



PEDILUVE — Tutorial D1

version #0



**The way is lit. The path is clear.
We require only the strength to follow it.**

Copyright

This document is for internal use at EPITA ([website](#)) only.

Copyright © 2022-2023 Assistants [<assistants@tickets.assistants.epita.fr>](mailto:assistants@tickets.assistants.epita.fr)

The use of this document must abide by the following rules:

- ▷ You downloaded it from the assistants' intranet.*
- ▷ This document is strictly personal and must **not** be passed onto some-one else.
- ▷ Non-compliance with these rules can lead to severe sanctions.

Contents

1	Foreword	4
1.1	Useful commands to get started	4
2	PDF viewer	4
3	Dmenu	5
4	Terminal and shell	5
4.1	What is the shell?	5
4.2	Practice time	6
4.3	Bonus: keyboard shortcuts for your shell	8
4.4	Bonus: quick history of existing shells	9
5	Editors	10
5.1	Vim	10
5.2	Emacs	12
5.3	Exercise	13
5.4	Do it yourself	13
5.5	Bonus: Code Editor and IDE	13
6	Files and directories	14
6.1	pwd(1)	15
6.2	ls(1)	15
6.3	Exercise: Lumos	15
6.4	cd(1)	16
6.5	File and directory management	17
6.6	Bonus: Filesystem basics	20
7	Window managers and i3	22
7.1	i3lock	22

*<https://intra.assistants.epita.fr>

7.2	Exercises	23
8	PIE	23
8.1	AFS	23
8.2	CRI	23
9	The Assistants' portal	24
9.1	The Assistants' intranet	24
9.2	The Assistants' trove	25
10	Communication 101	25
10.1	Newsreaders	25
10.2	Tickets	33
11	SSH	34
11.1	Use	34
11.2	Key generation	34
12	Exercises	35

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.

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 PDF viewer

Zathura and **Evince** are two document viewers. You have to use one of them in order to open your PDF files.

To open a document, you should use one of these commands in a terminal window (first the name of the program then the name/path of the PDF document).

```
42sh$ evince document.pdf
42sh$ zathura document.pdf
```

Try it! Download the PDF you are reading right now from your web browser, then open it with:

```
42sh$ evince Downloads/pediluve-d1.pdf
42sh$ zathura Downloads/pediluve-d1.pdf
```

3 Dmenu

You might not want to open a terminal every time you want to execute a program such as `evince`. You can use **dmenu** instead, by pressing `Meta + d`. When doing so, you will see a new field appear in the upper-left corner. If you type `evince` and press `Enter`, the pdf-viewer launches.

Tips

You can give arguments to commands in `dmenu`: for example, `evince document.pdf`.

4 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.

4.1 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 graphically (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.

4.2 Practice time

Time to work! To get started, open a terminal (actually a terminal *emulator*). There are many of them, such as `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 `urxvt` for example, you would use these shortcuts:

<code>Ctrl + Alt + C</code>	Copy selection to the clipboard.
<code>Ctrl + Alt + V</code>	Paste content of clipboard.
<code>Shift + Insert</code>	Paste 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.

4.3 Bonus: keyboard shortcuts for your shell

The shell being a fully textual interface, it can sometimes seem tedious to use for every day operations compared to a visual interface you manipulate with your mouse. The truth is, it can actually be **way** faster to use than any visual interface, but for that you need to master some keyboard shortcuts.

Here is a list of some of the most often used keyboard shortcuts you may find interesting. Do not try to remember them all right now, getting used to them is a habit that takes time to settle in, we simply encourage you to go back to this list from time to time throughout the year and see if you can incorporate one or two of them in your daily use of your shell.

Cursor movements:

- `left` and `right` arrow keys: move the cursor one character to the left or the right in your command;
- `alt+b` and `alt+f`: move the cursor one **word** to the left or the right in your command (this is a huge time saver, cumulatively, compared to `left` and `right`);
- `ctrl+w`: erase one word to the left of the cursor (like `backspace` but for one whole word instead of just a character);

- `ctrl+a`: move the cursor to the start of the line;
- `ctrl+e`: move the cursor to the end of the line;
- `ctrl+k`: erase everything to the right of the cursor.
- `ctrl+u`: erase everything to the left of the cursor.

History manipulation:

- up arrow key: recall the previously typed command (repeating the key recalls gradually older commands);
- down arrow key: recall a more recently typed command (only meaningful when exploring your command history with the up key);
- `ctrl+r`: start a backward search in your history of commands, you can then type the beginning of a command you remember typing some time earlier. The earliest match shows up as you type, you can hit `ctrl+r` again to get the next earliest match, then `enter` to run the command again or any cursor movement to edit that recalled command before running it again.

Miscellaneous:

- `ctrl+l`: clear the screen and show the prompt again on the first line of your terminal;
- `ctrl+c`: cancel the current command (do not run it and show the prompt again);
- `ctrl+d`: send the “end of file” information, effectively ending your interactive session and quitting your shell (in most cases this will also close your terminal window as a consequence, since no program runs in it anymore).

And more! See your shell’s documentation for even more weirdly specific shortcuts (try looking up `zsh`’s `alt+h`, `alt+q` and `alt+e` for example).

Going further...

Note that some systems do not have, or do not support, the arrow keys. On those systems you can use `ctrl+p` and `ctrl+n` instead of up and down, and `ctrl+b` and `ctrl+f` instead of left and right.

Be careful!

Also note that if your shell is currently running a program (that is, you are not writing a command after your shell’s prompt), then all these shortcuts may have very different meanings, since it is the running program that will read and interpret them, instead of your shell.

4.4 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 Project¹ 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**.

¹ <https://www.gnu.org/>

- `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.

5 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.

5.1 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?

5.2 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.

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!**

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.

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

5.3 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.

5.4 Do it yourself

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

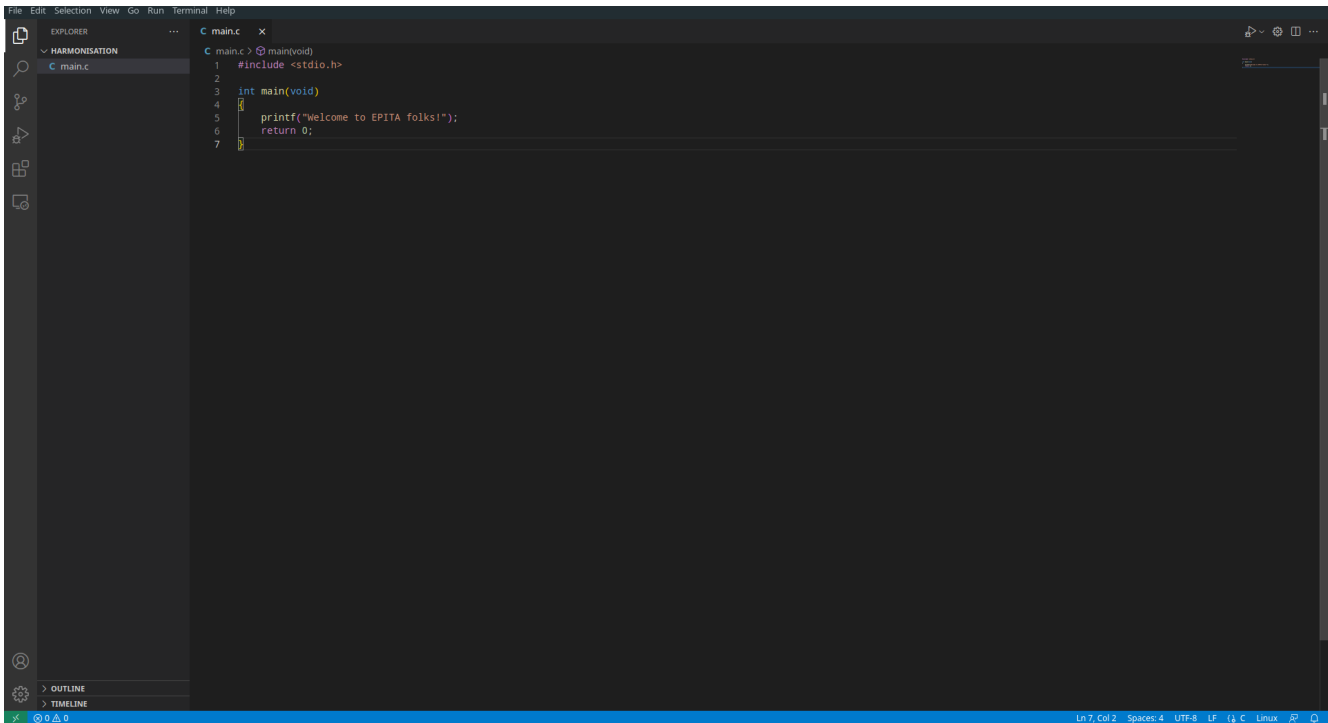
Be careful!

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.

5.5 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), their 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:



Now that you know that this kind of tool exists you can wonder why we forbid them for now. Firstly, 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. Secondly, 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. Finally, we also 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.

6 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.

6.1 pwd(1)

The `pwd` command can be used to display where your current shell “is” in the file system. It prints the full path of your current working directory, hence its name: **Print Working Directory**. Every process currently running in your system has a working directory and by using `pwd` you display your shell’s.

The working directory is an important detail of your shell’s current context: every command you run in your shell will execute relative to the working directory (unless explicitly told otherwise via its arguments).

6.2 ls(1)

The `ls` command lists the content of directories. `ls` means **LiSt**. You can use it without options to display the content of your current working directory.

`ls` has many options, let us look at some of them:

- ‘-a’: do not ignore entries starting with a single dot: ‘.’¹.
- ‘-l’: for each file, display the following extra information :
 - Type (regular, directory, link, etc.)
 - Permissions
 - Number of links to the file
 - User that owns it
 - Group that owns it
 - Size in bytes
 - Last modification date
- ‘-h’: display sizes in a way that is more easily readable by humans, by using powers of 2 suffixes: K, M, G, etc.

`ls` has many more options, it is up to you to read the man page to discover them. For example, what does the ‘-R’ option do? Compare it with the `tree` command.

6.3 Exercise: Lumos

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

¹ In UNIX systems, filenames that start with a dot are considered “hidden” by convention. This is not a security measure though, it only tells some tools like `ls` to ignore these files by default and to not display them, unless explicitly told otherwise, which is exactly what the `-a` flag of `ls` does.

6.4 cd(1)

6.4.1 Moving in the file system

The `cd` command can be used to change the working directory of your shell, this can be seen as moving in the file system. `cd` means **Change Directory**. It takes the desired directory as its argument, which can be an absolute path (starting with a `/`), or a path relative to the current working directory.

6.4.2 Basics of changing directories

There are many ways to change directories, these are some of the usual ones:

- Without options, the `cd` command takes you back to your home directory.
- The `'~'` (tilde) argument is a short alias for your home directory, therefore `cd` and `cd ~` do the same thing.
- On the other hand, if the `'~'` is followed by a user name, for example `~xavier.login`, it then refers to the home directory of **this** user. Therefore `cd ~xavier.login` will take you to the home directory of `xavier.login`, if you have the permission to do so.
- The special directory `'.'` is the current directory. Hence, `cd .` will re-enter your current directory, which shouldn't change anything in the vast majority of scenarios.
- The special directory `'..'` is the parent directory. Hence, `cd ..` will take you one level above in the file system.
- Finally, the `'-'` (dash) option can be used to go back to your previous directory. You can for example go to your download folder by typing `cd ~/Downloads`, type some commands, then go back to whatever folder you were in before simply by typing `cd -`.

Going further...

Notice that `'~'` is not a special argument of the `cd` command but a special character of the shell. So if you use `'~'` in others command in shell it will behave the same way.

6.4.3 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`.

6.5 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.

6.5.1 touch(1)

The `touch` command updates the last access and last modification time of files and directories to the current time without actually changing their content.

If the argument you pass to `touch` does not exist, `touch` will create a regular empty file with this name.

6.5.2 mkdir(1)

The `mkdir` command creates a new directory. `mkdir` means **MaKe DIRectory**. To create a full hierarchy of directories, use the `-p` option. For example to create a folder named `c` with the path `a/b/c/` while `a/` is currently an empty folder (`b` does not exist yet), you need to run `mkdir -p a/b/c`. If you run `mkdir a/b/c`, `mkdir` will complain about the folder `a/b` not existing.

6.5.3 mktemp(1)

The `mktemp` command creates a temporary file or directory in a predefined temporary directory. The path of the created file or directory is then printed back to you so you can work with it. On most systems, when your computer shuts down, all temporary files and directories created with this command are deleted.

6.5.4 cp(1)

The `cp` command copies files and directories. `cp` means **CoPy**. Here are some options you will probably find useful:

- `-r`: copy directories recursively.
- `-f`: force the copy, that is to say remove any file at the destination whose name conflicts with those we are moving, then retry².

² A lot of programs accept a `-f` flag that instructs them to follow through their operation even if it destroys things. This is a dangerous flag you should almost never use, and be very cautious if you do. Many hours of work are lost every year by students (and sometimes teachers too!) because of a misguided use of this flag.

6.5.5 mv(1)

The `mv` command moves files and directories to another destination. `mv` means **MoVe**. It is also the command we use to rename files and directories, you can see the renaming operation as moving a file to another name in the same directory. The `-f` option tells `mv` to overwrite (destroy) any file at the destination that happens to have the same name as those who are moving.

6.5.6 rm(1)

The `rm` command removes regular files and (by default) leaves directories untouched. `rm` means **Re-Move**. Here are some useful options:

- `-r`: remove directories and their content **recursively**³.
- `-f`: ignore nonexistent files.
- `-i`: prompt before every removal.

6.5.7 rmdir(1)

The `rmdir` command removes one or multiple directories if and only if they are empty. `rmdir` means **ReMove DIRectory**. The `-p` option can be used to remove an entire hierarchy of otherwise empty directories. It is a safer command to use than the previous one if you expect the folders you are removing to be empty.

Be careful!

There is no trash bin, it is completely impossible to recover things you deleted with `rm` or `rmdir`.

6.5.8 find(1)

The `find` command takes a path as an argument (".", i.e. the current directory, by default), and then execute an operation on it. There are two important types of operation, the `Tests` one (which will check if the file have some property) and the `Actions` one.

The `find` command is very complete and we will only explain how to search files now, if you want more explanation about it, we invite you strongly to read the `manpage` about it.

The default action of `find` is the `-print` action. So by default if you type just `find` in your shell, it will print every file in the current directory and all of the subdirectory. If you want to find a file with a particular name, you can add the `-name` options follow with the name of the file that you are searching. So a basic use of `find` would be:

```
42sh$ find [path] -name "your_file"
```

³ It is a [very dangerous](#) flag.

6.5.9 grep(1)

The `grep` command finds lines in files that have a pattern given as argument in it. A typical way of executing a simple `grep` command is to give a word you are looking for and then the file that you want.

There are a few options that can be interesting to know for starting:

- **'-n': write the line number on which the word was found before writing the line.**
- **'-r': read in a directory recursively, so it will read all files in it and will continue reading in its subdirectories.**
- **'--color': highlight the word that was searched for in the matched line.**

Going further...

The file argument is not necessary, if you do not put it, it will read the standard input by default, it can be really useful to extract information from previous commands, you will learn more about it later in the year.

6.5.10 sed(1)

The `sed` command is made to transform a file based on a command taken as an argument. You can do many operations with this command, but for now we will only focus on the 's' one. This command allows you to replace automatically a word by another, the syntax for this command looks like this:

```
42sh$ sed 's/previous_word/replacement_word/' [your_file]
```

Tips

By default the 's' command only replace the first occurrence of each line. If you want to do it for every occurrence, you need to add 'g' after the last '/'.

Going further...

But what if, for example, you want to replace all slashes (/) by dashes (-) in a file? Since / is a delimiter used by `sed` to understand the command, we need a way to "mark" a / as "a literal character /". There are two solutions to work around this problem with `sed`, one of them is a very common technique called "escaping" that many tools and programming languages use. If you have some time, try to look up this technique and write a `sed` command that replaces all slashes with dashes. The other one is to change the delimiter. For the example above, you can put as delimiter : so / will be read as a real character. Notice that you can put as delimiter any character.

6.5.11 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.

6.5.12 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.

6.5.13 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.

6.5.14 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.

6.6 Bonus: Filesystem basics

6.6.1 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 others⁴
- /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 system⁵. For example:
 - /usr/bin
 - /usr/sbin
- /etc (stands for **et cetera**, 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 **v**ariable 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

⁴ 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.

⁵ 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
```

7 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.

7.1 i3lock

7.1.1 Why?

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 and get yourself summoned to the lab.
- 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.

7.2 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+1` is a good candidate if you lack ideas).

8 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 **CRI** and usually includes every software and package you will need in your curriculum.

8.1 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.**

8.2 CRI

The **CRI**, meaning “Centre des Ressources Informatiques”, is an EPITA laboratory which takes care of:

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

8.2.1 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.

8.2.2 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 CRI to solve them. So here is how you can make these known.

Concerning computer equipment issues, an issue on your CRI account or the PIE, you can contact the CRI by sending a ticket at [<tickets@cri.epita.fr>](mailto: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 occurred in.

9 The Assistants' portal

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

9.1 The Assistants' intranet

The Assistants' intranet has many features. You will have to use them on a daily basis, so we advise you to get familiar with 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 Assistants' intranet is available at <https://intra.assistants.epita.fr>.

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

9.2 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>.

10 Communication 101

Regularly consulting newsgroups is **mandatory**: it is the main and standard way to communicate with the other students in EPITA.

When you have a question about a current project you can ask it on the related newsgroup.

List	Description
test	Test newsgroup
assistants.news	Assistants news newsgroup
assistants.piscine	newsgroup for the piscine
assistants.projets	newsgroup for the different projects
cri.news	CRI news newsgroup

This means that when you are going to send a news on a project newsgroup, everyone in your promotion and the assistants are going to see it. If you want us to answer you, you **must** respect the *Netiquette*.

You can find the Netiquette in the Assistants' Intranet in the 'Documents' section.

Be careful!

During the entire year, each message you exchange with the assistants (tickets, news...) will have to respect 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.

10.1 Newsreaders

Many newsreaders allow you to read and write articles. The following ones are already installed on the PIE:

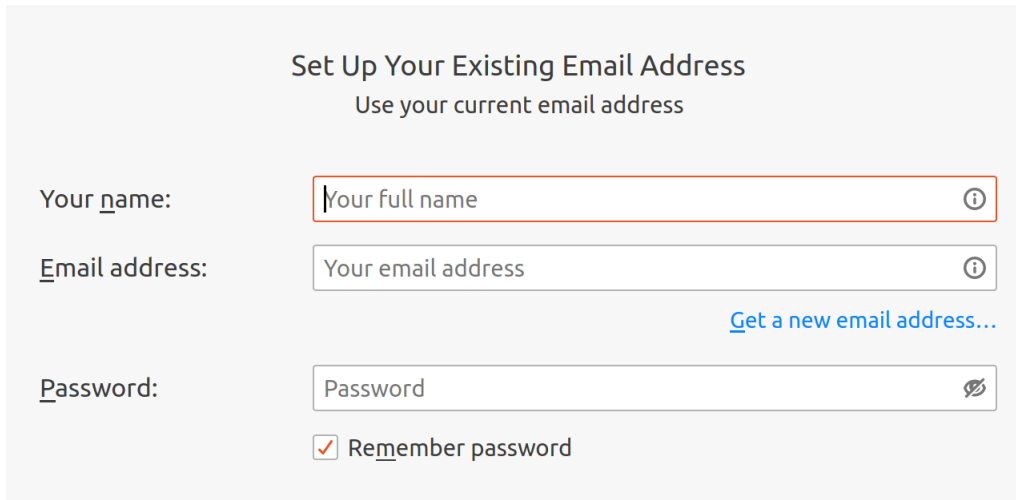
- Thunderbird
- Claws Mail
- Mutt
- Alpine
- Gnus

Most of these newsreaders are also email clients.

10.1.1 Thunderbird

Let us setup Thunderbird so that you can read Epita's news. First launch Thunderbird. If it is your first time, you will be asked to setup your email account. Complete the asked 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.



The screenshot shows a dialog box titled "Set Up Your Existing Email Address" with the subtitle "Use your current email address". It contains three input fields: "Your name:" with a placeholder "Your full name", "Email address:" with a placeholder "Your email address", and "Password:" with a placeholder "Password". Each field has an information icon (i) to its right. Below the password field is a checkbox labeled "Remember password" which is checked. A blue link "Get a new email address..." is positioned to the right of the email address field.

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

Your full name
xavier.login ⓘ

Email address
xavier.login@epita.fr ⓘ

Password
🔒

☒ Remember password

✓ Configuration found in Mozilla ISP database.

Available configurations

☒ **IMAP**
Keep your folders and emails synced on your server

✉ Incoming
IMAP outlook.office365.com SSL/TLS

✉ Outgoing
SMTP smtp.office365.com STARTTLS

👤 Username
xavier.login@epita.fr

☐ **POP3**
Keep your folders and emails on your computer

☐ **Exchange/Office365**
Use the Microsoft Exchange server or Office365 cloud services

[Configure manually](#) Cancel Done

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

Your full name
xavier.login

Email address
xavier.login@epita.fr

Password

☒ Remember password

✓ Configuration found in Mozilla ISP database.

Manual configuration

INCOMING SERVER

Protocol: IMAP

Hostname: outlook.office365.com

Port: 993

Connection security: SSL/TLS

Authentication method: OAuth2

Username: xavier.login@epita.fr

OUTGOING SERVER

Hostname: smtp.office365.com

Port: 587

Connection security: STARTTLS

Authentication method: OAuth2

Username: xavier.login@epita.fr

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

Congratulation, you have successfully setup your email client!

If you did not setup correctly you can still follow the [CRI tutorial](#).

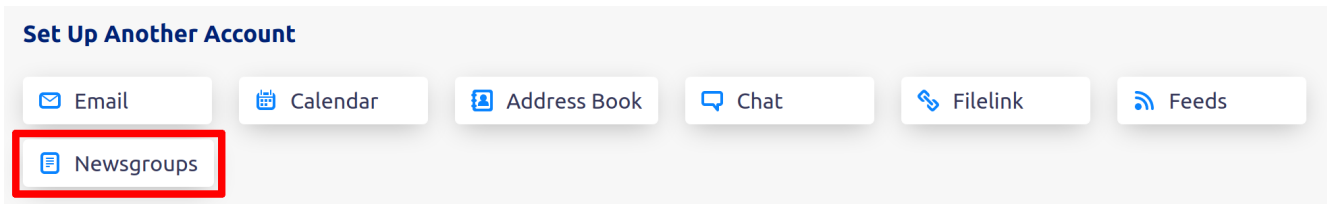
Before setting up your newsgroups, you will have to deactivate the option that composes your emails in *HTML*. Plain text mails are the only accepted format when sending emails to the assistants.

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 box for Compose messages in HTML format.

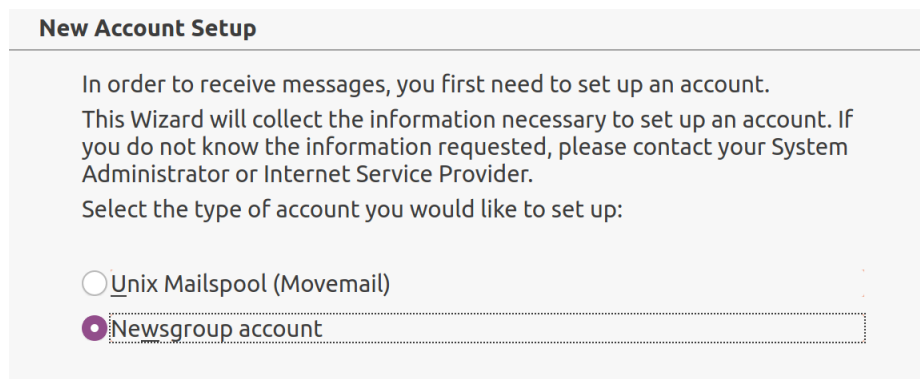
Be careful!

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.

Let us configure your newsreader now. Click on your email inbox and you should see a section Set up another account. Click on Newsgroups:



You should select Newsgroup Account.



You will be prompted to complete your identity:

- **Your Name** should be your full name.
- **Email Address** should be your *Epita* email address.

Identity

Each account has an identity, which is the information that identifies you to others when they receive your messages.

Enter the name you would like to appear in the "From" field of your outgoing messages (for example, "John Smith").

Your Name:

Enter your email address. This is the address others will use to send email to you (for example, "user@example.net").

Email Address:

Cancel Back Next

Click on **Next**. Thunderbird will now ask you for the Incoming Server Information: *Newsgroup Server* should be **news.cri.epita.fr**.

Incoming Server Information


Enter the name of your news server (NNTP) (for example, "news.example.net").




Newsgroup Ser...

Click on **Next**. You can give a name to the newsgroup now, click again on **Next** when you are done naming the newsgroup. Now Thunderbird will congratulate you for adding your newsgroup. Verify if your information is correct, then click on **Finish**.

Let us now focus on modifying the server settings to be able to read and post news. Click on the newsgroup inbox in the left pane navigation then select **Account settings**. Click on **Server settings** and change these *Connection security* to **SSL/TLS**.

Now, let us learn how to add a newsgroup subscription. Click on your newsgroup inbox in the left pane then click **Manage newsgroup subscriptions**.

 **news.cri.epita.fr**

 **Manage newsgroup subscriptions**  Search messages  Manage message filters

Thunderbird will prompt you to choose some newsgroups. We ask you to choose at least:

- assistants.news

- `assistants.projets`
- `assistants.piscine`
- `cri.news`

Feel free to choose other newsgroups, some of them might be interesting.

Be careful!

By clicking on a newsgroup you are subscribed to, you will have the choice to download all headers or download 500 headers. You should choose to download fewer headers (e.g 50 headers) as they are from former news you may not be interested in. You might mark other headers as read in the same time.

Tips

Thunderbird might hang for a while as it tries to download emails and headers. You can make sure it is over by looking at the bottom of thunderbird's window: if there is a progress bar, it is not done yet.

This can take a long time, just be patient.

10.1.2 Signature

To create and add your signature, 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 have to begin by two "-". Do not worry about it, as Thunderbird adds it for you.

Account Settings - xavier.login@epita.fr

Account Name:

Default Identity

Each account has an identity, which is the information that other people see when they read your messages.

Your Name:

Email Address:

Reply-to Address:

Organization:

Signature text: ☐ Use HTML (e.g., bold)

Xavier Login
ING1

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

Let us do the same thing for your newsgroup account so that every article you post will also be signed.

10.1.3 Configuration

Now that you have setup your newsreader, do not forget to add your `.thunderbird` directory to your `.confs` folder inside your AFS, otherwise you will need to redo everything the next time you log in on a new computer.

Execute this command. This can take a long time, be patient:

```
42sh$ cp -r ~/.thunderbird ~/afs/.confs/thunderbird
```

Now go into your `.confs` and open the `install.sh` file with your editor.

Going further...

This script will actually replace the dotfiles from your home by a symbolic link from the files saved in your afs to your home. These notions will be detailed in following tutorials.

Then, by accessing a dotfile from your home like `~/.vimrc`, you will actually open the file stored in `~/afs/.confs/` without even knowing it.

You can get more information about it on [the CRI's website](#).

```
#!/bin/sh

dot_list="bashrc emacs gitconfig vimrc"

for f in $dot_list; do
    rm -rf "$HOME/.$f"
    ln -s "$AFS_DIR/.confs/$f" "$HOME/.$f"
done
```

Add thunderbird to `dot_list`:

```
#!/bin/sh

dot_list="bashrc emacs gitconfig vimrc thunderbird"

for f in $dot_list; do
    rm -rf "$HOME/.$f"
    ln -s "$AFS_DIR/.confs/$f" "$HOME/.$f"
done
```

Run `install.sh` once:

```
42$ ./install.sh
```

From now on, when you will start your AFS, your configuration file for `thunderbird` will be loaded.

Tips

If you want to configure an email client, here are some configurations your mail client may ask you:

- Imap: outlook.office365.com
- Port: 993 SSL
- Smtplib: 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.

10.1.4 Say Hi

Let us now write your article in the `mylife` newsgroup. The article must conform to the Netiquette available on the Assistants’ Intranet. This article must have the following subject:

[PEDI][SUBJECT] xavier.login: my first news

You will tell us about your day and what you are expecting from the next one. Do not forget, you **must** respect the Netiquette when posting a news article!

Be careful!

A dialog box will appear when trying to post the news. To be authenticated, you have to fill the information with your **CRI login and password**.

Tips

Along your life, you will receive many emails about the same projects or events. If you want to quickly find emails with same subject you can use the `Quick Filter` options from the top bar.

If you prefer to sort your emails on 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` and then create or manage where your emails will be automatically redirected to.

10.2 Tickets

Emails are the main tool for solving 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 in either French or English.

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 ticket
- 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

11 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)`.

11.1 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.

11.2 Key generation

To be able to use this protocol, you have to generate a pair of keys. When running the following command, the program asks you to enter a file to save your key: you can simply press `Enter` to use the default file name `id_ed25519`. Moreover, it asks you if you want to create a passphrase to protect your key from unwanted uses. We **highly** recommend you to do so.

```
42sh$ ssh-keygen -a 100 -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (~/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_ed25519
Your public key has been saved in ~/.ssh/id_ed25519.pub
```

(continues on next page)

```

The key fingerprint is:
SHA256:1psNChvLm85fdL7YmQHyIe+lhLDH3AEQfs0GRPaGYSc login@acu_master_server
The key's randomart image is:
+--[ED25519 256]--+
|    o+E .      |
|    . + X      |
|    . + *      |
|    . + .      |
|    + o S .      |
|    . X @ @      |
|    = * 0 =      |
|    . + + = =      |
|    . = . + =      |
+-----[SHA256]-----+

```

Your keys will be available in the `$HOME/.ssh/` directory. The `id_ed25519.pub` key is your public key, the one you can share with others. The `id_ed25519` key is the private key and nobody should have access to it except you.¹

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.

12 Exercises

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

The way is lit. The path is clear. We require only the strength to follow it.

¹ If anyone asks for your private key, **do not** give it to them.