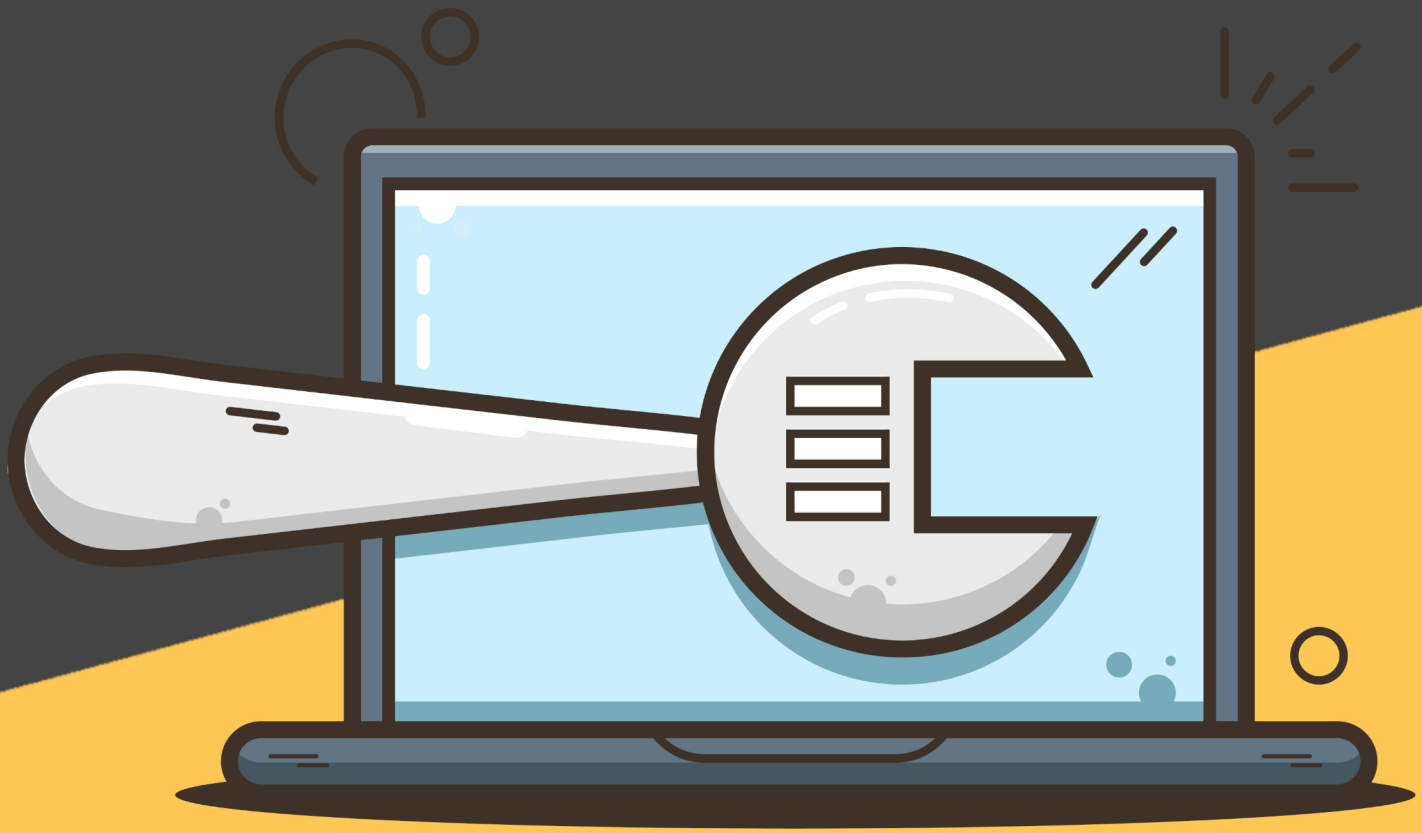
Building your

Mouseless Development Enviroment



Matthieu Cneude

Contents

Building Your Mouseless Development Environment

Welcome, Mouseless Developers!

Thanks for picking up this book. Together, we'll go in a creative journey where we'll build an efficient and mouseless development environment, from the void of a hard disk to a complete system you can mostly use with our friend, the keyboard. We'll evolve from the status of artisan, installing everything manually, to an omniscient entity able to summon the whole development environment with one command line.

There are huge benefits to install a complete system by ourselves: we'll learn a tone along the way. It's good to use tools to achieve our goals, but it's even better to know a minimum how they work. It allows you to find solutions quicker in case of problems, and you'll be able to modify your system depending on your needs.

Knowledge brings you flexibility, and flexibility brings you efficiency.

Our secret weapon? The command line. We'll use it for almost everything we'll do throughout this book, because it's simply the most powerful tool we have. It's consistent, it doesn't get out of fashion, it doesn't go in our way, it doesn't dramatically change when new versions are available. It will make your life easier and it will give you tremendous powers. Who doesn't want that?

If you need more arguments to convince you that the command line is what you need to level up to a higher plan of existence, here we go:

1. As developers, we often have to deal with Linux systems on servers (or containers) where only the command line is available.
2. Some Command Line Interfaces (CLIs) don't have any graphical interface. To use these programs, you need to use the shell.
3. The shell gives you the ultimate power we all seek: automation. It's difficult to automate the movements of your mouse on a graphical interface; it's easier when you deal with plain text, like command lines.

This book will explain everything for you to understand what we're doing, why we're doing it, and how you can personalize it according to *you*. To your workflow. To your personality. The goal: acquiring the knowledge you need by practicing and experimenting. You'll then be able to transfer this knowledge on whatever system you want to work on, even if it's based on another Linux distribution, or if you want to use a more standard IDE instead of Neovim for example. You can even use most of the tools in this book on macOS.

Now, a bummer: we won't dive deep in everything we'll speak about, or this book would be

way too long for all of us to survive it; both the poor writer and the poor reader. Instead, it will focus on showing how the tools we'll configure work together nicely in a Linux-based system.

That said, each chapter of the book concludes on a list of resource you can use to dive deeper to your heart's content.

Who Should Read This Book?

Everyone can read this book, but not everybody will get the most value out of it. So, should you read this book?

If you're a beginner in software development, this book is for you. I try to explain everything you need to know to understand what we're doing. Yet, it can be a bit tough on you because it's a lot of information to swallow at once. My advice: go slowly, don't hesitate to try things out, experiment, play with the command line, and be patient. The rewards are definitely worse the effort.

If you're a seasoned developer but you don't use the command line often, you're at the right place! This book is for you if you want to know more about Linux, the shell, and if a Mouseless Development Environment spark your infinite curiosity. When I discovered it, the spark looked more like an enlightenment. You think I'm exaggerating? Yes I do! But still.

If you already use the command line intensively and the tools I describe in this book, you might not learn much from it. I'm sorry. It breaks my heart more than yours. That said, Building Your Mouseless Development Environment can help you fill gaps in your knowledge. Buying the book can be a good way to support my work, too; if you like my blog, my Github, my style, or my limited charisma. You can still offer this book to some poor souls who still work on Windows, too. It's not that I judge you, Windows developers; I was in your situation for years. It's just that the grass is greener and tastier on this side of the fence.

What is a Mouseless Development Environment?

The kind of obvious goal of a Mouseless Development Environment is to use less the mouse. It doesn't mean that you can throw your mouses through your windows. First, because you might kill somebody, second because it's not nice to pollute, and third because the mouse is still useful in many cases. I'm not a dogmatic.

Using the mouse is great for some endeavors, like graphical purposes, video editing, or musical creations. I wouldn't draw in a terminal like I wouldn't write with a brush; it wasn't meant for that.

I'm sure you noticed but software engineers deal a lot with text: we spend our time writing code and (hopefully) documentation. It's where our keyboard and our command lines really shine. Text never get out of trend: you can write scripts to parse them and to automate whatever you want with them, thanks to the Holy Shell! Repeat after me: glory to the shell!

A Mouseless Development Environment let your hands on the keyboard most of the time, which is a blessing by itself. It's very comfy not to move your hands to reach your mouse, then your keyboard, then your mouse, in an almost infinite recursion of pain. Even if you think that it doesn't bother you, it might be unconsciously. I thought it wasn't a problem for a long

time, but trying to stay on the keyboard definitely changed my mind. It's difficult to admit, but I was plain wrong.

What Do You Need To Follow Along?

To follow this book, you need to have a *place* to install the whole system: Arch Linux, URxvt, Neovim, Zsh, tmux, and so on. If you have an empty hard disk, that's a very good container for that. You can use as well a virtual machine if you want to see first how it looks like, or if you just want to follow the book for learning purposes without the intention to use the final Mouseless Development Environment.

The good new: even if you finish the book on a virtual machine, you'll have a complete installer for the Mouseless Development Environment somewhere on your Github account. It's something we'll build together, too. You can use it on any standard computer you want, even on a Macbook Pro; tested and approved. If you fall in love with your new shiny environment, as I did, you'll be able to install it quickly on a new computer.

You can create virtual machines using Virtualbox. A virtual machine is a simulated computer using the resources of your physical computer. You can install whatever you want on it, without your physical computer to know about it. Both systems are decoupled as much as possible. Virtual machines are great to try different OS or Linux distributions without too much hassle.

If you install the Mouseless Development Environment on a physical computer, I advise you to find a way to be able to read this book while installing everything. You can use another computer, a tablet, or a reader for example.

Creating Your Own Cheatsheets

Since a Mouseless Development Environment focus on using the keyboard when it's appropriate, we'll see many shortcuts (or keystrokes) in this book. We'll go through each tool step by step to understand why and how to use these keystrokes, for you to remember them easily.

Still, you need to be able to come back to these keystrokes if you forget them. You can of course download already made cheatsheets on the Internet, but in my experience they are not very useful for beginners. You risk ending up in an ocean of keystrokes, not really knowing what are the most important ones, suffocating by too many possibilities.

This is what I would advise you to do: write your own cheatsheets while going through the book. One for each tool we'll see together. If you need to be convinced, here some arguments:

1. Writing will help you to remember the keystrokes. You can as well write some comments, categorize them, and even add some personal mnemonics. You can draw something funny near your keystroke. Humour is a great tool to help you to memorize. In short, you'll make them *yours*.
2. You can organize them in a way which makes sense to you.
3. When you'll come back to them, you'll feel they are some extension of your brain, not 10923810938 unknown keystrokes downloaded from a random, cold, and sad Internet corner.

I followed this technique when I learnt to build my own Mouseless Development Environment, and I believe that's one of the reason why I never found Neovim or anything else to have a high

learning curve. I was then able to switch to this mouseless system very quickly. Believe me, it's not because I'm a genius. I'm a very standard human.

In a similar fashion, you should as well write somewhere the command lines you'll learn in this book, for the same reasons.

Experimenting Is Key

To learn, you need to practice and experiment. It's a bit more work than watching passively a Youtube video, but it's more effective too. Write your own cheatsheet, comment your configuration files and bash scripts, and don't hesitate to play around with everything. The goal is not to memorize, but to understand how it works. Try to modify a command line and see what options you can use, for example.

In two words: be curious.

More you'll learn about the shell in general, easier it will be for you to learn what you can put on top and how it works.

Styling Conventions

In this book, there are not many special styling conventions. You'll know what to execute in the shell and what to write in what config files.

Sometimes, `<something>` will pop up in some command lines you need to execute. That is, an expression surrounded with `<` and `>`. For example, `fdisk <your_hard_disk>`, or `su <username>`. These are variables you, and only you, know the values of. No worries: I'll tell you each time how you can find the information you need.

The Choice of Tools

You can replace whatever tools we'll see in this book with others, but I would suggest you to follow the entire book first to understand how they are configured together. Then, you can modify whatever configuration file or replace whatever tool you want because, instead of copying and pasting the configuration of a random person on the Internet, you'll have built your whole system by yourself. You'll have a great control over it.

We'll use Neovim for every editing tasks from the beginning on. If you don't want to learn how it works, you can use Nano instead or whatever else you like.

In a Nutshell

We'll begin, in the next chapter, to shape our system to install Arch Linux afterward. What did we learn in this chapter?

- If you're not already using the command line and a Mouseless Development Environment daily, you'll learn many things from this book. Otherwise, you can still try to fill some gaps in your knowledge.

- Create your own cheatsheets to maximize your learning. Make the useful information your own.
- Don’t be afraid to experiment with the tools and the command lines we’ll see in this book.

Part I - Arch Linux

A General Linux Overview

Let's first see some generalities about the system we'll build, and more precisely we'll try to answer these questions:

1. What's Linux?
2. What's a Linux distribution?
3. What are repositories?
4. What's the shell?

This chapter is a very high level overview of a Linux-based system. Don't worry if you don't remember everything: we'll come back to remember concepts in later chapters.

I know. It might looks very boring for seasoned Linux users. Who knows? Maybe you'll learn from it. Of course, you're free to skip this chapter if you want.

Diving Inside Linux

Linux is an Operating System (OS) like Windows or macOS. The heart of any OS is a program called the *kernel*. It's the interface between you and the hardware of your computer. Thanks to the kernel, the programs running on your system can communicate with the hard disk or the memory, for example.

For you to speak with the kernel around a good cup of tea, you need another interface: the *shell*. For example, you can create a file using the shell, which is in fact you, the user, asking politely to the kernel to create something on the hard disk. The kernel might accept your request or deny it ruthlessly.

The shell is an interpreter, which means that it can read *command lines* and perform actions depending on them, like running a program or creating a file. You can write these command lines yourself using a standard input. The most obvious standard input is a keyboard, but you can use a script too, a file containing command lines to execute. We'll come back to that later in this book.

There are many shells out there you can use, like Bash or Zsh. We'll use Zsh to install Arch Linux, and then we'll continue with Bash for a while.

To use the shell, guess what? We need another interface. In computing, we heavily rely on layers of abstractions, and we use interfaces to communicate between these layers. This time, a *terminal emulator*, or *terminal*, will allow you to access the shell. Again, we'll come back to that later, and explain more in detail what it is.

You'll see throughout the book and in many other places the sentences "execute in a shell",

“execute in a terminal”, “run in a shell”, or “run in a terminal”. These sentences are all synonym. It means that you need to:

1. Type the command the book provide in a terminal.
2. Press the `ENTER` key to execute the command.

Then, our friend the shell will return an *exit code*, not directly visible, and possibly an *output*, some text appearing in the terminal.

The shell should be your friend. She should be your best buddy, because she can do a lot for you. Yes, the shell is a feminine character in my world, but your shell can be whatever you want. The sky’s the limit.

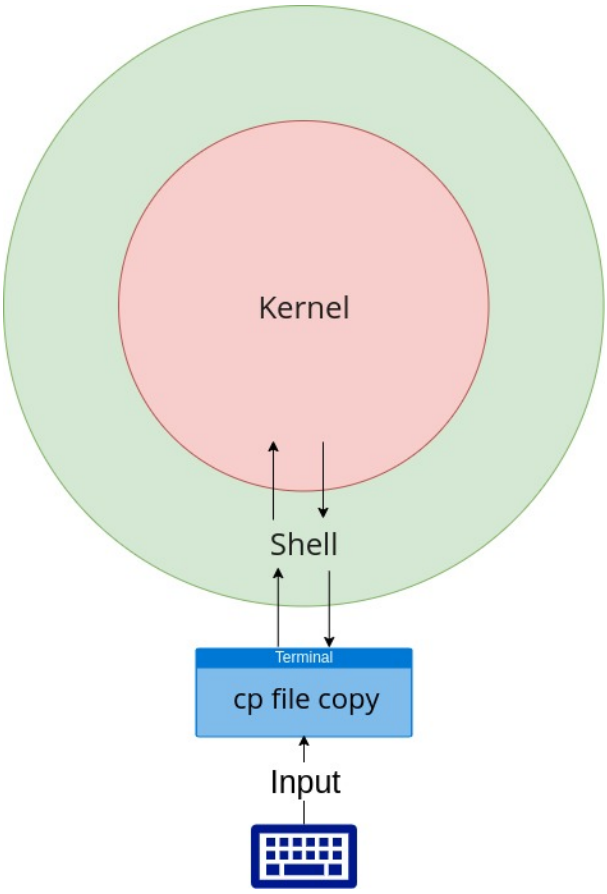


Figure 1: Kernel, shell, and terminal

The Linux Filesystem

The directories in a Linux based system are all children of a special directory called *root directory*. In the filesystem, it will seem hidden at first, known only under the name `/`. Don’t be fooled! It’s the most important directory because it contains every other. You know, a directory to rule them all.

Each direct children of the root directory have a precise purpose we’ll see all along the book.

How does look a path to a file (or filepath) in a Linux based system? For example: `/etc/zsh`. The first `/` is our powerful root directory, supporting everything else as Atlas support the world. The directory `etc` is a direct child of the root directory, and the directory `zsh` is at the

second level. Every `/` which are not the first character of the filepath is a *directory separator*. It's here to indicate a new level of directory. Confusing? I told you the root directory tried to hide!

If you run in a terminal the command `cd /etc`, you'll move from the directory you were to the directory `/etc`. It will become your *working directory*, called as well *current directory*. Each time you move from directory to directory, your working directory change too. You can think of it as the place you are, and you can jump from place to place. To display what your current directory is, you can run the command `pwd` (for `p`rint `w`orking `d`irectory) in a terminal.

There are many special directories in a Linux system. Here are two you'll encounter very often:

- `.` - Represent your working directory.
- `..` - Represent the direct parent of your working directory.

If you run `cd .`, you jumped from the directory you're in to the directory you're in. Was it useless? Absolutely! Now, if you run `cd ..`, you'll move to the parent directory. Your working directory will become this new directory. Similarly, if you run in a shell `cp ./cat_picture ./file`, it means that you'll `c`o`p`y the file named `cat_picture` in your working directory and "move" it to your working directory. It means that you'll have two identical files in your working directory with different names.

Linux Distributions

If you install an OS like Windows or macOS, it will include installing the kernel, a shell, and many other programs as well. For example, the Windows installer will provide a desktop environment, a bundle of programs sharing the same graphical interface. It includes the desktop where you can place shortcuts of your programs, often a status bar with a truckload of stuff in it, a wallpaper, and all of these things. The OS doesn't let you the choice to choose what programs and what desktop to install.

A Linux distribution, commonly called *distro*, is a Linux kernel, a shell, and many programs too. The difference: there are 200+ Linux distros you can choose from. For example, the distro Ubuntu will install some version of the Linux kernel, the shell Bash, a desktop environment called Unity, and many other programs, like some Amazon ads. Who doesn't want Amazon stuff? Thanks, Ubuntu.

Sometimes, when people use the word "Linux", it's often not very clear what they are speaking about: a Linux distro? The kernel? The Linux philosophy? What's important to remember is this: there are not many differences between Linux distributions. You can summarize them as follows:

- The set of applications a Linux distro install on your system.
- The set of applications available in the repositories.
- The philosophy attached to the distro, especially the way you can update the Linux kernel, the shell, and all the programs installed.
- The package manager used to install packages.

Some distros are build on top of other ones. For example, Ubuntu is build on top of another distro called Debian. It means that both will share many similarities.

Arch Linux is a distribution too, but it only installs a very minimal Linux-based system. You'll have the kernel, the shell, a set of programs commonly used in the shell, and a package man-

ager. No desktop environment, no Amazon thingy, not even a graphical environment to display Graphical User Interfaces (GUIs). We'll install all of that ourselves, and at the same time we'll see how these layers work together. By understanding this, you'll be able to install whatever you want and personalize everything following your craziest desires.

Packages and Repositories

We were speaking about programs and applications. In the Linux world you'll see as well often the term *package*. A package is a compressed archive bundling every file for a given application. This archive has metadata to know how to install the application itself. These packages are stored in some locations called *repositories*. To download and install packages, we'll use a *package manager*.

Different Linux distros often use different repositories and package managers. For example, Ubuntu uses APT as a package manager, and Arch Linux uses Pacman (for `Pac`kage `Man`ager).

Why Arch Linux?

Why will we install Arch Linux in this book, and not another distro?

First, Arch Linux users have often the reputation to be arrogant persons, pointing the finger to everybody who doesn't use Arch Linux. They always have a great pleasure to precise that they use Arch Linux in any discussion, even if it's about the price of the cheese at your favorite supermarket. I wanted to invite you in this very closed club. My pleasure.

On top of the fact that installing Arch Linux can teach a great deal about Linux, using it in our daily work has many advantages, too.

Glory to the Rolling Distribution!

As we saw above, the different Linux distros out there have different ways to manage updates for everything installed on your system, from the Linux kernel to the packages installed.

Arch Linux is a rolling distribution; the packages are keep up-to-date as much as possible. You can be pretty confident that you'll have access to the latest versions of your favorite applications. The small downside: you'll have to update your system pretty often, every week or every other week. Trust me, that's a small price to pay for having the most up-to-date system possible.

If you listen to the urban legend that Arch is "very unstable" because of that, don't believe it. Some friends run Arch Linux for years (otherwise they wouldn't be my friends of course!) and I do. too. We never had any problems doing so. I used Windows from Windows 98 to Windows 7, macOS, and some Linux distributions: Arch Linux is the most stable system I've ever had.

Arch Linux Community

The Arch Linux community is great. Arch Linux folks have solid knowledge in many areas, and they have very useful tips too. The best place to fall in admiration for the Arch Linux com-

munity is the Arch Linux wiki. It's simply the best resource you'll find regarding Linux, even if you use another distro.

Official Repositories and the Arch User Repositories (AUR)

The official repositories of Arch Linux propose many packages. I'm not exaggerating: you'll find most of the applications you need in there. Even if you don't, you'll have access to the Arch User Repository too, an unofficial repository where you'll find everything and anything. No need to install and compile the obscure application you seek anymore; an AUR helper can do that for you.

Manual and Help Options

If you want to read some documentation about a command line, you can run in a terminal:

```
man <command>
```

For example, you can run `man cd` to read the manual of the command `cd`. If you're a beginner, it might be a bit difficult to read at first. When you'll understand better some concepts distilled in the book you're reading right now, you'll find `man` less cryptic.

Keep in mind: you don't have to read everything. Often, reading the description is enough to understand what the command is about. If you search a precise *option*, it can be very helpful too. A command line option is very often a single letter prefixed by a minus `-`, or a word prefixed by a double minus `--`. Adding options will modify the behavior of a command.

A command line has often some kind of option to display some help to use it, something like `--help` `or -h`, for example. The output is often shorter than the man page for the same command. To understand what I mean, try to run:

```
ls --help
```

If you want to know more about man, you can read the man page of man by running in a terminal:

```
man man
```

So meta.

Troubleshooting

Installing manually everything following a book can lead to some problems. Heck, using a computer lead to some problems. In my experience, using an automatic installer for a Linux distro can lead to some problems, too. In short: we can have problems.

If you run into unexpected errors and weird behaviors, you should take a look at these resources:

1. The Arch Wiki is, as we just saw, the best resource for almost everything, except the meaning of life. If you search on the Internet " Arch Linux", you'll end up very often in the Arch wiki.

2. The official Arch Linux website is the place to go when you have problems updating your system. It describes small manual intervention you need to do sometimes. Everything is always explained clearly.
3. The Arch Linux forum is a good place to go if you have any question. Be sure that your problem is related to Arch Linux and that you've done some research before posting there.

In a Nutshell

Get ready! In the next chapter, we'll begin to prepare our hard disk for Arch Linux. What did we learn in this chapter?

- The kernel is the heart of a Linux-based system. You can talk to the kernel using the shell, which will read and interprets commands you type in a terminal emulator.
- In the Linux filesystem, every directory are children of the root directory, displayed as a single slash `/`. A filepath looks like this: `/etc/kernel`, where the first `/` is the root directory.
- A package is an installer for an application. They are located in repositories you can access using the Internet. You need a package manager to install the application contained in a package.
- Linux distros are not that different from each others. Mostly, they have a different philosophy to handle packages, different package managers, and different repositories. They come bundled with different applications, too.

Going Deeper

- Arch Wiki - Arch Linux
- Arch Linux Website
- Arch Linux Forum

Preparing Your System

It's time to get our hands dirty! In this chapter, we'll prepare our system for Arch Linux to feel at home. More precisely, we'll see:

- How to burn an Arch Linux ISO on a USB key.
- How to configure the Arch Linux live system.
- How to partition a hard disk using fdisk.
- How to create and mount new filesystems.

This is the beginning of The Journey, so I'll try to explain most things in details. Ready for the challenge? Let's go!

Prerequisites

To install our new Mouseless Development Environment, we'll need the following:

1. A computer with a 64-bit CPU (not older than 10 years).
2. An empty hard disk, or a hard disk you'll wipe out (at least 20GB to feel comfy).
3. A USB key (at least 1GB).
4. A second computer, a tablet, or whatever device allowing you to read this book while installing everything.
5. Internet access on your second device is highly recommended, in case you have a problem in the first chapters.

As we spoke about already, the computer can be a virtual machine or a physical one.

Burning Arch Linux ISO

If you wonder what the heck an ISO is, it's a file representing a physical disk. We can burn ISO files on a real disk, which means exactly copying everything from the ISO to the disk.

To install Arch Linux, we need to start (or *boot*) our computer using the Arch Linux live system, and install the OS from there. Here's how to do that:

1. Download the Arch Linux ISO. Find your country, click on the first link below, and download `archlinux-<the_last_release_date>-x86_64.iso`.
2. Verify that the ISO is the official one, and has not been modified by horrible hackers. You need to generate the MD5 hash of the ISO and compare it with the one provided on the download page.
   - On Windows, you can use `CertUtil -hashfile <path_to_ISO_file> MD5`.

    - On Linux, you can use the command line `md5sum < path_to_ISO_file >` to get the MD5 of the file.
3. Burn the ISO on a USB key. You can use:
    - On Windows: rufus.
    - On Ubuntu: Startup Disk Creator.
    - On another Linux distro: UNetbootin.
4. You need to change, in your BIOS, the boot order to have USB before your hard disk.
5. Plug your USB key and start your computer.

For the fourth point, you can normally access the BIOS interface by hitting a precise key when your computer starts. If you're lucky, it will be displayed at startup long enough for you to see it. It's often `ESC`, `DEL` or `F10`. Every computer is different on the matter, so I can't really help you more.

When you found a way to access your BIOS' interface, search any option concerning the boot. You can normally change the boot order of the devices: the goal is to put your USB device before the hard disk. When you're done with that, there'll be some options to save your changes and restart your computer.

If you did everything correctly, your computer will boot on your USB key and you'll see a menu inviting you to install Arch Linux. Let's do it!

Configuring the Arch Linux Live System

Your screen will then spit out many green "OK", telling you what's started. Then, you'll end up in a shell prompt. It will look like this:

```
root@archiso ~ #
```

A prompt is a set of character indicating that you can execute command lines. Here, it indicates what user you are (`root`) and the hostname `archiso` (the name of the system).

Welcome to the Arch Linux live system! What's a live system, you might wonder? For now, nothing has been installed on your hard disk. The system you see is running from your computer's memory (RAM). Many distros propose a live system for you to test it even before installing anything.

It means as well that everything you're doing in the live system won't survive if you shut down or restart your computer, except if you write deliberately something on your hard disk. For example, any program you'll install on the live system won't be installed on your hard disk.

The shell we're using now is called the "Z shell", or Zsh. It's similar to the most famous shell, Bash. We'll dig more into Zsh later in this book.

Here are some functionalities you can use with the shell:

- You can enter any command in a shell prompt and execute it by hitting `ENTER`.
- The shell can save a command line history:
    - You can quickly access executed command lines with the keys `ARROW UP` and `ARROW DOWN`.
    - You can search through the history with `CTRL+r`.
- You can use Zsh auto completion by hitting `TAB`.

- To stop a running command line, hit `CTRL+c`. Be careful with it: patience is sometimes safer. Think about what your command is doing before interrupting it, and what's the price you might have to pay by stopping it before it's done.

You are now the root user. You can create different users on a Linux system, but the root user is special. It can do everything a user can do. It has all the permissions. It has almost all powers.

Don't confuse the root user and the root directory `/` we saw in the previous chapter. A filesystem and a bunch of users are different things.

By being the root user, you're in fact the demigod (or demigoddess!) of the "archiso" live system. Why only demi? Because a user can't control directly the kernel. The kernel is the real master around here.

Keyboard Layout

By default, you'll end up with the American keyboard layout on the Arch Linux live system. If you're not comfortable with it, you can change it. Here are the two commands you can use to do so:

1. `ls /usr/share/kbd/keymaps/**/*.map.gz` - list all layout available.
2. `ls /usr/share/kbd/keymaps/**/*.map.gz | grep "< your_language_code >"` - filter all layout with grep.

For example, if I want to use a French keyboard layout, I can do this:

```
ls /usr/share/kbd/keymaps/**/*.map.gz | grep "fr"
```

The result will look like the following:

```
/usr/share/kbd/keymaps/i386/azerty/fr-latin1.map.gz
/usr/share/kbd/keymaps/i386/azerty/fr-latin9.map.gz
/usr/share/kbd/keymaps/i386/azerty/fr.map.gz
/usr/share/kbd/keymaps/i386/azerty/fr-pc.map.gz
/usr/share/kbd/keymaps/i386/bepo/fr-bepo-latin9.map.gz
/usr/share/kbd/keymaps/i386/bepo/fr-bepo.map.gz
/usr/share/kbd/keymaps/i386/dvorak/dvorak-ca-fr.map.gz
/usr/share/kbd/keymaps/i386/dvorak/dvorak-fr.map.gz
/usr/share/kbd/keymaps/i386/qwertz/fr_CH-latin1.map.gz
/usr/share/kbd/keymaps/i386/qwertz/fr_CH.map.gz
/usr/share/kbd/keymaps/mac/all/mac-fr_CH-latin1.map.gz
/usr/share/kbd/keymaps/mac/all/mac-fr.map.gz
/usr/share/kbd/keymaps/sun/sunt5-fr-latin1.map.gz
```

Then, you can choose your layout, by using the filename without the file extension `map.gz`:

```
loadkeys <filename>
```

For example, for my French keyboard layout, I would run the command `loadkeys fr-latin1`.

Connecting to Internet

What would we be without Internet? Nothing more than unconscious amoebas lost in a chain of misconceptions we call reality. Without Internet, I wouldn't have been able to write the

previous sentence!

To connect to this infinite amount of knowledge and cat pictures, we need some network device. The command `ip-link` will show you the truth about them. Let's try to run in the terminal:

```
ip link
```

Something like this will be output:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
   DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s25: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
   state DOWN mode DEFAULT group default qlen 1000
    link/ether f0:de:f1:84:f3:a6 brd ff:ff:ff:ff:ff:ff
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
   mode DORMANT group default qlen 1000
    link/ether 08:11:96:0a:12:b4 brd ff:ff:ff:ff:ff:ff
```

Don't worry about the first result `lo`. The second one `enp0s25` is my network device for Ethernet cable; it often begins with an 'e'. The third one `wlp3s0` is for the Wi-Fi; this one often begin with a "w". We don't have the same computer, so it's very likely we won't have the same network devices.

If you want to connect to Internet using a cable, simply plug it. For the Wi-Fi, you can run the following:

```
iwctl
```

A new shiny prompt will appear. Here are the commands you can use:

1. `device list` - List all your Wi-Fi devices.
2. `station <device> get-networks` - Replace `<device>` with the name of one of yours, to get a list of networks you can connect to with this particular device.
3. `station <device> connnect <network>` - Replace `<device>` and `<network>` by the device you want to use with the network you want to connect to.
4. If a password is required, enter it.
5. `station <device> show` - Show you if you're indeed connected.
6. `exit` - Exit `iwctl`.

If it failed, you might get the friendly message `Operation failed`. Don't worry and try again: even demigods fail, sometimes. Demigoddesses too. Another reason why we're only "demi" here, and not the full version.

Let's now verify if you're really connected. Millions of people use this command all around the word as we speak, to verify their Internet connection:

```
ping -c 5 thevaluable.dev`
```

It will send 5 request to the address `thevaluable.dev` and display the following:

```
64 bytes from v22017064675050008.supersrv.de (185.183.156.38): icmp_seq=1
    ttl=58 time=32.6 ms
64 bytes from v22017064675050008.supersrv.de (185.183.156.38): icmp_seq=2
    ttl=58 time=32.6 ms
64 bytes from v22017064675050008.supersrv.de (185.183.156.38): icmp_seq=3
    ttl=58 time=32.1 ms
64 bytes from v22017064675050008.supersrv.de (185.183.156.38): icmp_seq=4
    ttl=58 time=32.9 ms
64 bytes from v22017064675050008.supersrv.de (185.183.156.38): icmp_seq=5
    ttl=58 time=32.2 ms
```

If you've got something similar (the time will be different), you're connected! Lucky you.

If it doesn't work, try:

```
ping -c 5 185.183.156.38
```

If this last command works, it means that you can't resolve addresses like `thevaluable.dev` to its IP address `185.183.156.38`. In that case, try to run a dhcp client:

```
dhcpcd &
```

What's `thevaluable.dev`? It's my blog, to show you how to do some subtle product placement in your own book.

System Clock

Let's now use one of the oldest protocol on the Internet. Anxious? Don't worry, Internet is the only stuff which never breaks in the software development world. Not like enterprise Java code.

We can synchronize our system clock with the Network Time Protocol. Run this command:

```
timedatectl set-ntp true
```

That's all. Next!

BIOS or UEFI?

You remember the interface you used to change the boot order of your devices, to boot your USB key before another device?

This was the interface of a *firmware*, a piece of software tightly linked to a piece of hardware. In this precise case, this firmware is called a *BIOS* (for `B`asic `I`nput `O`utput `S`ystem) and it's directly embedded in a chip in your motherboard. The BIOS is the first software running when you're booting (starting) your computer.

This BIOS verify that your hardware works properly. It makes available an interface as well for you to configure some basics, like the order of the booting devices.

It will as well run the *bootloader*, another program which is used to boot an operating system, like Arch Linux.

That said, the BIOS is now considered deprecated for a better alternative, called *UEFI* (for `U` nified `E` xtensible `F` irmware `I` nterface). BIOS and UEFI are two different firmwares, but confusion arises when the UEFI pretends to be a BIOS, sometimes using the term "BIOS" even on its interface! It's often an ironic attempt not to confuse people who are used to have a BIOS.

Long story short, because the firmware of your motherboard start the bootloader of Arch Linux, we need to know if you have a BIOS or an UEFI on your computer. To verify that, simply run in the shell:

```
ls /sys/firmware/efi/efivars
```

If you see the error `no such file or directory`, it means that your computer has a BIOS. If it output a bunch of files, you have an UEFI. You need to remember what kind of firmware you have to install Arch Linux properly. My advice: write it somewhere, on a piece of paper for example. We'll need this information to create our boot partition and install the bootloader itself.

We've just learn the first piece of software which run on our computer. You know what we should do? Create a diagram we'll complete as we go, to describe the boot process of a Linux-based system:

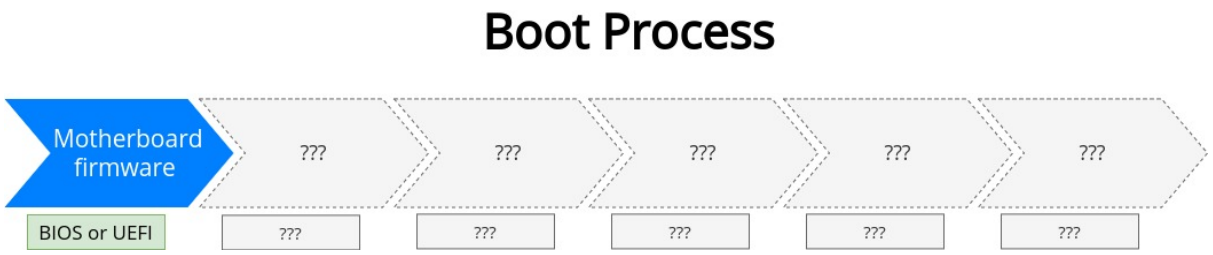


Figure 2: The boot process: BIOS and UEFI

This process is followed by all Linux distros out there. The green box under the arrow indicate the program (or group of programs) used during this process, sometimes specific to Arch Linux.

Partitioning the Hard Disk

Enough explanations. The serious things begin. We'll write soon on our hard disk. First, let's run the following in the shell:

```
lsblk
```

This command will display your *block devices*. A block device is a file which represent a physical device. Confused? How can a device, such as a hard disk, be represented as a file?

A Linux based system tries hard to have a consistent way for you, your applications, the kernel, and the hardware, to interact with each other. In other words, Linux offers a consistent *interface* between you and your physical system, and between the programs running and the physical computer.

Indeed, to facilitate this deep and intimate relationships, there is an important principle to understand on a Linux system: **everything is a file**. You don't need to ask yourself how a pro-

gram read or write a device, like your hard disk: the answer will always be a file.

For example, try the following:

```
cat /etc/proc/stat
```

The output will give you the CPU status at the precise time you read the file. That's what I mean when I speak about an interface between you (or some running programs) and a device, here your CPU.

We'll come back to this idea over and over: it's central to any Linux based systems. For now, if you look at the output of our command `lsblk`, you'll see something similar to that:

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 465.8G  0 disk
  sda1    8:1    0   200M  0 part /boot
  sda2    8:2    0    16G  0 part [SWAP]
  sda3    8:3    0 389.6G  0 part /
```

- `sda` is a file representing my hard disk (`TYPE disk`).
- `sda1` to `sda4` are partitions of the hard disk (`TYPE part` for partition). A partition is a virtual division of a physical hard drive.

If your hard disk is empty, you shouldn't see any partition.

You can see the files representing your hard disk in the `/dev` directory ("dev" for "device").

Wiping your Hard Disk

If your hard disk is not empty, you need to delete everything on it. All its data will be lost, forever.

Depending on the size of your hard disk, it can take a long time. Sometimes a whole day! If you're good with that, run the following in your shell:

```
dd if=dd if=/dev/zero of=<your_hard_disk> status=progress
```

You need to replace `<your_hard_disk>` with yours. For example, considering the output above when I run the command `lsblk`, my hard disk would be `/dev/sda`. As a result, to wipe it, I would need to run `dd if=dd if=/dev/zero of=/dev/sda status=progress`.

If you have multiple hard disks, be sure to select the good one! The command `dd` is merciless; its sweet little nickname is "Disk Destroyer". It won't ask you anything and wipe out whatever hard disk you provide. You don't want to mess up with `dd`!

It will copy everything from the file `/dev/zero` to whatever file representing your hard disk (for me, `/dev/sda`). The file `/dev/zero` is a file containing an infinity of zero; your hard disk will be filled with zeros till it's full. At that point, the command `dd` will stop and your disk will be "clean".

Instead of `/dev/zero`, you can use `/dev/urandom`, which will fill your hard disk with random numbers. It takes even more time, so do that only if you don't want anybody to be able to restore the old data of your hard disk.

Using fdisk

Time to virtually break our hard disk in pieces! We need to partition it for Arch Linux to come and make its nest. To do so, we'll use a simple and powerful Command Line Interface (CLI) called `fdisk`.

The first time I installed Arch Linux, `fdisk` felt very mysterious to me. It took me a long time to partition my disk, and I couldn't repeat the process afterward without going through the same pain. In fact, `fdisk` is quite easy to use.

First, we need a *partition table*. It's read by the OS to know what are the partitions on the disk.

Then, we'll create three partitions:

1. A *boot partition*, where the bootloader to start the operating system will be installed.
2. A *swap partition*, used as memory when your RAM is full. It's not perfect since the RAM is way quicker than a hard disk, but it's better than nothing.
3. A *root partition*, where everything else will be stored: Arch Linux files, your cat pictures, your secret poetry, and so on.

The root partition is a different thing than the root directory or the root user we saw earlier. They have the same name "root", but they live in different realms.

Let's now run the following command in your terminal:

```
fdisk <your_hard_disk_file>
```

For example, I can see that my disk is called `sda` when I run `lsblk`, so I should run the command `fdisk /dev/sda`. From there, a new prompt `Command (m for help)` will appear.

You can execute special one-letter commands in fdisk. Nothing will be written on your hard disk, except if you use the command `w` (for `w`rite). So if you do a mistake, relax: you can abort everything with `CTRL+c` if you need to, and follow the process again from the beginning.

First step: creating a GPT partition table. Let's execute the following command in our new prompt:

```
g
```

It can be surprising to use one-letter commands, but it's how fdisk works.

Boot Partition

Now that we've chosen our partition table, let's create the boot partition. Enter the following command:

```
n
```

At that point, `fdisk` will ask you the partition number. The default is normally `1`, so just hit `ENTER`. You can as well type `1` and press `ENTER`.

As command lines go, fdisk is very curious. It loves to ask questions. It will ask you then the first sector of the disk where the partition should begin. By default, it's the beginning of the disk, and that's what we want: simply hit `ENTER` again.

The last question will ask you what should be the last sectors for the partition. It's not obvious, but you can enter the size of the partition here. For the boot partition, 512M is enough, so let's type:

```
+512M
```

You'll come back to fdisk's prompt after that. The whole operation should look like this, including the creation of the partition table:

```
Command (m for help): g
Created a new GPT disklabel (GUID: <some_GUID>).

Command (m for help): n
Partition number (1-128, default 1):
First sector (2048-71567838, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-71567838, default 71567838)
   : +512M

Created a new partition 1 of type `Linux filesystem` and of size 512MiB.

Command (m for help):
```

We need now to change the type of the boot partition, depending on your boot firmware (UEFI or BIOS). To do so, enter the following command:

```
t
```

To list all the partition types possible, you can run the command `L`. To quit the list, use `q`.

UEFI Boot Mode

If you have an UEFI, we want the type of the boot partition to be `EFI System`. It's normally the first one in the list, so you should enter:

```
1
```

Here's how the whole operation should look like:

```
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): 1
Changed type of partition 'Linux filesystem' to 'EFI System'
```

BIOS Boot Mode

If you have a BIOS, we want the type of the boot partition to be `BIOS boot`. It's normally the fourth in the list, so you should enter:

```
4
```

Here's how the whole operation should look like:

```
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): 4
Changed type of partition 'Linux filesystem' to 'BIOS boot'
```

Root and Swap Partition

We have now configured the boot partition. Don't forget: we still didn't write anything on the hard disk, so if you quit fdisk now, you'll need to redo everything from the beginning.

You should still see fdisk's prompt at that point. We can now create a SWAP partition. If you have more than 8Gb of RAM and you don't do anything which takes a lot of memory (like video editing, for example), you can skip the creation of a SWAP partition entirely. That being said, I always like to create one of 8Gb, because "you never know".

Let's create a new partition with the command:

```
n
```

When asking the partition number (normally `2` by default), just hit `ENTER`.

Again, when `fdisk` will ask you the first sector of the partition on the disk, the default is just after the end of the first partition. Simply press `ENTER`, because that's what we want.

Then, we can enter the size of the partition. It needs to be bigger than 512M. I want `8G`, so I enter:

```
+8G
```

When it's done, you're back to the fdisk's prompt. Time to create the root partition! Execute the following:

```
n
```

Then, it's even easier: hit `ENTER` for every question fdisk asks. For the size of the partition, it will take by default whatever is left on the disk. When you're back at the prompt, enter the following command:

```
p
```

You should see something like that:

```
Device     Boot     Start       End   Sectors   Size  Type
/dev/sda1              2048      2560       513 256.5K  BIOS  BOOT
/dev/sda2              4096  15628287  15624192   7.5G  Linux filesystem
/dev/sda3          15628288  71567838  55939551  26.7G  Linux filesystem
```

If you have a UEFI boot mode, it will look like this. The difference? The type of the partition for the first partition (the boot partition), here `/dev/sda1`.

```
Device     Boot     Start       End   Sectors   Size  Type
/dev/sda1             2048      2560       513 256.5K  EFI System
/dev/sda2             4096  15628287  15624192   7.5G  Linux filesystem
/dev/sda3         15628288  71567838  55939551  26.7G  Linux filesystem
```

Again, the device name can be different on your system. After all, nothing guarantee we have the same hard disks. The things we have in common: the number of partition created and their types.

It's time to write our changes on the hard disk! Enter the following command to write the partitions:

```
w
```

That's it! You can now quit `fdisk` using the command `q`. When you're back in the shell prompt, you can again run the command `lsblk` to see the newly created partitions.

Formatting the Partitions

The partition table and the partitions themselves are now created, but it's not enough for Arch Linux to use them. Each partition needs to be formatted with a specific filesystem.

Basically, a filesystem group the data on a disk in files and directories. Using the filesystem, the OS can then display the content of a disk as we're used to. Different OS support different filesystems. For example, Windows support primarily the `NTFS` filesystem.

I won't describe every filesystem possible here, it would be too cumbersome and quite boring. We'll simply format our partitions to the filesystems we want, using the tool `mkfs` (`mk` for "make", `fs` for "filesystem").

The UEFI Boot partition

If you have an UEFI, we need first to format the boot partition using the filesystem `FAT32`. Let's run in the shell:

```
mkfs.fat -F32 <your_boot_partition>
```

For example, my boot partition is `/dev/sda1`, so I run: `mkfs.fat -F32 /dev/sda1`.

Root and Swap Filesystems

If you have a BIOS, no need to create a filesystem for your boot partition. It will be done automatically later.

For the Swap partition, you need to run the following commands whatever firmware you have, BIOS or UEFI:

- `mkswap < your_swap_partition >`
- `swapon < your_swap_partition >`

For example, my swap partition is `/dev/sda2`, so I replace `< your_swap_partition >` from the above commands to `/dev/sda2`.

Then, let's create an `Ext4` filesystem for the root partition with the command:

```
mkfs.ext4 <your_root_partition>
```

You root partition is normally the last partition listed when you run `lsblk`. It' as well the biggest partition you have.

Our partitions are now formatted! Glory and joy!

Mounting the Filesystems

If you run `lsblk` again, you'll see an empty `MOUNTPOINT` column. What's that?

For example, if I run `lsblk` on my current system, where Arch Linux is already running, I get this:

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda       8:0    0 465.8G  0 disk
  sda1    8:1    0   200M  0 part /boot
  sda2    8:2    0    16G  0 part [SWAP]
  sda3    8:3    0 389.6G  0 part /
```

A mountpoint is a directory where we can access the filesystem of a precise partition. For example, you can see that the mountpoint of `/dev/sda1` is `/boot`. It means that:

- If I list (read) the files in the directory `/boot`, the files on the partition `/dev/sda1` will be listed.
- If I create (write) a file in the directory `/boot`, the files on the partition `/dev/sda1` will be written.

You've created partitions but you can't access them yet. To do so, you need to mount your partition on a precise directory, too.

Mounting the Root Partition

First, let's mount our root partition on the directory `/mnt`:

```
mount <your_root_partition> /mnt
```

If you run the command `ls /mnt`, you'll see that your partition is totally empty, except for a `lost+found` directory. A Linux based system uses this directory on every partition to recover bits of corrupted data if it needs to.

UEFI and The Boot Partition

If you have an UEFI, you need to mount the boot partition on a new directory. Let's first create this directory:

```
mkdir -p /mnt/boot/efi
```

`mkdir` stands for "make (`mk`) a directory (`dir`)". What's the option `-p` , you might wonder? Try to run `mkdir --help` in your shell. You'll see the argument `-p` and the brief explanation `no error if existing , make parent directories as needed` .

The CLI mkdir only create the directory `efi` by default. If the directory `boot` wouldn't exist, it would return an error. The option `-p` automatically create the directory `efi` and its `p` arent if it doesn't exist.

Another difference: `mkdir` without the option `-p` returns an error if the directory we're trying to create already exists. With the option `-p` , it doesn't. If the directory exist already, nothing happens.

Let's now mount our boot partition on our newly created directory:

```
mount <your_boot_partition> /mnt/boot/efi
```

You can run `lsblk` to see if your mountpoints are correct.

Continuing The Installation

We've now built a comfy nest for Arch Linux to come and make your hard disk its home.

If you want to continue your installation later and stop your computer, you'll have to go through the part Mounting the Filesystems again, after loading the Arch Linux live system from your USB key. For now, we don't have any way to automatically mount them when the computer is starting.

It's not that big of a deal. If your brain is about to explode, make yourself a good cup of tea, do some Yoga, go into the forest singing with the squirrels, and come back when you're ready to install the one, the only, the unique Arch Linux.

In a Nutshell

We'll install in the next chapter the basic packages we need: the Linux Kernel and some basic tools. We learned, in this chapter:

- A prompt invite your to run command lines.
- The Arch Linux live system (like any Linux live system) doesn't run from your hard disk but from your RAM.
- A BIOS is a firmware on your motherboard. Modern computers implement a different firmware called UEFI.
- You can look at your network device with the CLI `ip` .
- You can make requests on a network with the CLI `ping` . Very handy to test your Internet connection!
- An ISO is a file which represents a whole disk (CD-ROM, hard drive...).
- You can copy a disk to another with the command `dd` .
- You can manage your partitions using `fdisk` .
- You can create new filesystems using `mfks` . A filesystem describes the data written on a hard disk in terms of files and directories.

- You need to mount your partitions on directories to access them, using the command `mount`.

Going Deeper

Taking the habit to read the manual of the command you use will bring you an insane amount of knowledge regarding Linux based systems. For example, you can read the description of fdisk by running `man fdisk`.

Don't worry if you don't understand everything. The most important: take the habits to look at the manual. You'll understand more and more what you're reading there as you go through this book. You don't have to read everything; often, only the description is enough at first. Don't put too much pressure on yourself.