three different sets of permissions:

- Permissions for the user owning the file: For those you created, you are the owner.
- Permissions for the group owning the file: Each user can be in multiple groups including one default group. Files you created are given to your default group[2].
- Permission for others, i.e. anybody that is neither the owner nor in the owning group.

It is possible to see the permissions associated with a file using the command `ls -l`. Let us take a closer look at an example:

```
42sh$ ls -l testsuite
-rwxr-xr-- 1 xavier.login students 26K Aug 29 23:42 testsuite
```

Here, the file is owned by the user *xavier.login* and the *students* group.

[1] And thus directory, since everything is a file in a UNIX system.
[2] You can see which groups you belong to using the command `groups`.

## 12.3 Permissions

Linux offers three types of permissions on a file, each of them being associated with a letter:

- **r**eading, noted '`r`'
- **w**riting, noted '`w`'
- e**x**ecuting, noted '`x`'

When the permissions of a file are represented, the three categories of users and their associated rights are displayed in this order: *owning user, owning group, others*.

For instance, when `ls` displays '`rwxr-xr--`' (like in the last example) we can analyze 3 types of rights:

| `rwx` | `r-x` | `r--` |
|---|---|---|
| Owner permissions | Group permissions | Others permissions |

- The first three characters show the permissions of the user that owns the file
- The three middle characters show the permissions of the users that are in the file group
- The last three characters show the permissions of the users that are neither the file owner nor in the file group (others)

Thus, here the *xavier.login* user has *read*, *write* and *execute* permissions on the file. Users in the *students* group have *read* and *execute* permissions and all other users have *read* permissions only.

**Tips**

The '–' character means that the permission is not present.

**Going further...**

UNIX permissions are strictly inclusive. This means if a user owns a file and is in the group owning the file but the file permissions are `---rwx---`, the user will not be able to do anything with this file because his owner rights overrule his group rights.

## 12.4 Octal representation

Each right is associated with a numeric value:

- **r**eading, noted '`r`' is value 4
- **w**riting, noted '`w`' is value 2
- e**x**ecuting, noted '`x`' is value 1

As you may have noticed, the numeric values correspond to powers of 2:

```
value : 4 2 1
letter: r w x
```

Thus, you can write a permission on 1 byte by simply adding all the values. For instance '`r-x`' has value `4 + 0 + 1 = 5`.

Here are all the combinations of permissions and their numerical representation:

| | | |
|---|---|---|
| 0 | `---` | no permissions |
| 1 | `--x` | execute permission (for a file: right to execute, for a directory: right to enter and search the directory) |
| 2 | `-w-` | write permission |
| 3 | `-wx` | write and execute permission |
| 4 | `r--` | read permission |
| 5 | `r-x` | read and execute permission |
| 6 | `rw-` | read and write permission |
| 7 | `rwx` | all three permissions |

**Going further...**

For your information, `ls -l` does not only supply permissions information.

```
42sh$ ls -l testsuite
-rwxr-xr-- 1 xavier.login students 26K Aug 29 23:42 testsuite
```

| | | |
|---|---|---|
| 1 | `-rwxr-xr--` | Permissions |
| 2 | `1` | Number of entries physically attached to the entry |
| 3 | `xavier.login` | User owning the entry |
| 4 | `students` | Group of users owning the entry |
| 5 | `26K` | Approximated entry size |
| 6 | `Aug 29 23:42` | Last modification date |
| 7 | `testsuite` | Name of the entry |

The first character in `Permissions` may be a `d` if the file is a directory or – else.

To understand the second information, `Number of entries physically attached to the entry`, take a look at the man page of `link(2)` and `symlink(2)` or in the glossary.

## 12.5 `chmod`

The `chmod` command changes the permissions associated with a file. `chmod` means *CHange MODe*. This command accepts two representations of the permissions.

### 12.5.1 Octal representation

As you have seen above, it is possible to specify the read-write-execute permissions as a numeric value. By concatenating the values obtained for the user, the group and the others, we get a three digits number that represents the permission in the octal base.

For example, the `rwxr-xr-x` permissions in octal representation is `755`, `rw-r-----` is `640`.

`chmod` allows this representation to specify the new permissions to be applied to a file. The syntax is straightforward, first the permissions then the file(s) to which they are applied.

For example, to give the 640 permissions to the file named not_executable, you will use:

```
42sh$ chmod 640 not_executable
```

### 12.5.2 Symbolic representation

chmod also accepts a different and more explicit notation: the symbolic representation. The attribution of permissions is defined using symbols, split in two kinds:

1. Permission symbols:
    - 'r' for the *Read* permission;
    - 'w' for the *Write* permission;
    - 'x' for the *eXecute* permission.
2. Group symbols:
    - 'u' for the owning *User*;
    - 'g' for the owning *Group*;
    - 'o' for the *Others*.

We can specify if we want to add or remove the permissions using operators:

- '+' to *give* the permissions;
- '–' to *take back* the permissions.

To modify multiple groups at the same time using the symbolic notation, it is possible to combine sequences of symbols separated by commas. For example, you can give the 640 octal mode using:

```
42sh$ chmod u+rw,u-x,g+r,g-wx,o-rwx not_executable
```

**Going further...**

AFS augments and refines the standard UNIX scheme for controlling access to files and directories. For example, in your afs directory, you can view the permissions of a directory with:

```
42sh$ ls -la my_directory
```

If you have time, you can check this link: https://docs.openafs.org/UserGuide/HDRWQ44.html#HDRWQ45

**Going further...**

You might also want to take a look at the chown man page to see how to change the *owner* of a file.

# 13 The Right Tarball

## 13.1 Goal

A tarball is often used for **backups** or **releases** purposes. In a release, you generally find information about the author(s) of the project, some documentation, and source code. Let us make one.

Generate a tarball with a specific architecture and permissions.

Start by creating a folder `my_tarball` at the root of your exercise directory.

## 13.2 File handling and permissions

This section must be done inside the `my_tarball` folder.

To check that you have understood notions about how to handle files, directories and their associated permissions correctly, you must:

- create an `AUTHORS` file and modify its permissions. An `AUTHORS` file contains any information about the author(s) of a project. It must have the `640` mode and must not be empty. The content is irrelevant.
- create a new directory named `doc`. It must have the `700` mode.

**Tips**

A conventional `AUTHORS` file contains a line that can easily be parsed:

```
42sh$ cat -e AUTHORS
* prenom.nom$
```

**Tips**

If the octal permission system is still unclear to you, take time to read the chmod manual.

Once this is done, you must create a text file:

- This file must be named `rights`, located in the `doc` directory you just created, and have the `640` access rights.
- It should be at least 5 lines long.
- The text must be wrapped at 80 characters, meaning no line should exceed 80 columns.

If you lack imagination, you can write about the permission system and the different commands you can use to manipulate them.

Finally, you must create an `src` directory containing a script:

- The `src` directory must have the `755` mode.
- The script must be located in the `src` directory and named `authors.sh` with the `644` access rights - thus, not executable.

The script will handle the **creation** of an `AUTHORS` file with **custom permissions** in the current working directory (the directory in which you are when you run the script). The created file must have `640` access rights.

### 13.3 Assignment tarball

To validate this exercise, you must compress your previous work in a tarball and submit it.

Your tarball must be called `my_tarball.tar.gz`. This file must be a tar archive compressed in the **gzip** format. Take time to read the manual of `tar` to find out how to create this archive.

The tarball must contain the directory `my_tarball` and all its content.

**Be careful!**

Do not include any trash or git related file.

You can check that your submission is correct by comparing it with this output:

```
42sh$ tar tf my_tarball.tar.gz
my_tarball/
my_tarball/AUTHORS
my_tarball/doc/
my_tarball/doc/rights
my_tarball/src/
my_tarball/src/authors.sh
```

**Tips**

Using the verbose option can be helpful when debugging your tarball.

## 14 Permission101

### 14.1 Goal

In this exercise you will have to deal with permissions.

### 14.2 What am I supposed to do?

You must create folders and files according to the below architecture:

```
42sh$ tar -tvf permission101.tar.gz
drwxr-xr-x  xavier.login/epita_2021 2017-09-10 23:00 permission101/

drwxrwx---  xavier.login/epita_2021 2017-09-10 23:00 permission101/group/
-r--r-----  xavier.login/epita_2021 2017-09-10 23:00 permission101/group/first
-r--r-x---  xavier.login/epita_2021 2017-09-10 23:00 permission101/group/second

drwx---rwx  xavier.login/epita_2021 2017-09-10 23:00 permission101/others/
```

(continues on next page)

(continued from previous page)

```
-r-----rwx  xavier.login/epita_2021 2017-09-10 23:00 permission101/others/first
-r-----r-x  xavier.login/epita_2021 2017-09-10 23:00 permission101/others/second
-r-x---rw-  xavier.login/epita_2021 2017-09-10 23:00 permission101/others/third

drwx------  xavier.login/epita_2021 2017-09-10 23:00 permission101/user/
-rwx------  xavier.login/epita_2021 2017-09-10 23:00 permission101/user/first
-rw-------  xavier.login/epita_2021 2017-09-10 23:00 permission101/user/second
```

**Going further...**

Names are case sensitive and files can in fact be empty.

## 15 Git Workshop

You can now check the `Git Workshop` activity on the intranet and do the available exercises.

*I must not fear. Fear is the mind-killer.*