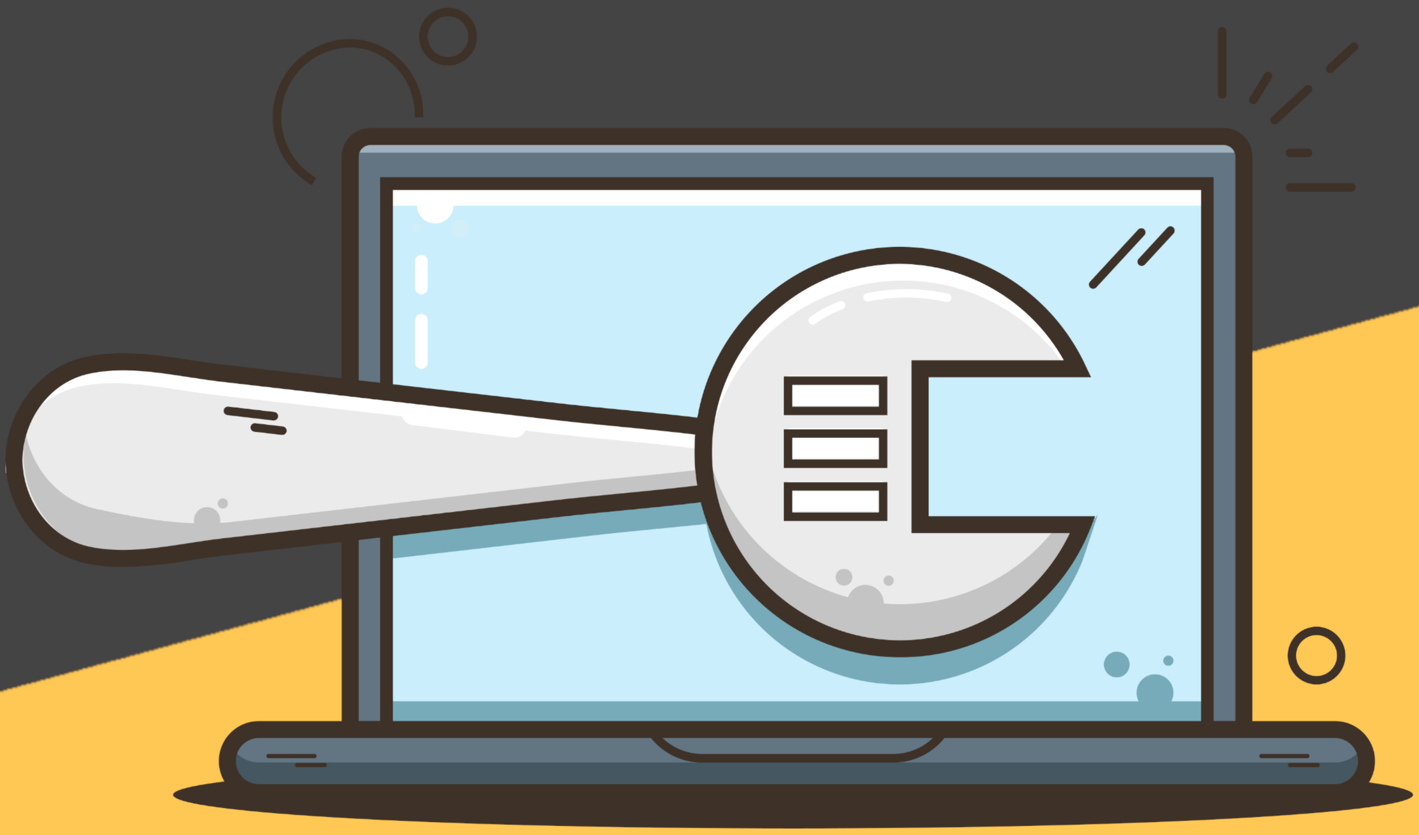
Building your
Mouseless Development Enviroment



Matthieu Cneude

Contents

Introduction

Welcome, Mouseless Developers

With this book we'll embark, together, in a creative journey where we'll build an efficient and Mouseless Development Environment. We'll go from the void of an empty hard disk to a complete system you can mostly use with a keyboard. We'll install everything manually at the beginning, and we'll become an omniscient entity able to summon the whole development environment with one command at the end.

There are huge benefits to install a complete system by ourselves: we'll learn a tonne along the way. It's good to use tools to achieve our goals, but it's even better to know a little about how they work. It will also allow us more flexibility to modify whatever we want according to our needs.

Knowledge brings flexibility, and flexibility brings 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 can use. It's consistent, it doesn't get out of fashion, it doesn't get 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 power. 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 plane of existence (at least), here we go:

1. As developers, we often have to deal with Linux-based systems on servers (or containers) where only the command-line is available.
2. Command-Line Interfaces (CLIs) don't have any graphical interfaces. To use these programs, we have no choice: we need the shell. Keep the acronym "CLI" in mind: we'll use it often throughout the book.
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 the command-line.

This book will explain everything for you to understand what we're doing, why we're doing it, and how you can personalize your system according to *your* needs. 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 any development environment you want, 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 described in this book on macOS.

Now, a bummer: we won't dive deep into everything we'll talk about, or this book would become way too long for us to survive it; both the poor writer and the poor readers. Instead, it will focus on showing how the different tools described in this book can work together.

That said, each chapter of the book concludes on a list of resources you can read to satisfy your infinite curiosity.

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, to experiment, to play with the command-line. This book will be truly rewarding if you put the effort and some patience.

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 sparks your infinite curiosity. Personally, when I discovered it, the spark looked more like an enlightenment! You think I'm exaggerating? Yes, I am. 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 the 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 decades. It's just that the grass is greener and tastier on the Linux side of the fence.

What Is a Mouseless Development Environment?

The obvious goal of a Mouseless Development Environment is to use less of the mouse. It doesn't mean that you can throw your mice 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 dogmatic.

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

I'm sure you noticed but software engineers deal with a lot of text: we spend our time writing code and (hopefully) documentation. It's for this kind of job that our keyboard and our commands really shine. Text never goes out of trend: you can write scripts to parse them and to automate anything you want, thanks to the Holy Shell. Repeat after me: glory to the Holy Shell!

A Mouseless Development Environment lets you keep 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 infinite recursion of pain. Even if you think that it doesn't bother you, you'll see the truth by simply trying not to use it. I thought it wasn't a problem for a long time, but trying to stay on the keyboard definitely changed my way of working.

What Do You Need to Follow Along?

To follow this book, you need to have a *place* to install the whole system and its tools: 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 also use a virtual machine if you want to see first how it looks, or if you just want to follow the book for learning purposes, without the intention to use the final Mouseless Development Environment.

The good news: even if you finish the book on a virtual machine, you'll have a complete installer for the Mouseless Development Environment somewhere on GitHub. 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 anything you want on it, without your physical computer to know about it. Both systems (host and virtual machine) are decoupled as much as possible. In general, 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 recommend 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 focuses 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 go back to these keystrokes if you forget them while using your system. You can of course download ready-made cheatsheets from 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 under too many possibilities.

That's why I would advise you to write your own cheatsheets while going through the book. Write one for each tool. If you need to be convinced, here some arguments:

1. Writing will help you to remember the different keystrokes. You can also 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 memorize. In short, you'll make these keystrokes *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 reasons why I never found Neovim or anything else to have a high learning curve. I was then able to get things done in this mouseless system very quickly. Believe me, it's not because I'm a genius; I'm a very standard human.

Similarly, you should also write down the different commands you'll see 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 passively watching 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 and see what options you can use, for example.

In two words: be curious.

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

Styling Conventions

There are only a few special styling conventions in this book. You'll know, while reading the book, what to execute in the shell, and what to write in what config files.

Sometimes, `<something>` will pop up in some commands you need to execute. That is, an expression surrounded with the characters `<` and `>`. For example, `fdisk <your_hard_disk>`, or `su <user_name>`. 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 to replace them with a good value.

Choose Your Tools

You can replace any tool we'll see in this book with another one or your liking, but I would suggest you to follow the entire book first to understand how they can work together. Then, you can modify any configuration file or replace any 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 task from the start. If you don't want to learn how it works, you can use Nano instead or anything else you prefer.

In a Nutshell

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 different commands and the tools we'll see throughout the book.

In the next chapter, we'll discuss about some important basics about Linux-based systems.

Part I - Arch Linux

A General Linux Overview

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

Let's first see some generalities about the system we'll build. More specifically we'll answer these questions:

- What's Linux?
- What's a Linux distribution?
- What are repositories?
- What's the shell?

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

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 applications you use) 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 the kernel politely 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 *commands* and perform actions depending on them, like running a program or creating a file. You can write these commands yourself using a standard input. The most obvious standard input is a keyboard, but you can use a script too, a file containing commands 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. This means that you need to:

1. Type the command the book provides into a terminal.
2. Press the `ENTER` key to interpret the command.

Then, our friend the shell will return an *exit code*, not directly visible, and possibly an *output* which will be displayed in the terminal by default.

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 anything you want. The sky's the limit.

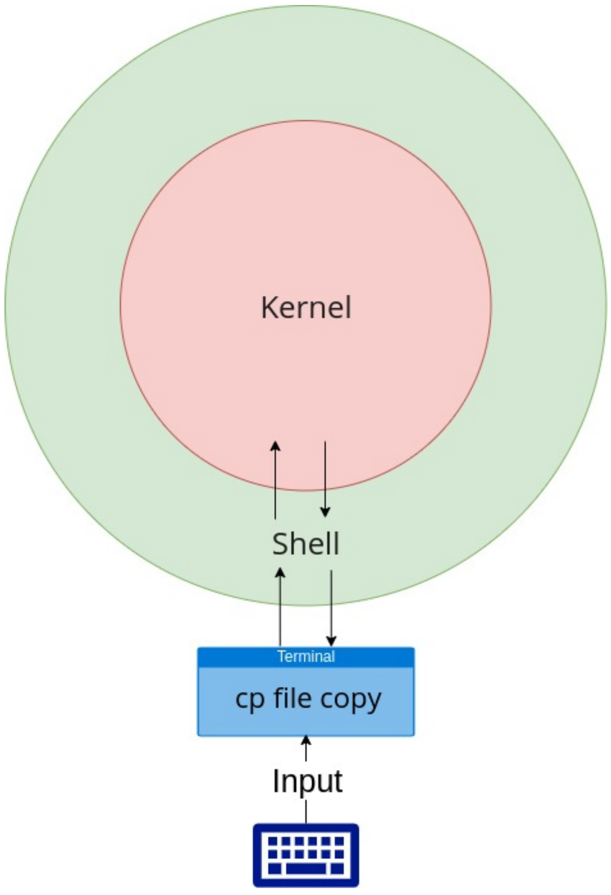


Figure 1: Kernel, shell, and terminal

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 one. A directory to rule them all!

Direct children of the root directory have specific purposes we'll see all along the book.

How does a path to a file (or filepath) in a Linux-based system look? For example: `/etc/zsh`. The first `/` is our powerful root directory, supporting everything else as Atlas supported the world. The directory `etc` is a direct child of the root directory, and the directory `zsh` is itself a child of `etc`. Every `/` which is 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, in a terminal, you run the command `cd /etc`, you'll move from the directory you were to the directory `/etc`. It will become your *working directory*, also known as the *current directory*. Each time you move from directory to directory, your working directory changes 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-based system. Here are two you'll encounter very often:

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

If you run `cd .`, you'll jump 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 of the working directory. Similarly, if you run in a shell `cp ./cat_picture ./file`, you'll `c`o`p`y the file named `cat_picture` in your working directory and "move" it to your working directory. At the end, 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 too. 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 choose what programs and what desktop you want 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 ads in a development environment? Everybody? 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 roughly summarize them as follows:

- The set of applications a Linux distro installs 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 built on top of other ones. For example, Ubuntu is built on top of another distro called Debian; both will have 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 manager. 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 anything 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 also often see 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 individuals, pointing the finger to everybody who doesn't use Arch Linux. They always have a great pleasure to mention 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!

Additionally to the fact fact that installing Arch Linux can teach a great deal about Linux-based systems in general, using it in our daily work has many advantages too.

The Glory of Rolling Distributions

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

Arch Linux is a rolling distribution; the packages are kept 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 Linux 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 other Linux distributions like Ubuntu. My conclusion: Arch Linux is the most stable system I've ever had.

The 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 community 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 provide 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, compile, and update manually the obscure applications you use. An AUR helper can do that for you in one command.

The Fabulous Manual

Almost every CLI out there has at least a `man`ual page describing what it is, how it works, and what it can do. This documentation is often very complete: I advise you to look at it as soon as you need some explanations.

Here's how to access the manual:

```
man <command>
```

For example, you can run `man cd` in a shell to read the manual page of the command `cd`. If you're a beginner, it might be a bit difficult to read at first. When you better understand some concepts distilled in this book, 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 a CLI can do for you. If you're searching for specific *options*, `man` can be very helpful too. A command-line option is 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 often has some kind of option to display some concise help, often something like `--help or -h`. The output is often shorter and more condensed than the man page for the same command. To understand what I mean, try to run:

```
ls --help
```

Manuals are often divided into sections. To look at a specific section, you can run the following:

```
man <section> <command>
```

I won't go into more detail here. If you want to know more about `man` (including this idea of sections), you should read the `man` page of `man` by running in a terminal:

```
man man
```

It's inception directly in your shell!

Troubleshooting

General Recommendations

Manually installing 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 something related to Linux, you'll very often end up on it.
2. The official Arch Linux website is the place to go when you have problems updating your system. It describes the manual interventions you need to do from time to time. 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.

Using VMWare Software

You'll need to install additional packages if you run this system on any VMWare product. I won't go into the details here, but you can look at this page on the Arch Wiki if you have any problem down the road.

In a Nutshell

What did we learn in this chapter?

- The kernel is the heart of an Operating System (OS). You can communicate with the kernel using the shell, which will read and interprets commands you type into a terminal emulator.
- In the Linux filesystem, every directory is a child of the root directory, named `/`. A filepath looks like this: `/etc/kernel`, where the first `/` is the root directory and the subsequent `/` are directory separators.
- 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 different philosophies on handling packages, different package managers, and different repositories. They come bundled with different applications, too.

The next chapter will be a short but important one: we'll see how to have good typing techniques. This is mandatory if you really want to be efficient with a keyboard.

Going Deeper

- Arch Wiki
- Arch Wiki - Arch Linux

- Arch Linux Website
- Arch Linux Forum
- Manual pages for Arch Linux packages

The Power Is In Your Fingers

You remember Frodo? He saved Middle Earth by throwing a ring into a volcano. He was quite small, but how important!

This chapter is quite small too, and like Frodo, very important.

You're reading right now a book to build a Mouseless Development Environment, so it's pretty obvious I'll advocate to use your keyboard as much as you can. The goal is simple: you should move your hands as little as possible from this magical device.

Knowing how to type properly is a stepping stone to master a keyboard driven workflow. You can practice the following techniques while reading the book. When you begin to replace your typing techniques, stick to it as much as you can. Don't come back to your old (and bad) habits.

The benefits are huge. Massive. Colossal!

- You'll feel much more comfy while developing the next trendy application which will make you rich. What a secret pleasure to have our hands on the keyboard!
- The room for progression (speed and accuracy) will increase drastically.

If you look at pianists playing, you'll notice that they never look at their hands. They don't want to spend brain power on those kinds of detail; their hands follow their will as fast as possible.

The analogy holds very well for us, computer addicts. When you are in a state of flow, focused on your craft, you don't want to be interrupted every five minutes by moving your hand to the mouse or by looking at your fingers. You don't want to think about all of that. You want to create, nothing else.

Let's see how we can achieve that.

Efficient Typing: The Two Rules

It's satisfying to see your typing techniques improving day after days, months after months, and even years after years. These techniques are easy to learn but difficult to master.

The first rule you need to be aware of: placing your hands correctly. The keys `a`, `s`, `d`, `f` and `j`, `k`, `l`, a `;` are called the *row keys*. They are the starting points for your hands. From there, you'll be able to grab any other key as efficiently as possible.

You'll notice that there are little bumps on the keys `f` and `j` on your keyboard: they are indicators for you to know where you need to put your indexes. When they are at the good position, simply place the other fingers on the other row keys.

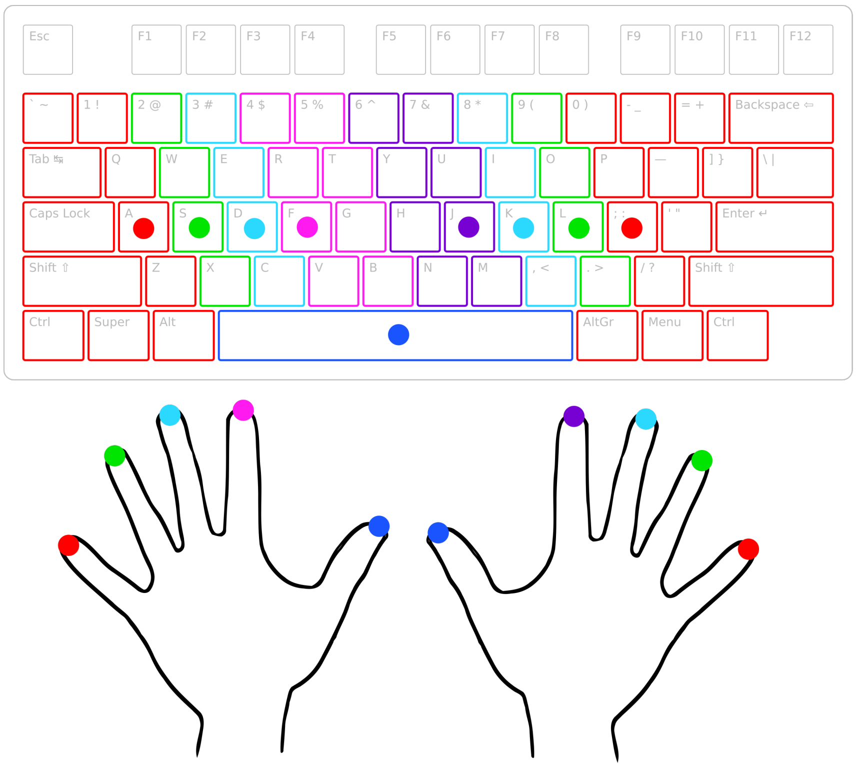


Figure 2: Correct hand placements - colorblind version

The second rule you need to train for: try not looking at your keyboard while you're typing. Of course, if you don't remember where a key is, look at it, but only after trying blindly where you think it is. We want to train your muscle memory here.

I was only typing with two fingers before trying to follow these two rules. It felt really weird to use these new techniques at first; now, I wouldn't type differently. It's efficient, it's comfortable, it's great!

The First Week

When you decide to use the two rules we saw above, you need to try to follow them *all the time*. We need 100% commitment here. If you surprise yourself using your bad techniques again, which will happen, don't worry: simply come back to the good ones. This is part of the learning process, not a horrible failure cursing your whole family on five generations.

The first three days are the most difficult. You'll alternate between good and bad technique without even noticing it. You'll do mistakes. You'll be slower. That's great! It's how you'll learn.

Fortunately, at the end of the week, the amount of mistakes you'll make will decrease, and the need to watch the keyboard will slowly disappear.

The Second Week

You'll notice during the second week the amount of mistakes decreasing even more, and you won't dare look at your keyboard while typing. At the end of the week, you'll see your typing speed improving already. The good feelings of reward will begin to please your brain. That's what we all want.

Speed and Accuracy

During your two weeks of initial training, you shouldn't focus on speed or accuracy. Just type, as much as you can, and don't worry about anything else yet. Not even about the mistakes you're making.

Only then, when you feel comfortable enough, you can shift your focus on speed and accuracy: how fast you can type while making as few mistakes as possible.

In A Nutshell

What did we learn in this chapter?

- We need to focus on our work, not on our hands. Learning the good typing techniques will help you tremendously in this regard.
- The first rule: place your hands correctly on your keyboard, using the row keys.
- The second rule: don't look at your hands while typing. Even if you don't remember where a key is, don't look at it before trying to get it without looking.
- Only train for speed and accuracy when you're used to these new typing techniques.

In the next chapter, we'll begin to prepare our hard disk for installing Arch Linux.

Going Deeper

As always, to learn as fast as possible, you need to practice. This book ask you to type many commands, so you'll have many occasions to learn these good typing techniques.

You can also use typing software to have concrete data about your speed and accuracy. Here are my favorites:

- Type Racer
- Online Typing Test WPM
- Speed Coder

Preparing Your System for Arch Linux

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

- 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 very 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 20 GB to feel comfy).
3. A USB key (at least 1 GB).
4. A second computer, a tablet, or any other 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 mentioned already, the computer can be a virtual machine or a physical one.

Burning the 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 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, 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 the command `CertUtil -hashfile < path_to_ISO_file > MD5`.
    - On Linux, you can use the command `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. Change the boot order to read USB devices before your hard disk. You can do that by using your BIOS interface.
5. Plug your USB key and start your computer.

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

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

If you did everything correctly, your computer will boot from 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" messages telling you what systemd is starting. We'll talk about systemd later in this book. Then, you'll end up in a shell prompt. It will look like this:

```
root@archiso ~ #
```

A prompt is a set of characters indicating that you can execute commands. Here, it indicates what user you are (`root`) and the hostname `archiso`. The hostname is the name of your system as it will appear on a network.

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

It also means that everything you're doing in the live system won't survive if you shut down or restart your computer, except if you deliberately write 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, too. 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 the command history:

    - You can quickly access executed commands 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, you can use the keystroke `CTRL+c`. Be careful with it: patience is sometimes safer. Think about what your command is doing before interrupting it, and consider 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 has almost all power a user can have. 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, even if they have both root as their name.

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
```

You can then 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 the 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 even be 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 it 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 more than likely we won't have the same network devices either.

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. It works similarly to the shell prompt you had before. 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 that 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 requests 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, you're connected to the infinite Internet! Lucky you.

If it doesn't work, try:

```
ping -c 5 185.183.156.38
```

If this last command works, this 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 protocols on the Internet. Anxious? Don't worry, Internet is the only stuff which never breaks in the computing world. Not like enterprise Java code.

Run this command:

```
timedatectl set-ntp true
```

We synchronized our system clock with the Network Time Protocol. 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 some *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 piece of software running when you're booting (starting) your computer.

This BIOS verifies that your hardware works properly. It also makes available an interface for you to configure some basics, like the order of the booting devices. It will also 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 types of firmware, but confusion arises when the UEFI pretends to be a BIOS, sometimes using the term “BIOS” on its own interface! It’s often an ironic attempt not to confuse people who are used to having a BIOS.

Long story short, because the firmware of your motherboard starts the bootloader of Arch Linux, we need to know if you have a BIOS or a 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`, this means that your computer has a BIOS. If it outputs a bunch of files, you have a 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 later to create our boot partition and install the bootloader itself.

We’ve just learned the first piece of software which runs 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:

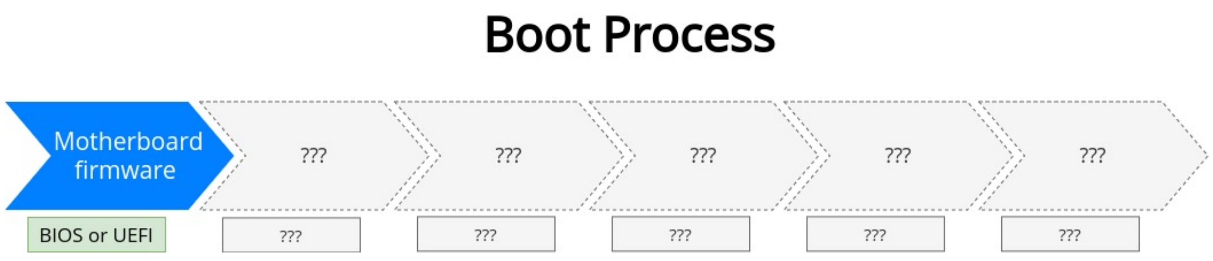


Figure 3: The boot process: BIOS and UEFI

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

Partitioning the Hard Disk

Enough explanations. Let’s do some serious stuff now. First, let’s run the following in the shell:

```
lsblk
```

This command will display your *block devices*. A block device is a file which represents 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, the applications running on your system, 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 your 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 to read a program or write to a device, like your hard disk: the answer will always be via a file.

For example, you can try to run the following in your shell:

```
cat /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 (your CPU in that case).

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

- `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. It can take a long time depending on the size and the type (mechanical or SSD) of the hard disk. As an example, for a mechanical disk of 1 GB, it took me 4 hours to entirely wipe it.

If you're good with that, run the following in your shell:

```
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`, the name of my hard disk is `/dev/sda`. As a result, to wipe it, I need to run `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 will wipe out everything you give. You don't want to mess up with `dd`!

The command copies everything from the file `/dev/zero` to any file representing your hard disk (for me, `/dev/sda)`. The file `/dev/zero` is a file containing an infinity of zeros; new zeros will be created each time you try to read from it. In our precise example, your hard disk will be filled with zeros 'till it's full. At this point, the command `dd` will stop and your disk will be "clean".

You can also use `/dev/urandom` instead of `/dev/zero`: it will fill your hard disk with random numbers instead of zeros. It takes even more time, so use this solution only if you don't want anybody to be able to restore the removed data of your hard disk.

Using fdisk

It's 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. The horrible truth is: `fdisk` is, in fact, quite easy to use.

First, we need a *partition table*. This table is 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 OS) will be installed.
2. A *swap partition*, a place on your hard disk used as memory when your RAM is full. It's not perfect since the RAM is 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.

Don't confuse the root partition with the root directory or the root user we saw earlier. These concepts are different.

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

```
fdisk <device_name>
```

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 will appear. It will look like this:

```
Command (m for help):
```

You can execute special one-letter commands in fdisk. Nothing will be written on your hard disk when doing so, except if you use the command `w` (for `w`rite). So if you do a mistake, don't panic: you can abort everything with `CTRL+c` if you didn't write anything yet.

Let's now create a GPT partition table. Let's execute the following command in fdisk's 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 this point, `fdisk` will ask you the partition number. The default is normally `1`, so just hit `ENTER`.

As CLIs go, fdisk is very curious. It loves asking questions. It will then ask you 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 you want for your partition here. For the boot partition, 512 MB 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):
```

Now, we need 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 use the command `L`. To quit the list, use `q`.

UEFI Boot Mode

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

```
1
```

Here's how the whole operation should look:

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

```
4
```