



PISCINE — Tutorial D0

version #v3.1.1



Copyright

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

Copyright © 2020-2021 Assistants <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 someone else.
- ▷ Non-compliance with these rules can lead to severe sanctions.

Contents

1	Foreword	4
2	PDF viewer	4
3	Dmenu	5
4	UNIX and Open-source	5
4.1	Operating systems	5
4.2	UNIX	6
4.3	A word on open-source	6
5	Linux	7
5.1	GNU/Linux distribution	7
5.2	Some examples:	7
5.3	Releases	8
5.4	Package manager	9
5.5	This year	9
6	Window managers and i3	9
6.1	i3lock	9
7	The Assistants' Intranet	10
7.1	Main sections	10
8	PIE	10
8.1	AFS	11
8.2	CRI	11
9	Terminal and shell	12
9.1	What is the shell?	12
9.2	History	12

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

9.3	Practice time	12
10	Elementary commands	14
10.1	Documentation and help	14
11	Files and directories	16
11.1	pwd	16
11.2	ls	16
11.3	Exercise: Lumos	17
11.4	cd	17
11.5	File and directory management	18
11.6	Filesystem basics	20
12	Editors	21
12.1	Vim	21
12.2	Emacs	23
12.3	Exercise	24
12.4	Do it yourself	24
13	Git	24
13.1	What is Git?	24
13.2	Why Git?	25
13.3	Some vocabulary	25
13.4	Git configuration	26
13.5	Obtain an ssh key	27
13.6	The git commands	28
13.7	Git during the <i>piscine</i>	32
14	Communication 101	33
14.1	Newsreaders	33
14.2	Tickets	39
15	Discord	39
15.1	Joining Discord	40
15.2	Practice sessions	40
15.3	Tutorials	40
16	PIE Virtual Machine	41

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, also called *Meta* key. By default, the modifier key is *Windows*.

A modifier key, *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.

Tips

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 this command 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/tutorial-d0.pdf
42sh$ zathura Downloads/tutorial-d0.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 UNIX and Open-source

You can skip this part and read it at home. Proceed to the `i3lock` section.

4.1 Operating systems

Before we can explain what **UNIX** is, it is important to understand what an *operating system* (OS) is. Simply put, an operating system is a bundle of software that serves as an interface between your applications (like your web browser), and the resources of your machine (like your hardware).

4.1.1 Multitasking operating systems

A multitasking operating system allows running multiple tasks at the same time on one computer. Although this seems normal today, it was not the case 50 years ago (mostly due to hardware limitations at the time).

4.1.2 Multi-user operating systems

A multi-user software is a software that allows access by multiple users of a computer. Thus, with operating systems, multiple people can share the resources of the computer.

4.2 UNIX

UNIX is an operating system that gave its name to a family of *multitasking* and *multiuser* computer operating systems.

It was originally developed in the 1970s by Ken Thompson and Dennis Ritchie at Bell Labs, amongst others. It is based on an interpreter, the *shell*, and many small utilities called from the command line interface (*CLI*).

The **UNIX** philosophy can be easily summarized by:

- (i) **Make each program do one thing well** [...]
- (ii) Expect the output of every program to become the input to another, [...]
- (iii) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

These are not the only design principles inherited from **UNIX**. For example, we could also quote the "everything is a file" design principle. You will learn more about it in future tutorials.

Nowadays, the original **UNIX** is no longer used in favor of - more modern - variants¹ of it that share some of its design principles. Those variants can be split in two categories:

- **BSDs**: What we call **BSDs** (Berkeley Software Distribution) are the direct descendants of a variant of UNIX called **BSD**. Today, what we call **BSDs** are, again, variants of the original **BSD**. The most notable ones are *FreeBSD*, *OpenBSD*, *NetBSD* and Apple's *MacOS*.
- **GNU/Linux**: **GNU/Linux** is not an **UNIX** per se (*GNU* literally means *GNU's not Unix!*) but what we call an *Unix-like* operating system. This means that, while it follows the design principles of the original **UNIX**, it may not fully respect **UNIX** specifications and does not contain proprietary code from the original **UNIX**.

4.3 A word on open-source

All the tools you will be using during this semester are *open-source*. This means the source code of those tools is available freely on the Internet and that you are free to contribute to them. It relies heavily on the concepts of *peer programming* and *free licenses*². Open-source projects have a major influence on software development in general and most of the software you use, even proprietary, are built using open-source tools.

¹ By *variants*, we mean systems that use all or part of the original source code.

² Licenses are a **very** important part of the open-source ecosystem. They describe what you can and can't do with an open-source software and there are a lot of them. Licenses are also the source of passionate debates among open-source communities. For example, try talking about GPLv3 to a BSD guy.

5 Linux

You can skip this part and read it at home. Proceed to the i3lock section.

5.1 GNU/Linux distribution

GNU/Linux is an open source operating system from the UNIX family. It is developed and distributed by the community and some companies.

A GNU/Linux distribution consists of:

- the Linux kernel (the core of the operating system)
- the GNU shell utilities (the terminal interface and many of the commands you will use)
- a package manager
- an X server (which produces a graphical desktop) *optional*
- a desktop environment (which runs on the X server to provide a graphical desktop) *optional*

There are lots of distributions, each having their own objectives and philosophy, these distributions are differentiated primarily by:

- ease of use and install
- community
- update frequency
- package manager (see below)
- distribution maintainers

5.2 Some examples:

5.2.1 Debian

This is probably the most widespread in the world because of its big community, huge amount of code, and very high stability.

Debian:

- Released in 1996.
- Free-software only, by default.
- Uses *stable* release.
- Targets server and desktop.

For instance, you will not find Firefox on Debian but Iceweasel because they do not want “[..] inclusion of trademarked Mozilla artwork” in the distribution.

5.2.2 Ubuntu

Ubuntu is a Debian-based Linux operating system and distribution for personal computers, smart-phones and network servers. Being *Debian-based* means it is originally built on top of Debian and shares a few characteristics with it, the most important one being its package manager.

5.2.3 Red Hat Linux

Red Hat Linux is one of the most popular Linux distribution among companies. It is mainly developed by Red Hat Enterprise.

Red Hat Enterprise made the decision to split its Red Hat Linux products into:

- Red Hat Enterprise Linux (RHEL) for customers who were willing to pay for it
- CentOS that attempts to provide a free, enterprise-class, community-supported computing platform
- Fedora that was made available free of charge. It is developed by the Fedora project but is highly supported by Red Hat Enterprise.

This Linux distribution is used by the US army and some US agencies because of its military-grade security.

5.2.4 Arch Linux

Arch Linux focuses on elegance, code correctness, minimalism and simplicity. It expects the user to be willing to make some effort to understand the system's operations.

- Appeared in 2002
- Based on *rolling* release
- As *lightweight* as possible
- Anyone can contribute

5.3 Releases

You may have noticed we have talked about *stable release* or *rolling release*. This refers to the way distributions are updated.

Debian or any distribution that follows a release schema for updates will have major changes due to updates only when updating the distribution to a newer version (for example from Debian 9 to Debian 10). A release has a pre-defined lifetime and will receive minor updates while its supported.

A *rolling release* does not have this concept, and will have newer versions of packages as soon as they are available, at the expense of stability.

5.4 Package manager

“A package manager is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system.”

An anonymous Wikipedia contributor

There are a lot of Package managers on Linux depending on your Linux distribution. It's a core part of a distribution. Red Hat Linux uses its own package manager which is called RPM (RPM Package Manager, previously Redhat Package Manager), Debian uses DPKG, and ArchLinux uses Pacman.

5.5 This year

This year you will use Arch Linux, we encourage you to read and learn more about it on your free time: <https://www.archlinux.org/>

Going further...

Going further: take a look at this graph of unix distribution over time to apprehend the diversity and history of Unix Operating Systems. https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

6 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 to maximize screen space and let you focus on more important things than rearranging your windows by yourself. i3 is probably the best-known window manager but there are other alternatives (Awesome, xmonad, dwm, just to name a few) you might want to try if you install Linux on your own computer. You will only be allowed to use i3 during the period though, 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.

6.1 i3lock

6.1.1 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.

6.1.2 Exercises

Try it now.

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).

7 The Assistants' Intranet

7.1 Main sections

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' C and C++ coding style, the netiquette, and various manuals.
- *Projects*: list of the current and past projects, exercises, groups, traces, and grades.

The Assistants' Intranet is available at <https://intra.assistants.epita.fr>.

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 here: <https://doc.cri.epita.fr/afs/>. It is **essential** that you understand this notion and how to edit or create your configuration files¹.

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

You can skip this section and read it at home. Proceed with the terminal and shell part.

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)
- **Promo/Class groups:** Have a view of current EPITA students
- **Maps:** Check the current occupation in SMs and report issues
- **Documentation:** Look up contact information and basic documentation

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 will soon learn to write proper tickets. Just remember you can contact the CRI 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 occurred in.

¹ <https://doc.cri.epita.fr/afs/#afs-configuration>

9 Terminal and shell

The notions you are going to see today are **very** important. Indeed, your shell is the tool you will use the most during this year and it will be used in many ways. Moreover you will be regularly evaluated on it, so do not hesitate to ask the assistants if you have any questions.

9.1 What is the shell?

In order to use a computer, you make use of an operating system. The core of such a system is the **kernel**. The shell is just an interface to simplify the communication with the kernel. The English word *shell* was chosen because it acts as a shell around the core.

9.2 History

The first shell was the Thompson shell and was minimalistic, 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:

- **csh**: 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**: 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.
- **bash**: the **Bourne Again shell, is the shell you will use this whole** semester. It is part of the GNU Project¹, and inherits from both `csh` and `ksh`. It is also the default shell on most Linux distributions. **It is the default shell installed on the PIE.**
- **zsh**: the **Z shell, inherits heavily from bash, and provides enhanced** completion of commands, options and arguments, shared history, typographical errors corrections...

9.3 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.

Once you have started the terminal, a window should appear. The terminal then automatically runs another program, called a *shell*.

When you open your terminal you should see something like:

```
42sh$
```

This is called the prompt of your shell, ours is `42sh$`, but it could be anything. The shell first displays this prompt and then waits for you to type a command.

¹ <https://www.gnu.org/>

Tips

You will discover that there are many ways to customize your shell.

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 be much better off. 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 as a good practice.

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

```
42sh$ date
Tue Sep 15 03:35:22 AM CEST 2020
42sh$
```

In this example, we are using the `date(1)` program to display the current date and time. You can see that the `date` command terminates because the shell displays its prompt again.

Most programs can take arguments, such as flags and data, that follow the program name, separated by spaces. Flags are easy to recognize because they start with a dash (-). For example, the `--universal` flag of `date` can be used to display the current date in the UTC timezone:

```
42sh$ date --universal
Tue Sep 15 01:35:43 AM UTC 2020
42sh$
```

Some flags can take an argument, in this case you type the flag followed by its argument, 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 the date and time in 3 weeks 5 day 12 hours and 15 minutes from now.

```
42sh$ date --date '3 weeks 5 days 12 hours 15 minutes'
Sun Oct 11 03:51:04 PM CEST 2020
42sh$
```

Finally, commands can take data separated by spaces, for example the `factor` command will display all the factors of the 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$
```

But some data might start with a dash and the program is confused because it parses it as an option!

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

To solve this important problem, you can indicate most programs when to stop trying to parse flags by using the dummy flag `--`.

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

These commands are simple and useful, but they can also be very complex, as you will see in the following days.

10 Elementary commands

10.1 Documentation and help

10.1.1 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.

Using this command must therefore be your first reflex when you 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 you¹.

The man pages cover a wide range of subjects: shell commands, C functions or even general notions. Indeed, pages are sorted in sections of common topic generally designated by a number. Thus, when you are looking for something you can ask for a specific section. The syntax to search for a page in a section is the following: `man [section] page`.

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
- 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 order². For example `man chmod` is equivalent to `man 1 chmod`, which is a different page from `man 2 chmod`. When you write about a specific man page, the usual syntax is *page(section)*. Thus, if you want to talk about `chmod` the user command, you can write *chmod(1)*, but if you want to talk about the Linux function you should write *chmod(2)*.

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.

² This order is specified in the `SECTION` directive of `/etc/man_db.conf`.

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.

10.1.2 `whatis`, `apropos`

In your endless search for answers and documentation, the `man` command can be assisted by other commands:

- `whatis` displays the description of the pages in all sections, as you could read it in the pages' headers.
- `apropos` searches in name and description of the man pages for the keyword you are looking for. It is very useful when you are searching for a command you know exists but have forgotten the name.

10.1.3 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's successively use `man` and `whatis` to discover the following commands:

Tips

Do not only read man pages and descriptions.

You have time, so try each of these commands by yourself.

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

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

10.1.4 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 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 is like a tree structure, the root directory being `/`.

You may have heard that *in UNIX, everything is a file*, but in our systems that's 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, so in the rest of the section when we say *file*, know that this includes directories and other kind of objects, you will encounter those later on.

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.

11.1 `pwd`

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 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.

11.2 `ls`

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

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

- `-a`: do not ignore entries starting with a single dot: `.'`¹.
- `-l`: display more information for each file:
 - kind of file (directory, link, device, etc.)

¹ Starting a filename with a dot `.'` is the UNIX convention to hide it.

- permissions
- owning user
- owning group
- size
- last modification date
- name

- '-h': display sizes to be readable by humans 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.

11.3 Exercise: Lumos

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

11.4 cd

11.4.1 Moving in the file system

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

11.4.2 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 the same as 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 represents the home directory of this user. Therefore `cd ~xavier.login` will take you to the home of `xavier.login`, if you have the permission to do so.
- The special file '.' is the current directory. Hence, `cd .` will refresh your current directory.
- The special file '..' 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.

11.4.3 Exercise: parkour

1. Go to the `/var/log` directory, and list its content.
2. Go back in your home directory.
3. Go to the root directory: `/`, then go back to your previous directory.

11.5 File and directory management

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

Would you like to know more? Read the man pages of each command.

11.5.1 touch

The `touch` command can be used to update and edit the last access and last modification time of files and directories.

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

11.5.2 mkdir

The `mkdir` command creates a new directory. `mkdir` means *MaKe DiRectory*. To create a full hierarchy of directories, use the `-p` option.

11.5.3 mktemp

The `mktemp` command creates a temporary directory. Its location is printed back to you. When your computer shuts down, it will be erased.

11.5.4 cp

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

- `-r`: copy directories recursively.
- `-f`: force the copy, that is to say remove the existing files if they exist and retry.

11.5.5 mv

The `mv` command moves files and directories to another destination. `mv` means *MoVe*. `mv` can be used to rename files and directories. The `-f` option forces the overwriting.

11.5.6 rmdir

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 entire hierarchy of directories.

11.5.7 rm

The `rm` command removes regular files and by default leave directories untouched. `rm` means *ReMove*. Here are some useful options:

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

Be careful!

It is impossible to easily recover files you deleted with `rm`.

11.5.8 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 tree using only one command: `poisson/nage/dans/la/piscine/`.
3. Create two regular files named `nemo` and `.tseT` in your home directory.
4. Create a temporary directory.

11.5.9 Exercise: copying

1. Copy the file named `.tseT` to the `aculover` directory.
2. Copy the `.tseT` file to the temporary directory you just created with the name `Test..` Is this file still hidden?
3. Copy the whole `aculover` directory in the `petit` directory.

11.5.10 Exercise: moving

1. Move the `.tseT` file from your home directory to the `aculover` directory in your home directory, use the option that prompts before overwriting.

11.5.11 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 remove recursively all of its content.
2. Go back to your home directory and delete the `petit` directory without using the `rm` command.

11.6 Filesystem basics

11.6.1 File System

File Systems (abbreviated fs) are used to store and retrieve data in a storage. They define a way to separate data into pieces (called 'file') and to group these files (called 'directory'). You can see most UNIX file systems as a tree where nodes are directories and leaves are files.

Some examples: FAT (FAT12, FAT16, FAT32), exFAT, NTFS, HFS and HFS+, HPFS, UFS, ext2, ext3, ext4, XFS

If you list all directories of `/` you may see something like:

```
42sh$ ls -l /
total 68K
lrwxrwxrwx  1 root root    7 Sep 30  2015 bin -> usr/bin/
drwxr-xr-x  3 root root 4.0K Aug  1 22:51 boot/
drwxr-xr-x 19 root root 3.3K Aug  6 19:22 dev/
drwxr-xr-x 107 root root 4.0K Aug  9 00:59 etc/
drwxr-xr-x  6 root root 4.0K Apr  8 22:24 home/
lrwxrwxrwx  1 root root    7 Sep 30  2015 lib -> usr/lib/
lrwxrwxrwx  1 root root    7 Sep 30  2015 lib64 -> usr/lib/
drwx----- 2 root root 16K Aug 25  2015 lost+found/
drwxr-xr-x  2 root root 4.0K May  2 10:40 media/
drwxr-xr-x  2 root root 1.0K Jul 15 15:48 mnt/
drwxr-xr-x  2 root root 4.0K Aug 25  2015 music/
drwxr-xr-x 17 root root 4.0K Jul  5 15:28 opt/
dr-xr-xr-x 229 root root    0 Jul 15 01:40 proc/
drwxr-xr-x 22 root root 4.0K Aug 12 17:48 root/
drwxr-xr-x 23 root root  760 Aug  1 22:55 run/
lrwxrwxrwx  1 root root    7 Sep 30  2015/sbin -> usr/bin/
drwxr-xr-x  5 root root 4.0K Oct 19  2015 srv/
dr-xr-xr-x 13 root root    0 Jul 15 01:40 sys/
drwxrwxrwt 13 root root  300 Aug 12 12:44 tmp/
drwxr-xr-x 10 root root 4.0K Nov 19  2015 usr/
drwxr-xr-x 12 root root 4.0K Mar  1 22:27 var/
```

Here we will take a look at some of these directories:

- `/bin` contains the essential binaries like `pwd`, `echo`, `mkdir` and many others

- **/sbin** contains the system binaries (for administrators)
- **/usr** (for **U**nix **S**ystem **R**essources) contains directories like the ones in **/** but that are not necessary to minimal usage of the system. For example:
 - **/usr/bin**
 - **/usr/sbin**
- **/etc** (refer to *et cetera*) contains system configuration files
- **/tmp** contains temporary files. These files are deleted after a reboot.
- **/var** contains **v**ariable files. The main goal of this is to have a read-only **/usr** and all modifications made in **/var**

Tips

If you want to know more about the content of your system you can type:

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

12 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 talking of the extension possibilities.

Be careful!

In the two following parts you must try out each command we show you. 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.

12.1 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 fundamental commands of *Normal mode*:

i	Switch to <i>Insert mode</i> .
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 won't 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?

12.2 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, 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, 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.

12.3 Exercise

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

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 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.

12.4 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.

13 Git

13.1 What is Git?

Git is a VCS, for *Version Control System*. It is a program that helps you keep a history of your work by maintaining a history of all the versions of your files. If you have never used a VCS before (Git, Mercurial, SVN, CVS, ...), you have probably managed your file versions “by hand”, by copy/pasting your files to keep a working revision “just in case I break everything”. This manual, tedious and unpractical way of doing version control is over.

After a quick introduction to a tool like Git, you will be able to do “saves” of your work at a given point in time, as many times as you want. The saves will be quick, consume less space than a crude copy/paste and will allow you to keep a chronological history of the different changes you have made to your work. You will then easily be able to go back to a previous version of one, many, or all the files in your project.

13.2 Why Git?

For those who have already used a tool like this, the recurrent question is: “Why Git?”. First, let’s keep in mind that there is no perfect VCS, each of them has upsides and downsides. These are the reasons that motivated our choice for Git:

- It is “*distributed*”, which means it can work without any interaction with a server acting as a “master repository”, which is not the case with, for instance, SVN.
- It is the most used VCS in the world. It is used by large projects you probably already know, like Android, or the Linux kernel (for which it was created!).
- Although it is mainly developed for Linux, it also works on Windows and Mac OS X. It can thus easily be used for any of your projects during your studies, or even side projects.

Only Git will be presented today, but you are invited to document yourself about other tools like Mercurial or SVN to form your own opinion on the different existing tools. However, for all your projects and submissions, only Git will be available.

13.3 Some vocabulary

VCSs have their own vocabulary. We are going to stop on a few terms you have to know to understand the rest of the explanations:

repository Name given to the folder of your project managed by Git. Only the files present in this directory (and its sub-directories, obviously) can be tracked by the Git repository of your project.

remote repository Distant repository accessible by you and your collaborators. You can synchronize your local repository with it. In this way you can collaborate and backup your work by pushing and pulling data to and from this remote.

commit “Committing” is recording changes to the repository.

push “Pushing” is propagating your local saves (commits) to a remote repository.

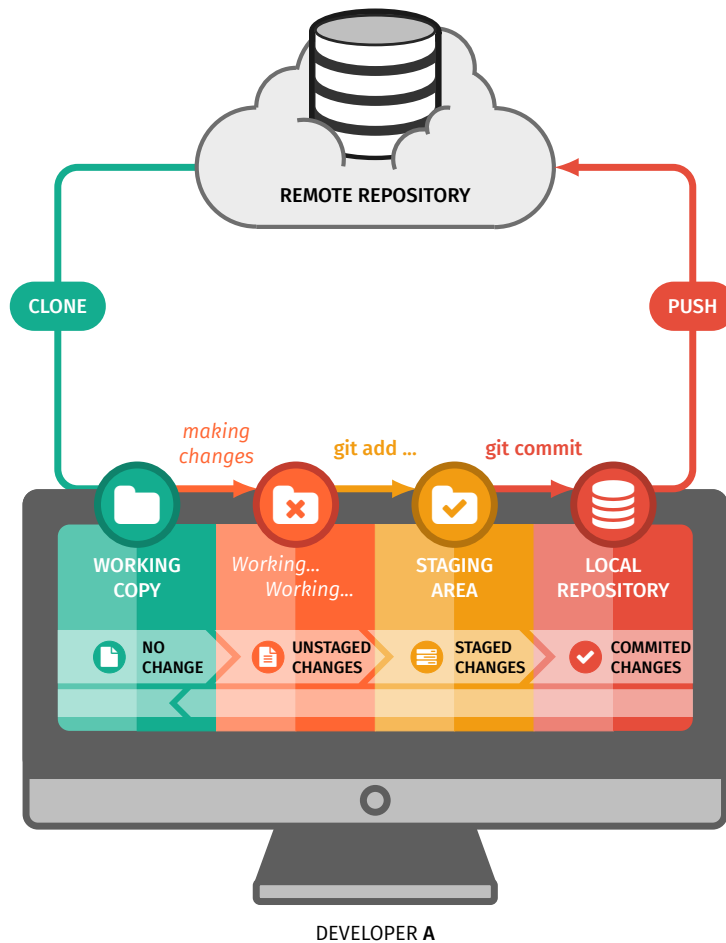
revision Save of your project at a given point in time. For each commit you make, you create a new revision, or version, of your project.

changeset (or SHA-1) Git needs to attribute a version number to each of your commits. This number should not be “sequential” because of obvious conflicts problems. To do so, git attributes a unique SHA-1 (something like *5500bd8e0ae52775c9abf69749cde9c34001650b*) to each of your commits.

HEAD HEAD is the name given to the currently active revision of your project. It thus often matches your last commit.

working copy/workspace The current state of the files of your hard drive. It regularly differs from HEAD.

Here is a schema representing what happens when you edit a file, commit, and push the modifications.



13.4 Git configuration

Git stores its configuration options in three different files, which allows to have different options for individual repositories, users or the whole system. The files, ordered by descending precedence, are:

- `<repository>/ .git/config`: settings of the repository
- `~/.gitconfig`: settings of the user. This is where the settings defined with the `--global` option go.
- `/etc/gitconfig`: settings for the whole system

Here are the commands to use to do a basic configuration:

1. `git config --global user.name "Xavier Loginard"`
2. `git config --global user.email xavier.loginard@epita.fr`

There are obviously a lot of configuration options, like aliases, colors, diff formats, pagers and merge tools. If you want more information on what to configure in Git, see *git-config(1)*.

Going further...

If you open the `~/.gitconfig` file you may see how your git configuration is saved and edit it if needed.

13.5 Obtain an ssh key

SSH (*Secure SHell*) is a secured communication protocol made in order to connect to a remote computer's shell. This protocol is based on asymmetric encryption and need you to generate a pair of keys.

First, generate a pair of **SSH keys** that will allow you to connect to the server containing the `Git` repositories. Then, when the program asks you if you want to create a passphrase to protect your key from unwanted uses, we highly recommend you to do so.

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.

```
42sh$ ssh-keygen -b 4096
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_rsa.
Your public key has been saved in ~/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:N1p5gQKtsD7k6YMJYPyeWIE30T1lH1Ptn/c5ILS6f3w login@acu_master_server
The key's randomart image is:
+---[RSA 4096]-----+
|      ..          |
|      .  .  .     |
|.    o  .  o  .   |
|.o  o  .  o  +  .  |
|o .+..  S B o     |
|.  +=o = + * .    |
|.+++ .B + . + o   |
|.ooo* . .  = E.   |
|      .o  .o.. o.+|
+-----[SHA256]-----+
```

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

You can now copy the content of `id_rsa.pub` in the corresponding field in your profile on the CRI's Intranet².

Beware not to change or move the generated keys, else you will not be able to use them easily.

You will see that your passphrase will be asked several times during this year. You may find it annoying to write it every time you will want to save your work (you will see this in few minutes). Thus, you will be able to use a `ssh-agent` that will keep record of your passphrase and won't ask you to fill it every time. This is the procedure to make the `ssh-agent` work.

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

² To set your editor for commits, use `git config`. For example: `git config --global core.editor "vim"`.

```
42sh$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-ZzU1iSRPrisK/agent.9796; export SSH_AUTH_SOCK;
SSH_AGENT_PID=9790; export SSH_AGENT_PID;
echo Agent pid 9790;
```

This command shows what variables must be declared and exported.

```
42sh$ SSH_AUTH_SOCK=/tmp/ssh-ZzU1iSRPrisK/agent.9796; export SSH_AUTH_SOCK;
42sh$ SSH_AGENT_PID=9790; export SSH_AGENT_PID;
```

```
42sh$ ssh-add
Enter passphrase for /home/login/.ssh/id_rsa:
Identity added: /home/login/.ssh/id_rsa (/home/login/.ssh/id_rsa)
```

This one will ask you your passphrase in order to save it.

After you've done this, your passphrase is kept in cache and won't be asked every time.

13.6 The git commands

13.6.1 git help

All of the git commands that you will type will begin with the name of the binary: **git**. The command you will use the most at first is *git help*. It gives you a non-exhaustive list of the available git commands. For more information about a command, type `git help <command name>`. Do not worry, we will not see all the available commands today, only the ones that will be useful to simply manage your projects and submissions.

13.6.2 git clone

During this semester we will create a remote repository for you for each project. To work on them, you will need to obtain a local copy of these repositories on your local computer. This step can be done with the `git clone` command on your machine. This creates a directory named like the repository you are cloning. You are now able to work on it!

13.6.3 git status

This command allows you to see the current state of your repository. This command is a list of files divided in categories. The first one contains the list of files that will be part of the next commit, the second one lists the modified files that will not be part of the next commit, and the last one lists the files untracked by Git.

13.6.4 git add

Git does not track any file on your repository when you start. You need to ask it explicitly to track files that will be important for the next commit. To achieve this, we use the command: `git add myfile`.

Caveats:

- When you call `git add` on a repository, all of its contents are added to the list of files to be committed.
- Git **cannot track an empty directory**.
- Git **cannot track the file permissions** other than the execution bit (+x).
- Before every commit, you have to do a `git add` of every file you want to commit in your repository.

Be careful!

You must not push any binary files! You will be sanctioned if we see any in your remote git repository.

Exercise

Clone your repository with the url given in the `project/piscine-d0` section of the intranet.

```
42sh$ git clone git@git.assistants.epita.fr:p/2023/piscine-d0/xavier.login-piscine-d0.git
```

Create a `first-submission` folder containing an empty folder named `empty`, and a `joke.txt` file in your repository.

```
42sh$ tree
.
├── first-submission
│   ├── empty
│   └── joke.txt
└── 2 directories, 1 file
```

1. Add the different files in the repository
2. Look at the current state of the repository with the `git status` command. The following result should appear:

```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   first-submission/joke.txt
```

The empty folder is not present: git does not track empty folders.

13.6.5 git commit

The commit is one of the most essential commands. It creates a save of the changes you staged for commit with `git add`. Using this command will open your favorite editor³ and ask you for a commit message: a short comment to describe your commit.

It is very important to put a useful and relevant comment that explains the goal of each commit! The following commit messages are completely useless and will therefore be **sanctioned**: “update”, “fix”, “commit”, “lol”, “mdr”, “prout”, “I hate this project”, “chair”, “kebab”, “flan”. Keep in mind to always explain exactly what the commit changes, and avoid commit messages like “typo” (345 lines added, 568 lines removed). The commit messages will be listed in your repository and allow you to know precisely which version is associated to each commit. On a very large project, when you want to find what broke your project through several weeks of commit logs, you probably would rather find commit messages like “*change the behavior of XXX in the YYY feature to avoid potential infinite loops*” than “lol”.

The commit description does not need to be really large, as soon as it is auto sufficient, for instance: `fix the factorial function when N < 0`.

The `-v` option show the diff of your commit at the bottom of your commit message.

To ensure that you correctly use ``Git`` during your projects, we will check your repositories. The absence of logs, empty logs, stupid commit descriptions and unjustified low number of commits will be sanctioned.

Exercise

1. Commit your previous modifications with the following commit message: “*first-submission: add joke.txt*”.

13.6.6 git log

This command allows you to see the history of your repository as a list of commits, with their author, their date and the description of the commit. This command is really useful to know where we are or to find an old version of your repository that you want to restore.

Exercise

Try the following commands:

- `git log`
- `git log --oneline`
- `git log --stat`
- `git log -p`

³ <https://cri.epita.fr>

13.6.7 git diff

This command allows you to see all the modifications you have done on a file since the last commit, before adding it and committing its changes. You can use:

- `git diff file.txt` to show the changes for `file.txt` only.
- `git diff` to show the changes for all the files.

We advise you to read `git help diff` to see other options, that will allow you to see changes between two commits⁴.

Exercise

1. Write a joke in the `joke.txt` file in the `first-submission/` directory.
2. Add the changes to do a new commit.
3. Change the file again.
4. Look at the `diff` on this file.

13.6.8 git tag

The tags are an important tool provided by Git. They will allow you to identify a revision with a name instead of a SHA-1. For instance, your submissions will have a `submission` tag, so that you know which step of your work you chose to submit for the evaluation. To tag your commit, you will use the following command:

```
git tag -a <tag name> -m <message>
```

Note that you cannot have two tags with the same name in a repository. If you want to change the commit pointed by a tag, use the `git tag -f -a <name of your tag> -m <message>` command.

You can list the tags of your repository with the `git tag` command.

Exercise

Tag your commit with the tag `exercises-first-submission-v1.0`

⁴ or even two branches/tags/remotes/...!

13.6.9 git push

When you commit your work, `Git` keeps track of the versions locally. If you want to propagate your versions on the servers from which you cloned your repository, you need to use the `git push` command. This operation will send all of your local commits to the servers.

Tips

In order to push your tags, you have to add the `--follow-tags` option.

13.6.10 Tag checking

To check that your tag is on the right commit, you can use the `--decorate` option of `git log`.

If you failed your tag and you want to put it on another commit, you can use the `-f` option of `git tag` to overwrite the old tag.

You can also check the tag message with `git tag -n`.

Exercise

1. Push your changes to the server by indicating `git push origin master` for the remote server.
2. Push, this time including your tags.
3. Go on the intranet in the section `project/piscine-d0/exercises`, find the exercise `first-submission` and check that your submission has been accepted.

Tips

As submission is the core of your grading, do not hesitate to call an assistant if you failed to submit your work.

13.7 Git during the *piscine*

We remind you that during the *piscine* you are only going to see a really basic usage of `Git`. Once the *piscine* is over, you will learn a lot of other features of `Git` that will be useful during the rest of the year.

Knowing the commands shown in this tutorial by heart will be a very good start. The web is full of other tutorials if you want to learn more, and you can start by this online book⁵, particularly the chapter on branching.

⁵ <https://git-scm.com/book/en/v2>

14 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

That means that when you are going to send a news on a project newsgroup, everyone in your promotion and the assistants is going to see your news. 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.

14.1 Newsreaders

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

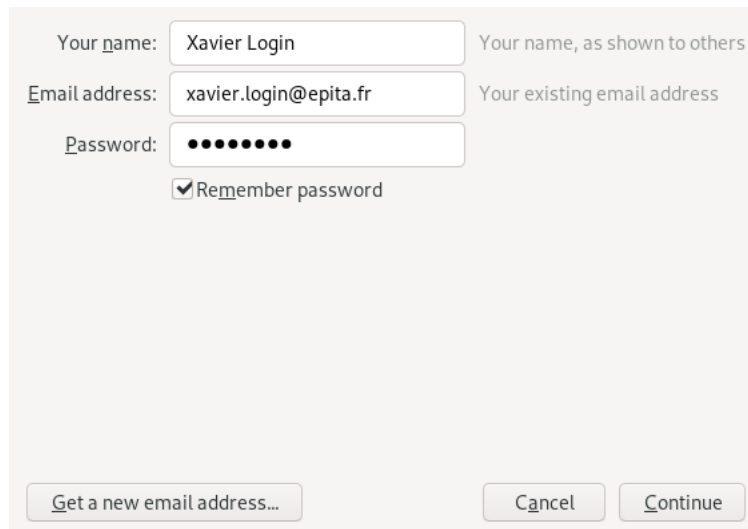
- Thunderbird
- claws mail
- mutt
- alpine
- gnus

Most of these newsreaders are also email clients.

14.1.1 Thunderbird

Let's 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.

A screenshot of the Thunderbird account setup window. It contains three input fields: 'Your name' with the value 'Xavier Login', 'Email address' with the value 'xavier.login@epita.fr', and 'Password' with masked characters. To the right of each field is a label: 'Your name, as shown to others', 'Your existing email address', and 'Remember password' (which is checked). At the bottom are three buttons: 'Get a new email address...', 'Cancel', and 'Continue'.

Press on **Continue** and wait until the button becomes **Done**. Press the button **Done**. Congratulation, you have setup you email!

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

To do that, click on your email inbox, under **Accounts**, click on **View settings for this account**. 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 mails, as you don't have much space on your AFS, try not to download too many mails. You can go to **Accounts**, then **Account Settings**, then go to **Synchronization & Storage**, in the section **Disk Space** you can manage how many mails you want to keep on your AFS.

Let's configure your newsreader now. Click on you email inbox, under *Accounts*, you should see a section **Set up an account:**. Click on **newsgroups**:



You will be prompted to complete your identity:

- **Your Name** should be your 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:

Click on **Next**. **Thunderbird** will now ask you for the **Incoming Server Information**:

- **Newsgroup Server** should be **news.epita.fr**

Incoming Server Information

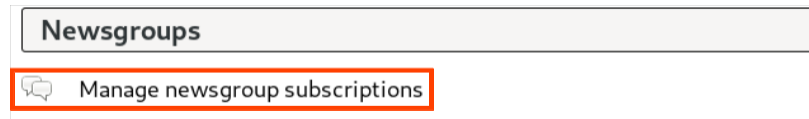
Enter the name of your news server (NNTP) (for example, "news.example.net").

Newsgroup Server:

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**.

Now, let's learn how to add a newsgroup subscription. Click on your newsgroup inbox, under **News-**

Groups, click Manage newsgroup subscription.



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 be asked to 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.

Be careful!

Thunderbird might hang for a while as it tries to download mails and headers. You can make sure it is over by looking at the bottom of thunderbird's window.

This can take a long time, just be patient.

14.1.2 Signature

Create a file named `.signature` in your home directory.

This file contains the signature you will use when posting on newsgroups and when sending emails. It must conform to the netiquette available on the Assistants' Intranet.

Depending on the mail client and newsreader, the signature file can or cannot start with sig dashes ("-- \n"). We ask yours not to include them.

You must be able to read and write to the `.signature` file. Your group and the other users must only be able to read it.

Here is an example of a `.signature` file:

```
42$ cat .signature
Xavier Login <xavier.login@epita.fr>
Ing1
```

After creating your `.signature` file, you have to link it to **Thunderbird**.

To do that, click on your email inbox, under **Accounts**, click on **View settings for this account**. Check the box for **Attach the signature from a file instead**. Now choose your `.signature` file. Click on OK when you are done. From now on every email you will send from your **Thunderbird** will be signed with the content of your `.signature` file.

Tips

`.signature` is a hidden file, in order to unhide it, press `Ctrl-h` when you are in the file explorer.

Let's do the same thing for your newsgroup account. Click on your newsgroup inbox, click on **View settings for this account**. Check the box for **Attach the signature from a file instead**. Now choose your `.signature` file. Click on OK when you are done. From now on every article you will post from your **Thunderbird** will be signed with the content of your `.signature` file.

14.1.3 Configuration

Now that you have setup your newsreader, don't forget to add your `.signature` file and your `.thunderbird` directory to your `.confs` or else the AFS won't save your configuration.

Be careful!

Beware: do not retry the next two commands without deleting the created files.

Here's why:

```
42sh$ mkdir folder
42sh$ touch folder/file
42sh$ cp -r folder new_folder
42sh$ tree new_folder
new_folder
  file

0 directories, 1 file
42sh$ cp -r folder new_folder # same command as before
42sh$ tree new_folder
new_folder
  file
  folder
    file

1 directory, 2 files
```

To avoid this situation, just delete `new_folder` before retrying the command.

Execute those commands. This can take a long time, be patient:

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

Now go into your `.confs` and open `install.sh` 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.

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

Get more information here: <https://doc.cri.epita.fr/afs/#afs-configuration>

```
#!/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 and signature to dot_list:

```
#!/bin/sh

dot_list="bashrc emacs gitconfig vimrc thunderbird signature"

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 and signature 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.

14.1.4 Say Hi

Let's now write your article in the `assistants.piscine` newsgroup. The article must conform to the netiquette available on the Assistants' Intranet. This article must have the following subject:

[D0][SUBJECT] xavier.login: first day

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

Going further...

As Netiquette is not easy to learn, we have written a bot to read your messages and inform you if they do not comply with the Netiquette rules. If it answers one of your messages, it will tell you which rule you did not respect and give you an indication on how to improve your message.

If the information Leodagan gives you does not seem clear to you, do not hesitate to ask for more information from an assistant!

14.2 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 French or English.

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

- Subject: [D0][SUBJECT] xavier.login - first day
- Body: J'aime les ACU, c'est de la folie !
- A signature.

Be careful!

Do not forget:

- greetings and salutations around the body.
- replace xavier.login with your login

15 Discord

All remote tutorials and practicals will happen on the EPITA Discord server. Therefore, it is essential that you understand what will follow. Be sure to ask any assistant for help if needed.

15.1 Joining Discord

In order to join the EPITA Discord server, you will have to follow the latest link posted in the *assistant.news* newsgroup.

We recommend you have a discord account linked to easily access the server later.

When joining the server for the first time, the server's bot, Morpheus, will greet you and ask you to link your CRI account to your Discord account. Once this is done you will have access to public channels.

15.2 Practice sessions

During practice sessions, you can ask assistants for help by sending the following command to Morpheus in a private message:

```
!request Describe your problem here
```

An assistant will take your request as soon as possible, and a new private channel will be created for you to communicate with them.

If you somehow end up solving the problem yourself while waiting for an assistant, you can cancel the request by using the `!cancel` command:

```
!cancel
```

15.3 Tutorials

From D1 on, until the end of the *piscine*, you will have tutorials with the assistants in the morning during the week. These tutorials are group learning sessions that the assistants will lead. You will be in groups and free to ask questions about the content of the current TD.

However, the particular current sanitary situation does not allow these tutorials to be performed in traditional ways. Therefore, these tutorials will take place on the EPITA Discord server. Before each session, the assistants will open voice channels and associated text channels for each group. The choice of groups is free but the groups have a limited fixed size. We expect all students to be present at the start of the session.

During the session, each group will have a text chat allowing students from the group to interact with assistants or other students. Assistants will also be able to interact with these channels and bring to the group's attention relevant items that may have been written there.

Students are vocally muted by default. Nonetheless, the assistants may invite the students to interact vocally during the tutorial, feel free to volunteer.

16 PIE Virtual Machine

The CRI provides a virtual machine that reflects the environment present on the school's computers. This allows you to work at home in pretty much the same conditions.

The instructions to setup the virtual machine are available here:

https://doc.cri.epita.fr/cri_vm/

The only way out is through